Towards Expressive Publish/Subscribe Systems
by Alan Demers, et al.
presented by
Risi Thonangi
Feb 5, 2008

Applications
• Monitoring large computing systems, networks
  — Detect failures and security threats
  — Compliance with Service Level Agreements
• Business Activity Monitoring, Business Process Management
  — Supply chain management with RFID tags
  — Monitoring of industrial processes
• Expressive publish-subscribe (pub/sub) over RSS feeds, blogs

Cayuga
• Real-time processing of event streams
• Expressive query language
  — Filter, project, aggregate, join (correlate) events from multiple streams
  — Fully composable operators with formal semantics
• Distinguishing feature: Effective multi-query optimization
  — Throughput of tens of thousands of events per second for hundreds of thousands of active queries (depends on query complexity and similarity, of course...)

Cayuga Algebra: Data Model
• Event stream
  — denoted as S or $S$
  — Possibly infinite set of tuples
• Follows a fixed schema: $(a; t_0, t_1)$, where $a = (a_1, a_2, ..., a_n)$
• $t_0$ – start timestamp of the event
• $t_1$ – end timestamp of the event
• Events arrive in temporal order (ordered on $t_1$)

Cayuga Algebra: Operators
• Unary operators:
  — Projection : $\pi_X$
    • project attributes $X$
  — Selection : $\sigma\theta$
    • select event if it satisfies $\theta$
    • can contain predicate of the form: $DUR > t$. ‘DUR’ stands for ‘duration of the event’.
  — Renaming : $\rho_f$
    • rename variables denoted by $f$
• Unary operators are ‘Stateless’ – information from events already matched is not used

An Example Scenario:
Notify: when the price of IBM is above $100, and the first MSFT price is below $20

IBM,$180 MSFT,$22 IBM,$100 MSFT,$20
Cayuga Algebra: Operators

- Binary operators:
  - Union: \( S_1 \cup S_2 \) — union events from two streams
  - Conditional sequence: \( S_1 ; \theta S_2 \) — concatenate consecutive events that satisfy the condition \( \theta \).
  - Iteration: \( \mu F, \theta S_1 \) — \( S_1 \) is repeated as \( \theta \) parameter explains how to modify the result of each iteration.
  - Aggregate: \( g \) introduces aggregation variables

Conditional Sequence Operator

- \( S_1 \) is \( \sigma_{\theta_1}(S) \) and \( S_2 \) is \( \sigma_{\theta_2}(S) \) where \( \theta_1 \) is (stock = IBM and price > $100) and \( \theta_2 \) is (price < 25).

Implementation of 'Cayuga Algebra'

- Cayuga Automatons
  - Similar to the classic NFAs
  - Except that:
    - each automaton edge is labeled with a predicate
    - at each traversed automation state, attributes and values that contributed to the state transition are stored
  - Left-Deep expressions are translated to single automatons

Creating Cayuga Automatons

Architectural of 'Cayuga System'

- The system stores the queries that are being evaluated and their partial matches
  - Done by storing Cayuga automatons and their active states
  - [optimization] All the automatons currently in the system are merged into a query DAG and identical states are merged
    - Decreases cost of evaluating state transitions
Architecture of ‘Cayuga System’

- [optimization] Automaton instances are indexed based on state transition conditions
  - Helps in quickly determining affected instances
- FR Index: keys are forward/rebind static conditions
- AN- & AI-Indices: keys are conditions on filter predicates

Processing steps on an event arrival:
- FR Index generates Instance/Edge matches
- AN- & AI-Indices generates set of active instances
- For each active instance, evaluate dynamic parts of the FR edges

Experimental Results

Subscriptions were generated from 5 different templates:
- LinearStat – \( \sigma_{\theta_1}(\sigma_{\theta_2}(\sigma_{\theta_3}(S_1); S_2); S_3) \) (static conditions)
- LinearDyn -- \( \sigma_{\theta_1}(\sigma_{\theta_2}(\sigma_{\theta_3}(S_1); S_2); S_3) \) (dynamic conditions)
- Filter -- \( \sigma_{\theta_1}(\sigma_{\theta_2}(\sigma_{\theta_3}(S_1); \theta_4 S_2); \theta_5 S_3) \) (non-consecutive)
- NonDeterministic -- \( \sigma_{\theta_1}(\sigma_{\theta_2}(\sigma_{\theta_3}(S_1); S_2), S_3) \) (more than 3)
  - NonDeterministicAgg – implements aggregation

Some thoughts

- Some language constructs explicitly require the user to give hints about query execution
  - What about query optimization?
- How scalable is the system
  - Are we efficiently using main memory and caches
- Comparison with existing work

Points of discussion on Cayuga Paper

- Some language constructs explicitly require the user to give hints about query execution
  - What about query optimization?
  - Trade off between restrictive and flexible
- How scalable is the system?
  - Are we efficiently using main memory and caches
- Comparison with existing work
- NFA for Database systems?
- Cayuga algebra – Confusing or amusing?