Designing Low Latency Continuous Queries in Stream Processing Systems

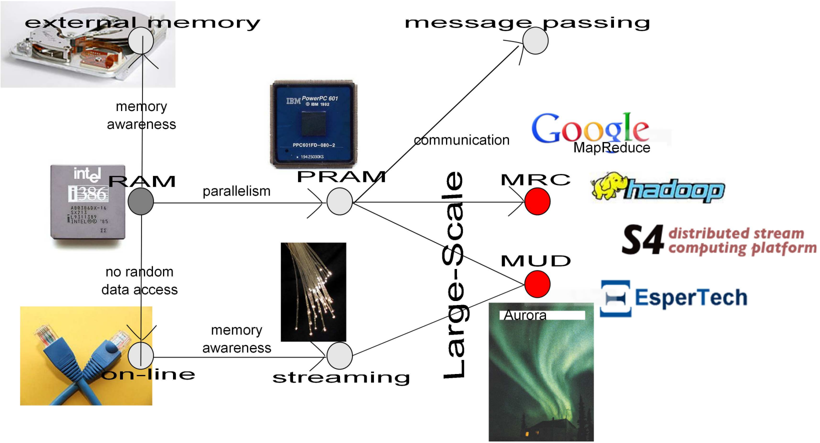
Winter School: ''Hot Topics in Secure and Dependable Computing for Critical Infrastructures''

Donatella Firmani°

° Sapienza, University of Rome
Dipartimento di Ingegneria Informatica, Automatica e Gestionale
"A. Ruberti"

17 January 2012

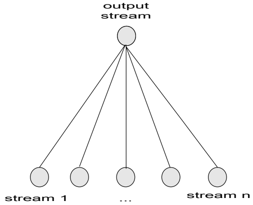
Evolution of Systems and Models of Computation I



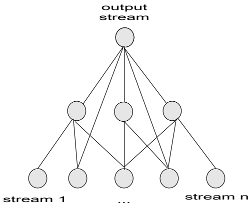
Evolution of Systems and Models of Computation II

Continuous Distributed Monitoring

- Given a set $\mathcal{S}$ of n streams (of items, evtents, etc.)
- Given a property p defined over $\mathcal{S}$
- When the property p “happens”, alert the user $\cdots$
- $\cdots$ as soon as the property p happens



MUD model



unfolding query $\longrightarrow$ hierarchies

Pros of a Model of Computation I

What a model of computation is not

- profile tool $\rightarrow$ it cannot assess performances of a code fragment
- simulation tool $\rightarrow$ it cannot forecast how many seconds a code fragment will take

What a model of computation should do

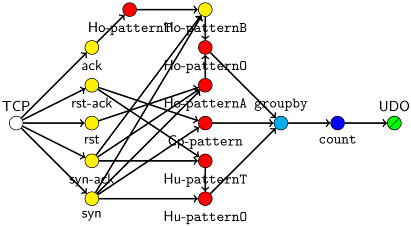
Provide a way to evaluate an algorithm independently from its implementation / deployment on the real system that it models

Pros of a Model of Computation II

```
function insertionSort(array A)
i ← 1
for i < length[A] do
    value ← A[i]
    j ← i − 1
    while j ≥ 0 and A[j] > value
    do
        A[j + 1] ← A[j]
        j ← j − 1
        A[j + 1] ← value
    end while
end for
```

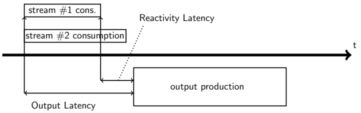
```
function quickSort(array A)
n ← length[A]
if n < 1 then
    return
else
    p ← random element ∈ A
    A1 ← elements ∈ A ≤ p
    A2 ← elements ∈ A > p
    quickSort(A1)
    quickSort(A2)
    merge(A1, A2)
end if
```

Pros of a Model of Computation III



Problem Statement

Time *from* the occurrence of the monitored property *to* the update of the output stream



Latency

- Find a significative abstraction of the system
- Find a metric that models the latency of the continuous query

- Results, Work in Progress and Open Issues . . .

Introduction

Continuous Query Model

Low Latency Query Design

Data-Flow Graph

- **EPU**. An Event Processing Unit is a function that takes streams as input and originates a single stream as output for downstream consumption.
  - a relational operator (e.g., Esper);
  - any user-defined operator (e.g., Spade).
- **DFG**. A data-flow graph is a DAG $G = (V, E)$ s.t.
  - V contains all the EPU nodes needed for the computation;
  - in E there exists an edge (v, u) iff there exists an EPU $v \in V$ that produces an event stream which is consumed by an EPU $u \in V$.

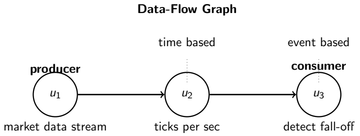
Data-Flow Graph Example: *market data feed*

EPU	operation

u_1	`String symbol;`
	`FeedEnum feed;`
	`double bidPrice;`
	`double askPrice;`

u_2	`insert into TicksPerSecond`
	`select feed, count(*) as cnt`
	`from MarketDataEvent.win:time_batch(1 second)`
	`group by feed`

u_3	`select feed, avg(cnt) as avgCnt, cnt as feedCnt`
	`from TicksPerSecond.win:time(10 seconds)`
	`group by feed`
	`having cnt < avg(cnt) * 0.75`

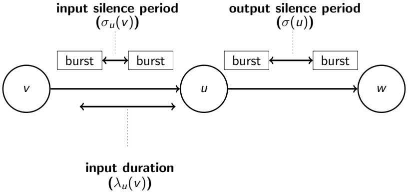


Query: Process a raw *market data feed* and detect when the data rate of a feed falls off unexpectedly, in order to alert when there is a possible problem with the feed.

Model Abstraction I

Let a *burst* be a continuous sequence of events. During the execution of a continuous query, bursts and silence periods happen: an EPU *updates* the output stream by producing a burst, and then a silence period follows.

Bursts and silence periods can either be propagated from an EPU u to the consumer or disappear during the computation.



Evaluation of *DFG metrics* is performed on the basis of EPU bursts consumption and bursts / silence periods production.

Model Abstraction II

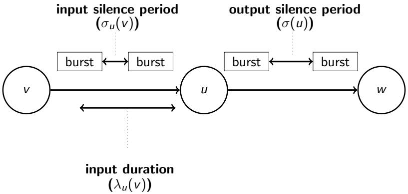
EPU u behavior, or "modes":

- **ASB/O** All-Streams Batch/Online Processing (e.g., logical `and`/`or`)
- **EB/TB** Event/Time Based (e.g., `detect fall-off`/`ticks per sec`)

EPU u parameters:

- input size producing an output update:
  - TB $\rightarrow t_u(v)$. time window w.r.t. output stream produced by v
  - EB $\rightarrow n_u(v)$. # events w.r.t. output stream produced by v
- output update length: $n(u)$
- time in which u computes the function (and update the output stream): $p(u)$.

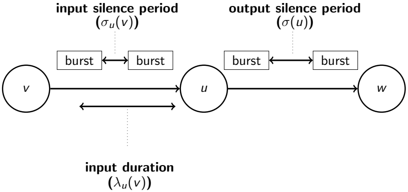
EPU Input Silence Period



- input silence period

$$\sigma_u(v) = \begin{cases} \sigma(v) & \text{if } u \text{ is EB} \wedge n_u(v) \bmod n(v) = 0 \\ 0 & \text{otherwise, e.g., } u \text{ is TB} \end{cases}$$

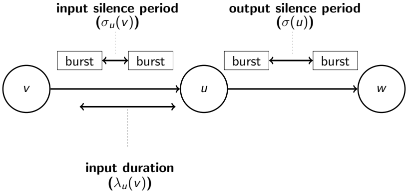
EPU Output Silence Period



- output silence period

$$
\begin{aligned}
\sigma(u) &= p(u) + \sigma_u(\tilde{v}) \\
\tilde{v} &= \begin{cases} \underset{v \in I(u)}{\operatorname{argmax}} \; \lambda_u(v) & \text{if } u \text{ ASB} \\ \underset{v \in I(u)}{\operatorname{argmin}} \; \lambda_u(v) & \text{otherwise} \end{cases}
\end{aligned}
$$

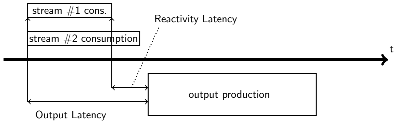
EPU Input Duration



- input duration producing an output update

$$\lambda_u(v) = \begin{cases} n_u(v) + \sigma(v)(\frac{n_u(v)}{n(v)} - 1) & \text{if } u \text{ is EB} \\ t_u(v) & \text{otherwise} \end{cases}$$

Data-Flow Graph Metrics



Given a data-flow graph G and a set of input streams S that produces an output stream update, compute:

- **Output Lat**: begin of the input $\rightarrow$ begin of the output update
- **Complexity**: event consumption period producing an output update
- **Reactivity Lat:**
  event triggering output update $\rightarrow$ begin of the output update

Metric proposal to model continuous query latency: Reactivity.

Latency Evaluation

Computation of Output Latency and Complexity of a DFG G

- compute $\sigma_u(*)$, $\sigma(u)$ and $\lambda_u(*)$ for each u (use a topological sort of G)
- execute the OL (resp. C) algorithm
  it consists a graph visit that finds the OL-critical path (resp. C), i.e., the set of EPUs determining its final value

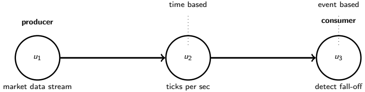
Definition of Reactivity Latency:

$$RL(G) = OL(G) - C(G) \tag{1}$$

OL(G) DFG G Output Latency, **C(G)** DFG Complexity

Latency Analysis Example: *market data feed*

- "x" variables depend on the semantic of the input
- "y" variables do not
- **compute mdf reactivity**



	$I(u)$	$n_u(*)$	$t_u(*)$	$n(u)$	$p(u)$
u_1	$\emptyset$	$<>$	¡¿	1	y_1
u_2	$\{u_1\}$	-	$< 1 >$	1	y_2
u_3	$\{u_2\}$	$<x_3>$ $x_3 \in [1, \infty)$	-	1	y_3

	$\sigma_u(*)$	$\sigma(u)$	$\lambda_u(*)$
u_1	$< 0 >$	y_1	$< 0 >$
u_2	$< 0 >$	y_2	$< 1 >$
u_3	$< y_2 >$	$y_2 + y_3$	$< x_3 + y_2(x_3 - 1) >$

EPU metrics:

	$C(u)$	$OL(u)$	$RL(u)$
u_1	1	y_1	y_1
u_2	1	$y_2 + 1$	y_2
u_3	$x_3 + y_2(x_3 - 1)$	$x_3 + y_2(x_3 - 1) + y_3$	y_3

Reactivity Analysis in *market data feed* I

```
for all u in V do
    // Initialization.
    if u.isASB() then
        outputlat_to[u]= 0;
    else
        outputlat_to[u]= ∞;
    end if
end for
for all v in topological_sort(G) do
    for all u s.t. v ∈ I(u) do
        // Weight of the edge.
        weight_vu =OL(v);
        // Does v belong to the OL-critical path?
        if (u.isASB() ∧ outputlat_to[u] ≤ outputlat_to[v] + weight_vu) ∨ (u.isASO() ∧ outputlat_to[u] ≥
        outputlat_to[v] + weight_vu) then
            // Edge contribution.
            outputlat_to[u] = outputlat_to[v] + weight_vu;
        end if
    end for
end for
return outputlat_to[c] + OL(c);
```

(1) Output Latency evaluation. $y_1 + y_2 + 1 + x_3 + y_2(x3 - 1) + y_3$
Dependency from input stream due to x_3.

Reactivity Analysis in *market data feed* II

```
for all u in V do
    // Initialization.
    complexity_to[u]= 1;
end for
for all v in topological_sort(G) do
    for all u s.t. v ∈ I(u) do
        // Weight of the edge.
        weight_vu = C(u)/n(v);
        // Does v belong to the C-critical path?
        if complexity_to[u] ≤ complexity_to[v] · weight_vu then
            // Edge Contribution.
            complexity_to[u] = complexity_to[v] · weight_vu;
        end if
    end for
end for
return complexity_to[c];
```

(2) Complexity evaluation. $x_3 + y_2(x_3 - 1)$
Dependency from input stream due to x_3.

Reactivity Analysis in *market data feed* III

(3) Apply Reactivity Latency definition.

$$
\begin{aligned}
OL(G) - C(G) &= (y_1 + y_2 + 1 + x_3 + y_2(x3 - 1) + y_3) \\
&- (x_3 + y_2(x3 - 1)) \\
&= y_1 + y_2 + 1 + y_3
\end{aligned}
$$

- In mdf no dependency from input stream!
- Formal proof still missing...

Introduction

Continuous Query Model

Low Latency Query Design

CQMTool

Software tool for Metrics evaluation

- written in Python
- symbolic calculus with SymPy

Input XML file

```
<query>
 <epu>
  <name> MarketDat </name>
  <mode> ASB, EB </mode>
  <max_ou> 1 </max_ou>
  <proc_t> y1 </proc_t>
 </epu>
 <epu>
  <name> TicksPerS </name>
  [...]
</query>
```

Output XML file

```
[...]
<output_lat>
 x3+y1+y2+y3+ (y2+1.0)*(ceil(x3)-1) +1.0
</output_lat>
[...]
[...]
[...]
<reactivity_lat>
 x3+y1+y2+y3+ (y2+1.0)*(ceil(x3)-1) +1.0
</reactivity_lat>
[...]
```

Target Application I

Real case study

$\boxed{\text{Distributed Half Open port scan detection } \textit{problem}}$

The scanner S sends a SYN packet to a target T on port P:

- SYN-ACK received: P is open
- RST-ACK received: P is closed
- no packet and T reachable: P is filtered
- otherwise: unknown state of P

Target Application II

Design of a continuous query: *Line Fitting* [Aniello et al. [Ani+11]]

- implemented using the CEP engine Esper
- two data flow graphs are used to represent Line Fitting
- their performances are evaluated through the tool

Line Fitting Analysis I

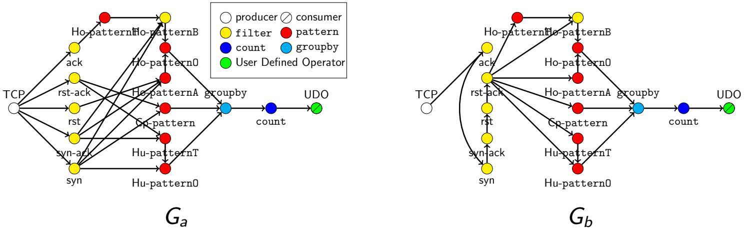
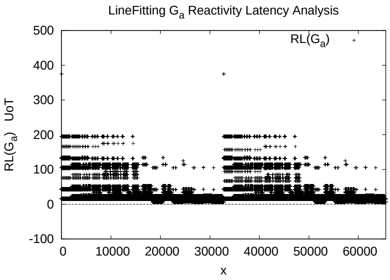


Figure: The data-flow graphs for Line Fitting, namely G_a and G_b.

- G_a represents actual Line Fitting implementation
- G_b represents a different solution, to evaluate `CQMTool`
- up to 30 parameters involved

Line Fitting Analysis II



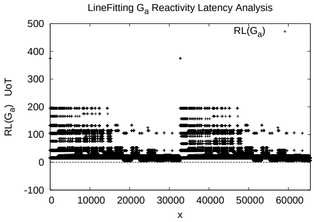
Data Representation $\rightarrow$

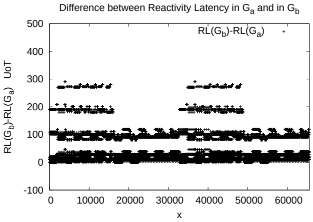
metric $M(G) : X \rightarrow \mathbf{R}$

- X: N-dimensional
- $X_{d,b} \subseteq X$: 0-dimensional space containing only the vectors in X s.t. $\forall i x_i$ equal to b^{d_i} (d *sampling factor*, with $0 \leq d_i < d$, and b *base*)
- $M_{d,b}(G) : X_{d,b} \rightarrow \mathbf{R}$ can be easily analyzed!

$$\boxed{d = 2,\ b = 10 \text{ and } d^N = 65535 \text{ points}}$$

Line Fitting Analysis III





G_a is more *reactive* than $RL(G_b)$: quantify the performance gain

- $RL(G_b) - RL(G_a)$ it is never negative: $RL(G_a) < RL(G_b)$ in 65535 points

RL difference	points
$\in [300, 200]$	1%
$\in (200, 100]$	4%
$\in (100, 50]$	10%
$\in (50, 0]$	remaining

THANK YOU.

$\ddot\smile$