Scalable Continuous Query Processing and Result Dissemination

Jun Yang
Duke University
Joint work with
Pankaj Agarwal, Radish Chandramouli, Junyi Xie, Hai Yu

Announcements (Dec. 5)

- Homework #4 due today
- No class on Thursday
- Project demos start next week; schedule through email
- Final exam on Dec. 15 (9am – 12pm)
  - Open book, open notes
  - Final review session on Dec. 14 (3pm – 5pm)
  - Similar format as sample final
    • Solution available today
- Course evaluation forms
- Missing handouts and graded assignments: check handout box or email me

A shift in query paradigm

- One-time query over a static snapshot of database
- Continuous query over an input update stream
- Applications
  - Environmental monitoring (NSF NEON)
  - Network management (Ganglia)
  - Personal publish/subscribe (Google Alert)
- Scalability challenges
  - Too much data
  - Too many continuous queries
  - Results needed all over the network
Challenge: too many queries!

For each incoming update...

- Naïve: For each CQ, compute & send result update
  - Linear in # of CQs; not scalable
- Group processing: share work across queries!
- Query-data inversion: treat CQs as data, incoming update as query
- If all CQs are filters (e.g., $60 < \text{PRICE} < 80$), use an index on filters (e.g., interval tree) for finding affected queries in sub-linear time

But how about (select-)joins?

Database relations: $R(A, B), S(B, C)$

$Q_i$: $(\text{SELECT}_{\text{rangeAi}} R) \text{ JOIN } (\text{SELECT}_{\text{rangeCi}} S)$

- $\text{JOIN}$ matches $R, S$ tuples with equal $B$ value
- $\text{SELECT}_{\text{rangeAi}} / \text{SELECT}_{\text{rangeCi}}$ select only those passing local (range) selection conditions
- Example: matching Supply & Demand
  - $\text{Supply.product} = \text{Demand.product}$ AND
  - $\text{Supply.rating} \in [7, 10]$ AND
  - $\text{Demand.quantity} > 1000$

Method 1: select first

$Q_i$: $(\text{SELECT}_{\text{rangeAi}} R) \text{ JOIN } (\text{SELECT}_{\text{rangeCi}} S)$

Given data update new $r(a, b) \in R$

- Find subset of CQs whose selection condition on $R$ is satisfied by $r$
  - Use an index on all $\text{rangeAi}$’s
- Process each such $Q_i$
  - Use an index on $S$ (e.g., B-tree w/ compound key $BC$) to identify $S$ tuples with $S.B = b$ and $S.C \in \text{rangeCi}$
- But what if lots of $Q_i$’s survive the first step?
Method 2: join first

Given data update new $r(a,b) \in R$

- Find all $S$ tuples that join with $r$
  - Use an index on $S$
- Process each such tuple $s$
  - Use an index on all CQs
    (e.g., R-tree on $(\text{range}A_i \times \text{range}C_i)$)
to identify $Q_i$’s for which
  $a \in \text{range}A_i$ and $s.C \in \text{range}C_i$
- But what if lots of $S$ tuples join with $r$?

Problem of intermediate result size

- Each method forces a particular processing order
  - Method 1: select first
    - Cost depends on $n’$ (# of $\text{range}A_i$’s containing $a$)
  - Method 2: join first
    - Cost depends on $m’$ (# of $S$ tuples that join with $r$)
  - Both $n’$ and $m’$ can be huge even if final output size is small $\approx$ “OpenBSD birthday pony”

- Can we make processing cost independent of $n’$ & $m’$?

Idea: exploit input characteristics

- CQs (=user interests) often are naturally clustered
  - Take advantage of clusteredness in processing
- Stabbing Set Index
  - Partition intervals into disjoint stabbing groups, where
    in each group all intervals are stabbed by a same point
  - Stabbing number $\tau = \# \text{of stabbing groups}$
  - Fast construction and maintenance
    - Can be constructed optimally (with smallest $\tau$ possible) in $O(p \log n)$ time
    - Can be maintained within $1 + \varepsilon$ of the optimal in $O(1 + 1/\varepsilon \log n)$ time
Algorithm based on stabbing groups

- Use a stabbing set index on all rangeCi's
- For each stabbing group (with common point p)
  - Find the two points on the a line (i.e., two S tuples joining with r) closest to p
    - Use an index on S (e.g., B-tree w/ compound key BC)
  - Find all rectangles in the stabbing group stabbed by one of the two points
    - Use an index (e.g., R-tree) on this stabbing group of CQs

Cost analysis

- O(τ × (three index lookups) + output)
  - Cost depends on τ, not on m' or n'
  - Input-sensitive
    - More clusteredness in CQs
      → Smaller τ
      → Lower cost

- Compare with:
  - Method 1: \(O(n' × (index\ lookup) + output)\)
  - Method 2: \(O(m' × (index\ lookup) + output)\)

Experiments

100K CQs; 100K-row relations
Experiments

100K CQs; 100K-row relations

Throughput (# of updates/sec)

Avg. # of joining tuples (m')

Experiments

100K CQs; 100K-row relations

Throughput (# of updates/sec)

# of stabbing groups (τ)

More input-sensitivity

- Input-sensitive dynamic optimization
  - For each incoming update, look at τ, m', and n' to decide how to process it
  - Maintain the stabbing set index, but only process large groups in the new way

- Input-sensitive scalable processing of band joins
  - Join condition: R.B ∈ rangeBi
  - First attempt at scalably group-processing joins with different join conditions
Just covered: challenge of too many queries
- [Agarwal, Xie, Yu, Yang; VLDB 2006]

Next: delivering results all over the network
- [Chandramouli, Xie, Yang; SIGMOD 2006]

Dissemination bottleneck

- Traditional DB-centric approach
  - Focused on subscription processing
  - Ignored notification dissemination
- Implicit assumption: output a list of notifications, one for each affected subscription
  - \((Q_{t_1}, msg), (Q_{t_2}, msg), (Q_{t_3}, msg), \ldots\)
  - Potentially a very long list
  - Sending them to subscribers one at a time (unicast) can overwhelm the server and its outgoing network links

Network-centric approach

- Unicast/broadcast
- Multicast = channel-based subscriptions
- Content-based networking (CN):
  - Supports message-based filter subscriptions directly in network
  - Message:
    \((\text{attr}_{\text{val}_1}, \text{attr}_{\text{val}_2}, \text{attr}_{\text{val}_3}, \ldots)\)
  - Subscription:
    \(\text{“attr}_1 = \text{’foo’ and attr}_2 \in \text{range and …”}\)
- Gets close to SQL-style declarative CQs, but still doesn’t support stateful CQs
Stateful subscription example

- Range-min subscription
  - Q: select MIN(PER) from STOCK where RISK between 20 and 40
- Update message (SYM:foo, RISK:35, PER:25 → 20)
  - Stateful: cannot determine its effect on Q just by looking at the message itself
    - Is there another stock in RISK range with PER < 20?

Supporting stateful subscriptions

- Just stick the DB-centric approach and a network together?
  - “List of affected subscriptions” leads to unicast
  - Multicast: map the list to group(s) first, then send
    - Too many possible subsets! What groups to form?
- Push state support into network of smart brokers?
  - Complicates system design and deployment
- Content-based network?
  - Naïve method: “relax” subscription into a stateless one
    - select MIN(PER) from STOCK where RISK between 20 and 40
    - select PER from STOCK where RISK between 20 and 40
    - Too many unnecessary notifications!

Message/subscription reformulation

- DB reformulates messages to add state info
- Reformulate subscriptions into stateless ones over new message format
  - Naïve: put entire database state into message!
  - Optimization problem: what’s the minimal amount of info to embed?
Range-min revisited

- Q: MIN(PER), where RISK between $x_i$ and $y_i$
- Update (SYM:foo, RISK:35, PER:25 $\rightarrow$ 20)

What info should DB send out to help decide whether a subscription is affected by an update?

- Maximum Affected Range (MAR): extends left & right until a lower PER is encountered
- Affected $\iff$ RISK of update $\in [x_i, y_i] \subseteq$ MAR of update
- Can be computed in $O(\log |STOCK|)$—does not depend on how many subscriptions are actually affected!

Reformulation for range-min

- Message reformulation (at runtime):
  (SYM:foo, RISK:35, PER:25 $\rightarrow$ 20)
  Say MAR is (17, 52)
  $\implies$ (NewMinPER:20, RISK:35, MARLeftRISK:17, MARRightRISK:52)

- Subscription reformulation (at registration time)
  Q: MIN(PER), where RISK between $x_i$ and $y_i$
  $\implies$ Q': NewMinPER, where
  MARLeftRISK $<$ $x_i$ $\leq$ RISK and RISK $\leq$ $y_i$ $<$ MARRightRISK

Changing role of DB
- From producing the set of affected subscriptions
- To producing a semantic description of the set

Experiment

- Content-based network (CN) substrate: Meghdoot (UCSB; based on CAN)
- Yahoo! stock updates + synthetic subscriptions

Orders of magnitude difference

Reformulation + CN

Avg network traffic per event (bytes)

Number of subscriptions (x 1000)
**Bigger picture**

- Spectrum of DB/network interfaces to explore
- Message/subscription reformulation is a general technique for handling stateful subscriptions over a stateless dissemination interface
  - Clean, modular system design
- Input-sensitive dynamic optimization
  - Choose best dissemination method at runtime
- Think of dissemination networks as database indexes!
- Input-sensitive dissemination network design
  - Analogous to workload-aware index design

**Conclusion & take-away points**

- Static queries → continuous queries
- Scalability challenges
  - Lots of data: [Xie, Yang, Chen; SIGMOD 2005]
  - Lots of queries: [Agarwal, Xie, Yu, Yang; VLDB 2006]
  - Distributed subscribers: [Chandramouli, Xie, Yang; SIGMOD 2006]
- Exploit data/query characteristics with dynamic input-driven processing
- Rethink database/network interface
- Jointly optimize data processing/dissemination

**Related work**

- High data rates
  - Focus of most work on stream processing: Aurora/Borealis (Brandeis/Brown/MIT), STREAM (Stanford), TelegraphCQ (Berkeley), etc.
- Lots of queries
  - Multi-query optimization
  - Lots of work on predicate indexing
  - Beyond predicates: TriggerMan (Florida), NiagraCQ (Wisconsin), CACQ/PSoup (Berkeley)
- Widely distributed subscribers
  - IP- and application-level multicasts
  - Content-based networking (IBM Gryphon, Colorado)
  - YFilter/ONYX (Berkeley), SemCast (Brown)
  - DEBS Workshop
Thanks!

Duke Database Research Group
http://www.cs.duke.edu/dbgroup/