Linked Stream Data Processing Engines – Facts and Figures

Danh Le-Phuoc*, Minh Dao-Tran, Minh-Duc Pham, Peter Boncz, Thomas Eiter and Michael Fink

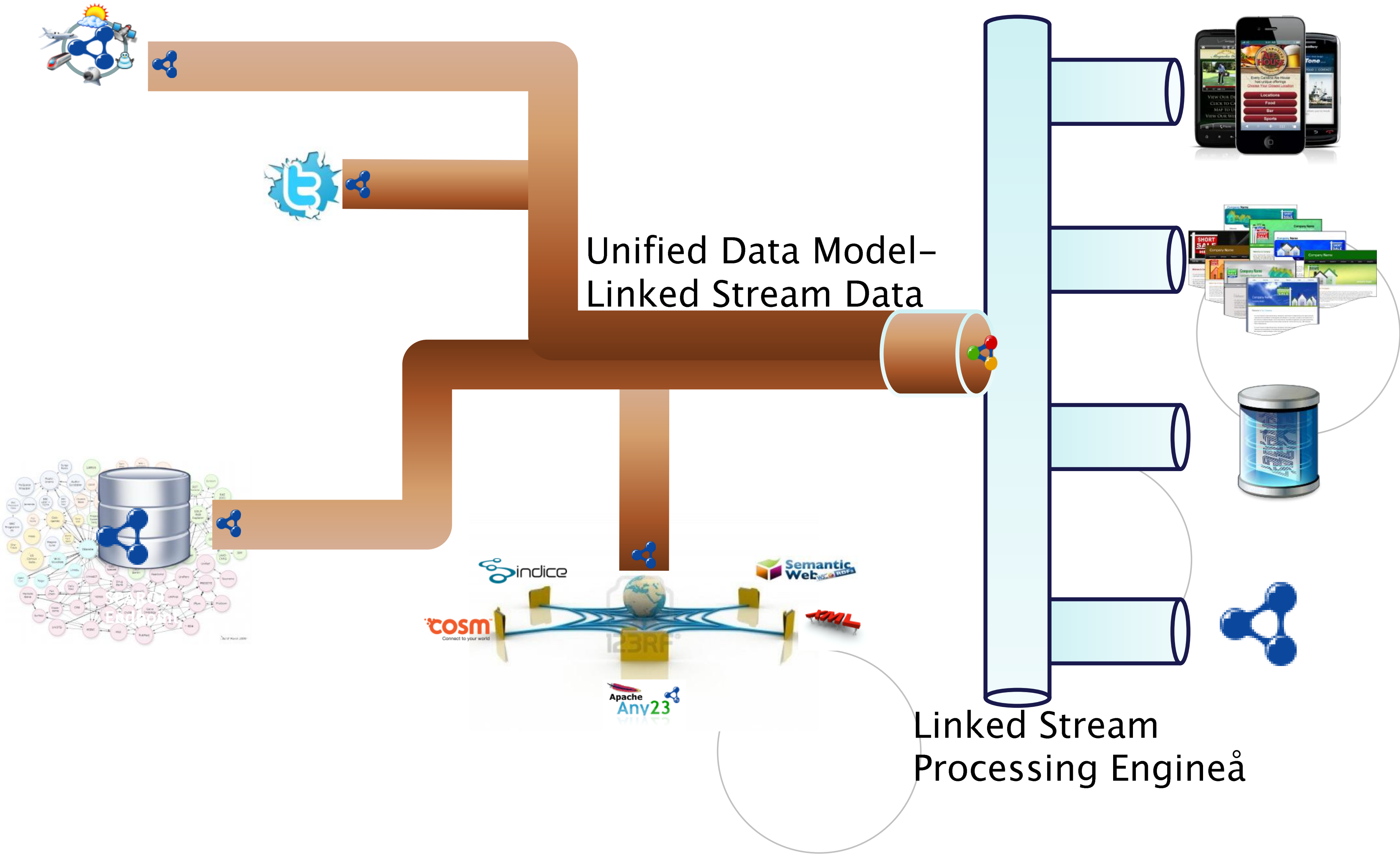
*DERI – National University of Ireland, Galway

ISWC 2012, Boston

Outline

- Motivation for Linked Stream Data Processing
- State-Of-The-Art Linked Stream Processing Engines
- Challenges of cross-system evaluation
- Evaluation framework
- Experiments
- Findings
- Summary

Motivation for Linked Stream Data Processing



Unified Data Model– Linked Stream Data

Linked Stream Processing Engineå

State-Of-The-Art Linked Stream Processing Engines

- State-Of-The-Art Engines
  - □ Streaming SPARQL (2008)
  - □ C-SPARQL (2010)
  - □ SPARQL$_{\text{stream}}$ (2010)
  - □ ETALIS (EP-SPARQL, 2010)
  - □ CQELS(2011)
  - □ Sparkware(2012)

- Too few cross-system comparisons
  - □ CQELS(2011)
  - □ Sparkwave(2012)

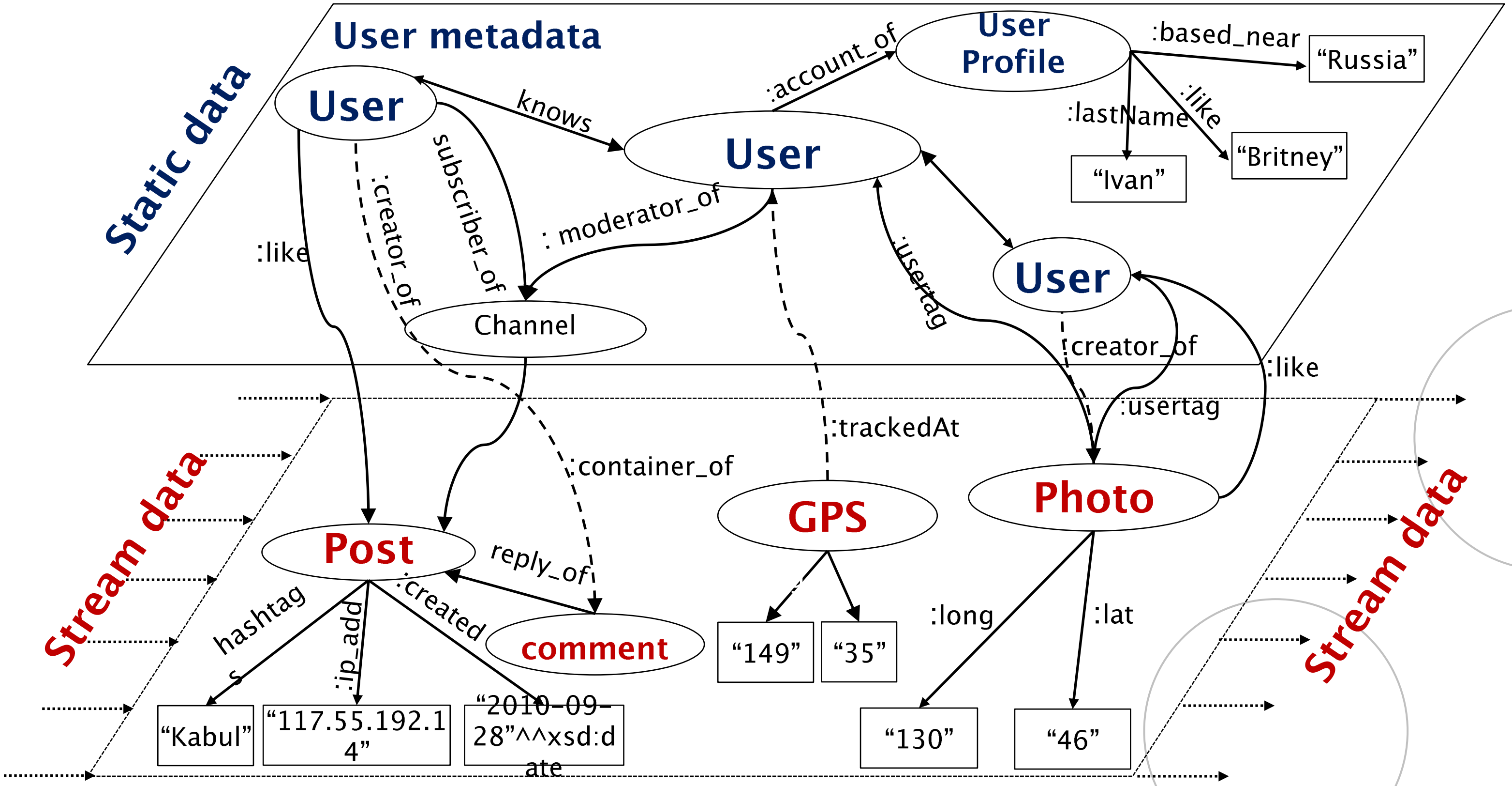
Challenges in cross-comparison of Linked Data Stream Processing engines

- Differences in query semantics
- Differences in execution mechanisms
- Running environment and experiment parameters can lead to different outputs in continual execution context.

Evaluation framework

- Test suits:
  - □ Functionality tests : query patterns supported
  - □ Correctness tests : outputs are comparable to for cross-comparisons
  - □ Performance tests : compare throughputs that meet comparability conditions
- Evaluation toolkit (Linked Stream Benchmark-LSBench, http://code.google.com/p/lsbench/) :
  - □ Data generator : Stream Social network data Generator **(S2Gen)**
  - □ Test drivers and Evaluators

Data schema



- Simulates data schema in social networks
  - □ Stream data: GPS, Posts & comments, Photos
  - □ Static data: User metadata (user profile, users' relationships)

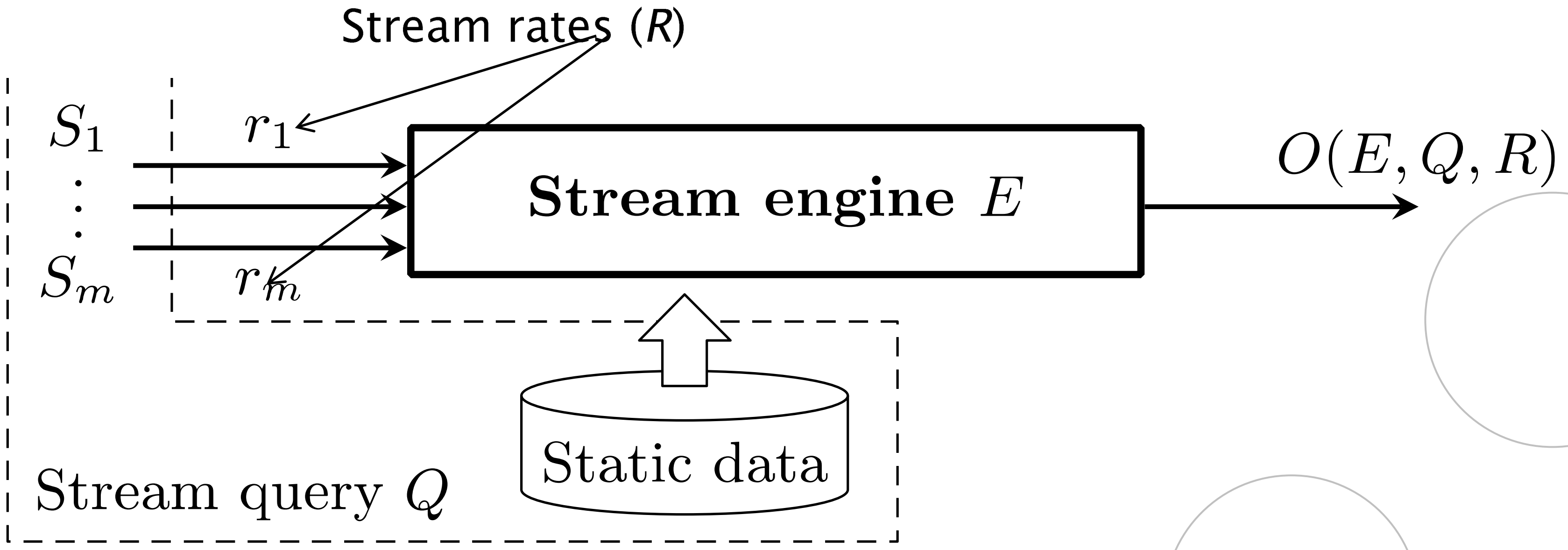
Data generator (S2Gen)

- A novel data generator for Linked Stream Data
- Models realistic data correlations and skewed data distributions in stream-based social networks
  - □ Use *window sliding* approach for generating social data streams
- Offers various parameters for different test cases
  - □ Scalability
  - □ Different stream input rates
  - □ Different correlation probabilities

Experiments

- Experiment Design
- Test runs
  - Functionality tests
  - Correctness tests
  - Performance tests
    - Execution throughput
    - Scalability tests

Experiment Design



- Engine as a black box
- Data streams via stream players
- Record output stream

Functionality tests

- None of the engines support all the SPARQL query patterns

	Patterns covered							S	N_P	N_S	Engines		
	F	**J**	**A**	**E**	**N**	**U**	**T**				CQ	CS	JT
Q_1	✓								1	1	✓	✓	✓
Q_2		✓						✓	2	1	✓	✓	✓
Q_3		✓						✓	3	1	✓	✓	✓
Q_4	✓	✓							4	1	✓	✓	✓
Q_5		✓						✓	3	2	✓	✓	∅
Q_6		✓						✓	4	2	✓	✓	∅

	Patterns covered							S	N_P	N_S	Engines		
	F	**J**	**A**	**E**	**N**	**U**	**T**				CQ	CS	JT
Q_7	✓	✓						✓	7	2	✓	Ⓢ	∅
Q_8		✓			✓				3	2	×	Ⓢ	∅
Q_9	✓	✓				✓		✓	8	4	✓	E	∅
Q_{10}			✓						1	1	✓	✓	✓
Q_{11}		✓	✓	✓					2	2	×	✓	×
Q_{12}			✓				✓		1	1	×	✓	×

F: filter **J**: join **E**: nested query **N**: negation **T**: top k **U**: union **A**: aggregation **S**: uses static data

N_P: number of patterns, N_S: number of streams, Ⓢ: syntax error, E: error, ∅: return no answer, ×: not supported

CQ: CQELS, CS: C-SPARQL, JT: JTALIS

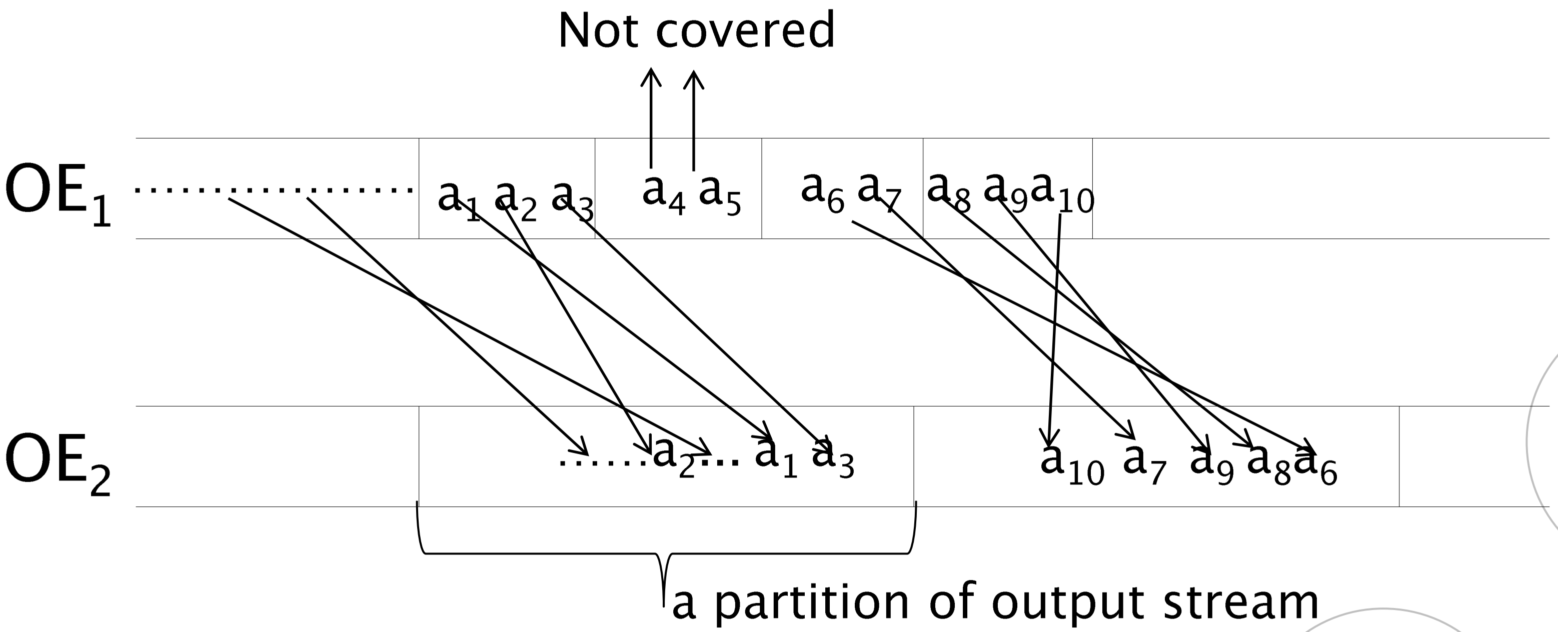
Correctness tests

- All the engines don’t output the same numbers of results

	Rate :100 inputs/sec			Rate : 1000 inputs/sec		
	CQELS	C-SPARQL	JTALIS	CQELS	C-SPARQL	JITALIS
Q1	68	604	68	68	662	68
Q2	68	124	68	68	123	68
Q3	533	1065	533	533	1605	533
Q4	11984	125910	1442	11945	127026	4462
Q10	28021	205986	28021	28021	209916	28021

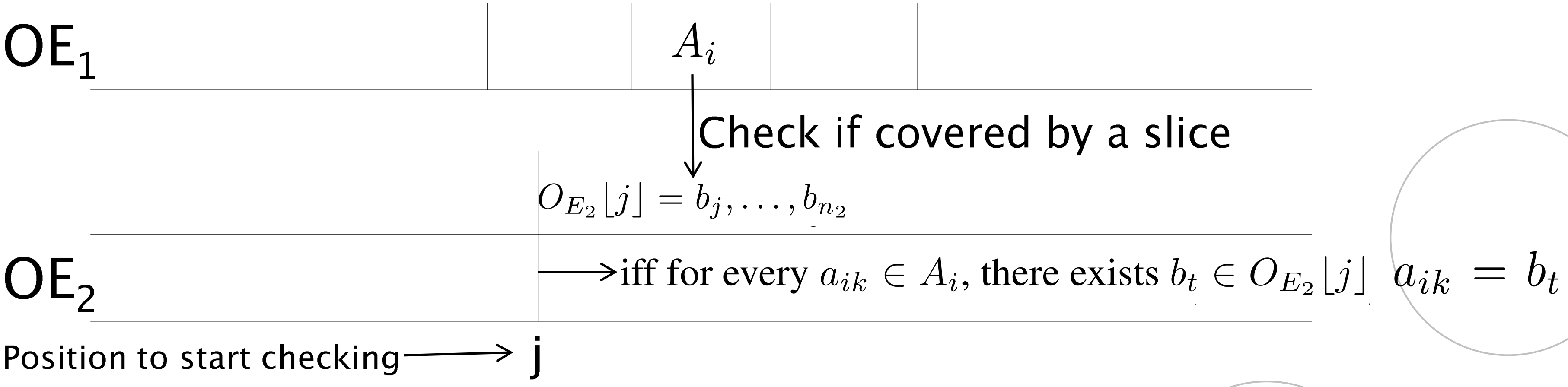
☛ Comparisons on input/output throughputs are invalid

Output stream coverage



- Not full ordered output streams but the output partitions might be
- A partition of OE_1 can be covered by groups of elements from OE_2
- The output with the greater number of outputs might not cover the one with the smaller number.

Coverage checking for a output partition



OE$_1$

A_i

Check if covered by a slice

$O_{E_2}\lfloor j \rfloor = b_j, \ldots, b_{n_2}$

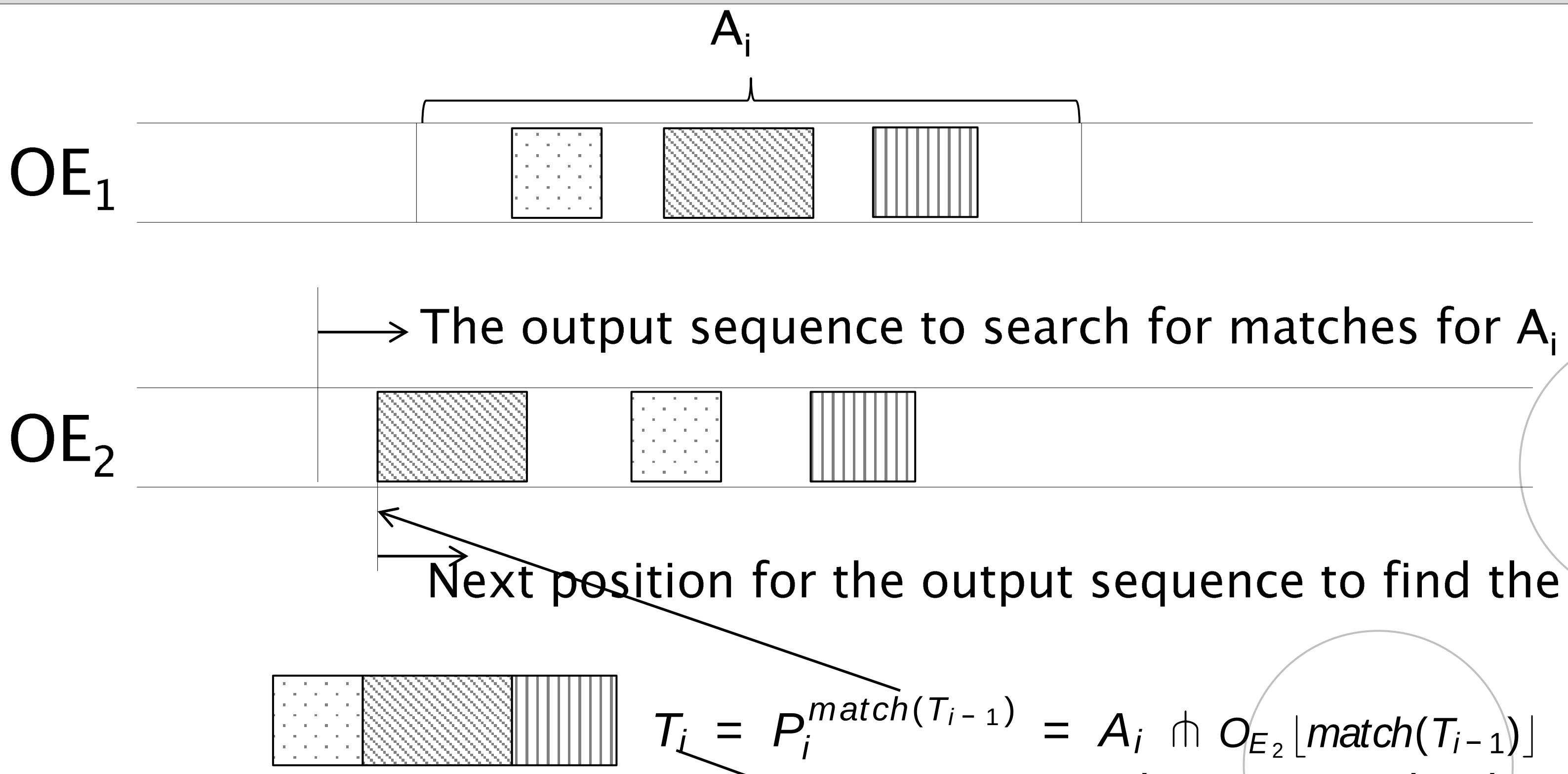
OE$_2$

→iff for every $a_{ik} \in A_i$, there exists $b_t \in O_{E_2}\lfloor j \rfloor$ $a_{ik} = b_t$

Position to start checking → j

- Slice the stream output to output sequences to check if a output partition is covered by a sequence
- If not covered ➔compute the mismatch metric

Mismatch metric for non-coverage



A_i

OE_1

The output sequence to search for matches for A_i

OE_2

Next position for the output sequence to find the matches

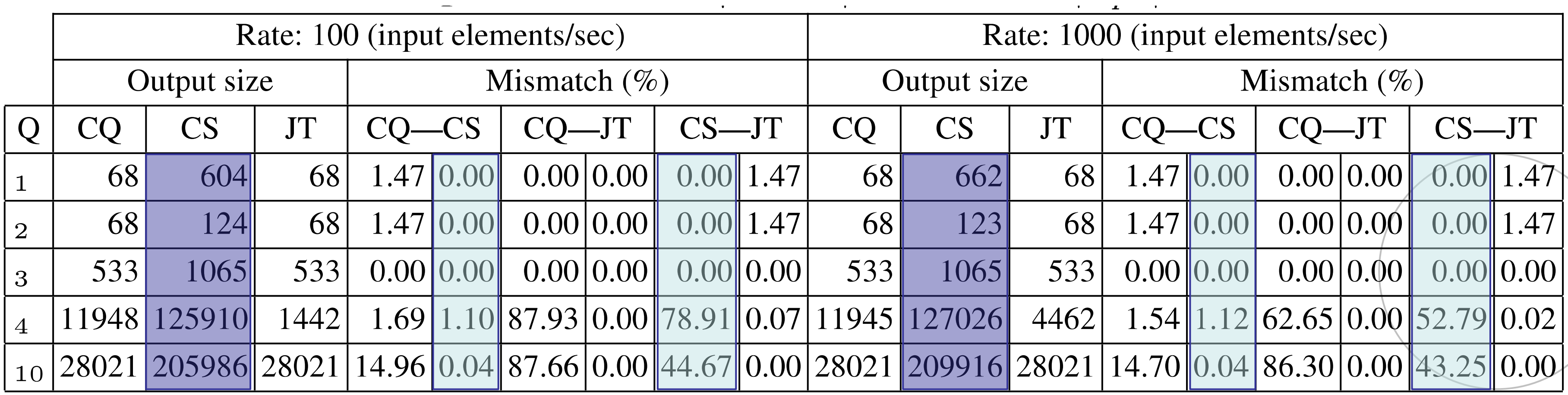
$$T_i = P_i^{match(T_{i-1})} = A_i \pitchfork O_{E_2}\lfloor match(T_{i-1})\rfloor$$

Maximum subsequences that have matches

$$mm(E_1, E_2, Q, R) = \frac{\Sigma_{i=1}^{m}(|A_i| - |T_i|)}{\Sigma_{i=1}^{m}|A_i|} \times 100\%$$

Mismatch metric

Some mismatch values vs. output size



	Rate: 100 (input elements/sec)									Rate: 1000 (input elements/sec)								
	Output size			Mismatch (%)						Output size			Mismatch (%)					
Q	CQ	CS	JT	CQ—CS		CQ—JT		CS—JT		CQ	CS	JT	CQ—CS		CQ—JT		CS—JT	
1	68	604	68	1.47	0.00	0.00	0.00	0.00	1.47	68	662	68	1.47	0.00	0.00	0.00	0.00	1.47
2	68	124	68	1.47	0.00	0.00	0.00	0.00	1.47	68	123	68	1.47	0.00	0.00	0.00	0.00	1.47
3	533	1065	533	0.00	0.00	0.00	0.00	0.00	0.00	533	1065	533	0.00	0.00	0.00	0.00	0.00	0.00
4	11948	125910	1442	1.69	1.10	87.93	0.00	78.91	0.07	11945	127026	4462	1.54	1.12	62.65	0.00	52.79	0.02
10	28021	205986	28021	14.96	0.04	87.66	0.00	44.67	0.00	28021	209916	28021	14.70	0.04	86.30	0.00	43.25	0.00

Periodical and recurrent execution(CS-CPARQL) might output greater number of output but they are repeated and contained in output results of eager and incremental execution (CQ-CQELS, JT-JTALIS)

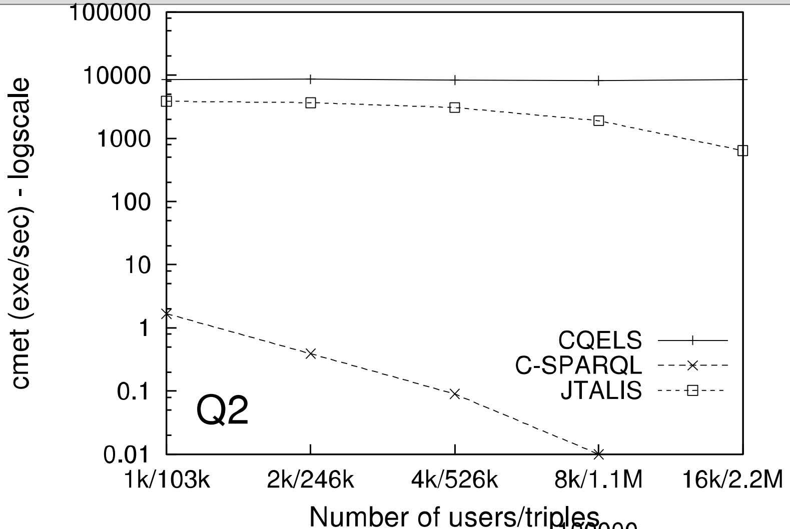
Comparable maximum execution throughputs(executions/sec)

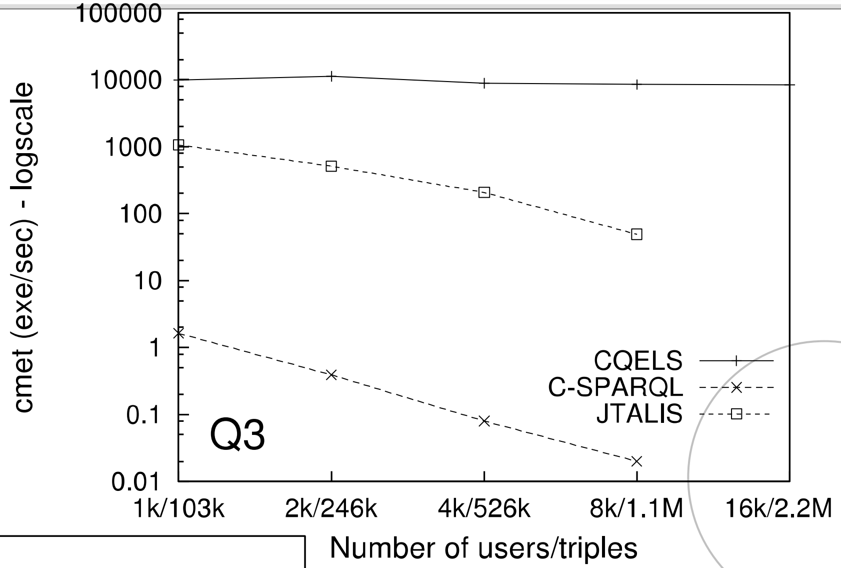
How fast they are?

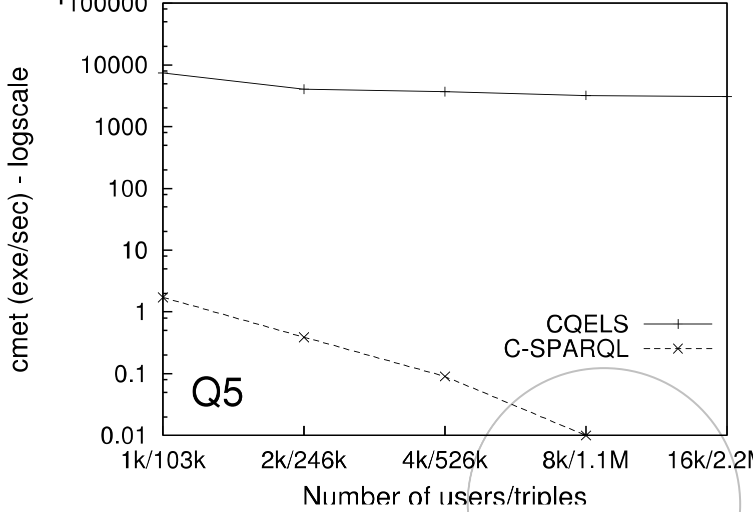
	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_{10}
CQELS	24122	8462	9828	1304	7459	3491	2326
C-SPARQL	10	1.68	1.63	10	1.72	1.71	10
JTALIS	3790	3857	1062	99	—	—	87

- Twitter might deliver 20k–100k tweets/sec
- Berlin Benchmark for triple storages (Feb 2011) : ~50–400 queries/sec
- ESPER : 200k–500k events/sec
- SASE :10–150k events/sec

Maximum execution throughput when varying static data size

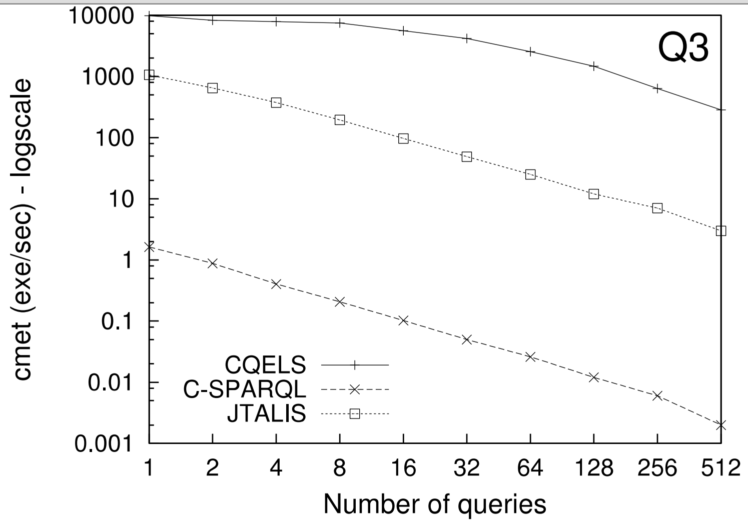


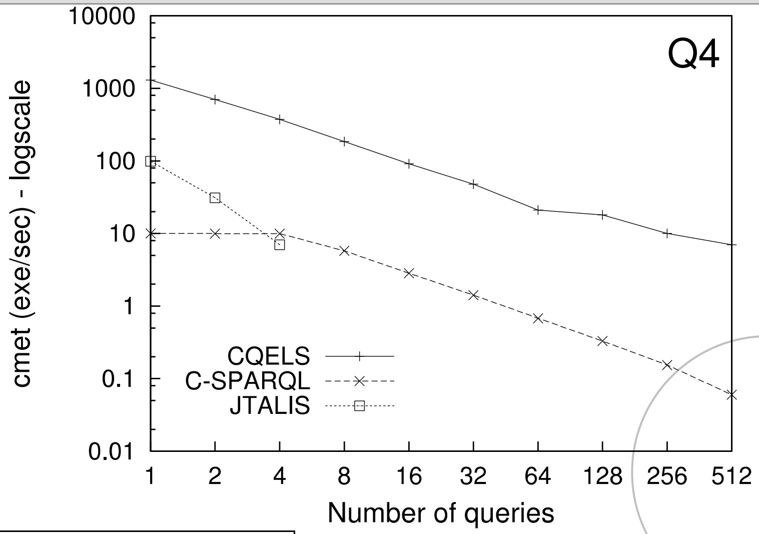


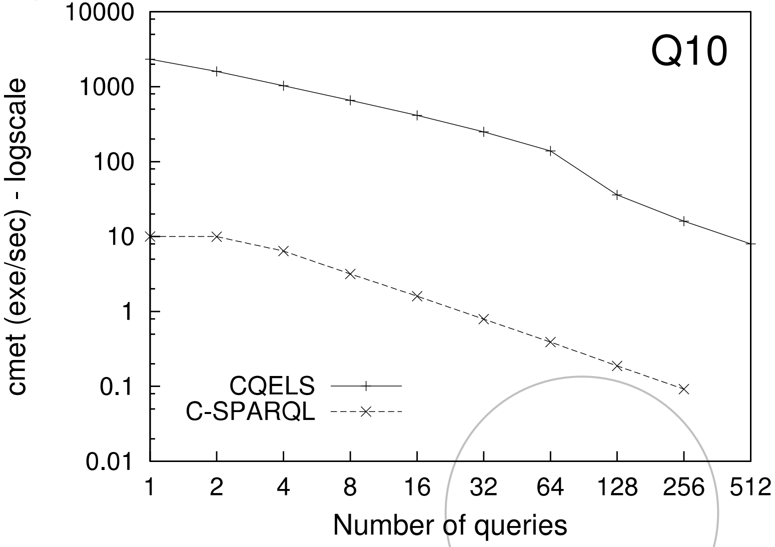


C-SPARQL and JTALIS could not scale with big static data size

Maximum execution throughput for multiple queries







None of the systems could share the processing among multiple queries

Findings

- Expressiveness: None of state-of-the-art continuous query languages support all SPARQL query patterns
- Validity: Throughputs based on the number outputs/inputs are invalid.
- Performance: Incremental&eager execution (JTALIS,CQELS) outperforms over periodical&recurrent execution (C-SPARQL)
- Scalability: Most of the engines have poor scalability
  - □ On static data size : only CQELS can scale with static data sizes by pre-computing and caching static sub-queries
  - □ On the number of queries : none of the engines apply multiple query optimisation techniques.

Summary

- Evaluation Framework (LSBench)
  - □ Test suits
  - □ Data generator for social networks
- First extensive cross system evaluations
  - □ Validity of evaluations
  - □ Evaluation methodologies
- Findings
  - □ Expressiveness
  - □ Performance
    - – Execution throughput
    - – Scalability
      - – Static data size
      - – Number of current queries