Towards Expressive Publish/Subscribe Systems

by Alan Demers, et al.

presented by
Risi Thonangi
Feb 5, 2008

An Example Scenario:

Notify: when the price of IBM is above $100, and the first MSFT price is below $25

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

Slide borrowed from Alan Demers, et al. CIDR 2007.

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…)

Slide borrowed from Alan Demers, et al. CIDR 2007

Cayuga Algebra: Data Model

- Event stream
  - denoted as S or S_i
  - Possibly infinite set of tuples
- Follows a fixed schema: $(\boldsymbol{a}; t_0, t_1)$, where $\boldsymbol{a} = (a_1, a_2, \ldots, a_n)$
- 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 : π_X
    - project attributes X
  - Selection : σ_θ
    - select event if it satisfies θ.
    - can contain predicate of the form: `DUR` > t. `'DUR'` stands for 'duration of the event'.
  - Renaming : ρ_f
    - rename variables denoted by f
- Unary operators are 'Stateless' – information from events already matched is not used

Cayuga Algebra: Operators

- Binary operators:
  - Union: $\cup$
    - $S_1 \cup S_2$ -- union events from two streams
  - Conditional sequence: $;_\vartheta$
    - $S_1 ;_\vartheta S_2$ -- concatenate consecutive events that satisy the condition ϑ.
  - Iteration: $\mu_{F,\vartheta}$
    - $S_1 \mu_{F,\vartheta} S_2$ -- $(S_1 ;_\vartheta S_2) \cup (S_1 ;_\vartheta S_2 ;_\vartheta S_2) \cup \ldots$
    - F parameter explains how to modify the result of each iteration.
  - Aggregate: α_g
    - g introduces aggregation variables

Conditional Sequence Operator

Notify: when the price of IBM is above $100, and the first MSFT price is below $25

S1

Stock	Price	time
IBM	$180	22s
IBM	$100	35s
IBM	$120	40s

$;_\vartheta$

$;_\vartheta$

S

Stock	Price	time
IBM	$180	22s
MSFT	$22	25s
IBM	$100	35s
IBM	$120	40s
MSFT	$20	47s

S1 = $\sigma\theta$(S); θ is (S.stock=IBM and S.price > $100)
Answer = $\sigma_{\theta 2}$(S1 $;_{\theta 1}$ S); θ_1 = S.stock = MSFT, θ_2 is S.price < 25

Iteration Operator

Notify: when the price of a stock increases monotonically for 15s

S1, S2

Stock	Price	time
IBM	$100	22s
MSFT	$22	25s
IBM	$110	35s
IBM	$120	40s
MSFT	$20	47s

Answer = $\sigma_{\theta 1}(\mu_{F,\vartheta}$ (S1,S2);

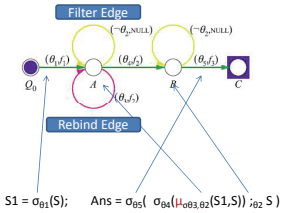
θ is (S1.stock = S2.stock),
θ1 is (DUR > 15s)

F is $\sigma_{\theta 2}$ where θ2 is S2.price > S2.price.last

Implementing 'Cayuga Algebra'

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

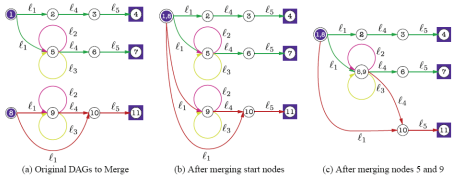
Creating Cayuga Automatons



S1 = $\sigma_{\theta 1}$(S); Ans = $\sigma_{\theta 5}$($\sigma_{\theta 4}(\mu_{\sigma\theta 3, \theta 2}$(S1,S)) $;_{\theta 2}$ S)

Architecture of 'Cayuga System'

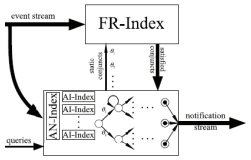
- The system stores the queries that are being evaluated and their partial matches
  - Done by storing Cayuga automatons and their active states
- [optimization] All the automatons currently in the system are merged into a query DAG and identical states are merged
  - decreases cost of evaluating state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



(a) Original DAGs to Merge (b) After merging start nodes (c) After merging nodes 5 and 9

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



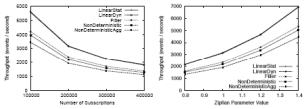
Processing steps on an event arrival:
› FR-Index generates Instance/Edge matches
› AN- & AI-Indices generates set of active instances
› For each active instance, evaluate dynamic parts of the FR edges.

Experimental Results

Variable	Value	Variable	Value
Number of events	100,000	Number of attributes per event	8
Number of discrete attributes	4	Number of continuous attributes	4
Number of subscriptions	200,000	Domain size of discrete attribute	100
Number of atomic predicates (discrete + continuous)	2 + 2	Number of distinct ranges that can be selected for inequality predicates	25
Selectivity of atomic inequality predicate	0.7	Number of steps per sequence query	3
Zipf parameter, first step ($zipf_1$)	1	Zipf parameter, second step ($zipf_2$)	1
Zipf parameter, third step ($zipf_3$)	0.8	Duration constant (t)	20

Subscriptions were generated from 5 different templates:
LinearStat – $\sigma_{\theta 3}(\sigma_{\theta 2}(\sigma_{\theta 1}(S1)\ ;\ S2);\ S3)$ (static conditions)
LinearDyn -- $\sigma_{\theta 3}(\sigma_{\theta 2}(\sigma_{\theta 1}(S1)\ ;\ S2);\ S3)$ (dynamic conditions)
Filter -- $\sigma_{\theta 3}(\sigma_{\theta 2}(\sigma_{\theta 1}(S1)\ ,_{\theta 4}\ S2);_{\theta 5}\ S3)$ (non-consecutive)
NonDeterministic -- $\sigma_{\theta 3}\mu(\sigma_{\theta 2}\mu(\sigma_{\theta 1}(S1),S2),S3)$ (more than 3)
NonDeterministicAgg – implements aggregation

Experimental Results



(a) Throughput vs. number of subscriptions (b) Throughput vs. Zipf skew

Effect of MQO

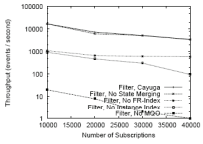


Fig. 7. Effect of multi-query optimization

Mode Name	StateMerge	FR-Index	Instance Index
Cayuga	on	on	on
No State Merging	off	on	on
No FR-Index	on	off	on
No Instance Index	on	on	off
No MQO	off	off	off

Table 4. Meaning of the curves

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?