PeerCQ: A Decentralized and Self-Configuring Peer-to-Peer Information Monitoring System

B. Gedik, L. Liu

Outline

- Design Goal
- Continual Queries
- PeerCQ Overview
- PeerCQ Protocol
- Routing / Membership Management
- Simulation Results
- Experimental Results
- Conclusions
- Discussions

PeerCQ

- **Goal:**
  - **Decentralized Internet scale distributed information-monitoring system**
- **Approach:**
  - **Uses Continual Queries (CQ) to monitor info**
  - **Routes CQs to peers**
  - **Respects peer heterogeneity and user characteristics**
  - **No global information is needed**

Continual Queries

- **"Standing queries that monitor updates and return results whenever the updates have reached specified thresholds."**
- **cq: (cq_id, trigger, query, stop_cond)**
  - **trigger: (mon_src, mon-item, mond_cond)**
  - **Result from query is returned to the user**
  - **stop_cond specifies terminating condition**

Continual Queries

- Two types of trigger conditions
  - Time-based trigger condition
    - Absolute points in time
    - Regular / irregular time interval
    - Relative temporal event
  - Content-based trigger condition
    - Database queries

Continual Queries

- **Event Detection**
  - **Synchronous observation: Event occurrence communicated explicitly to and in sync with the event observer. For example, database triggers in RDBMS systems.**
  - **Polling: The observer periodically checks for occurrence of event.**
- **OpenCQ: Implementation of CQ**

(Ling Liu, Calton Pu, Wei Tang)

Continual Queries

- **Example 1:**

  "Report to the manager every day at 6:00pm all the banking activities of the day for those customers whose total withdraws reach $2,000."

  Create CQ banking_activity_sentinel as

  Query:

  SELECT cust_id, acct_no, withdraw_amt
  FROM Account
  GROUP BY cust_id having SUM(withdraw_amt) > 2000;

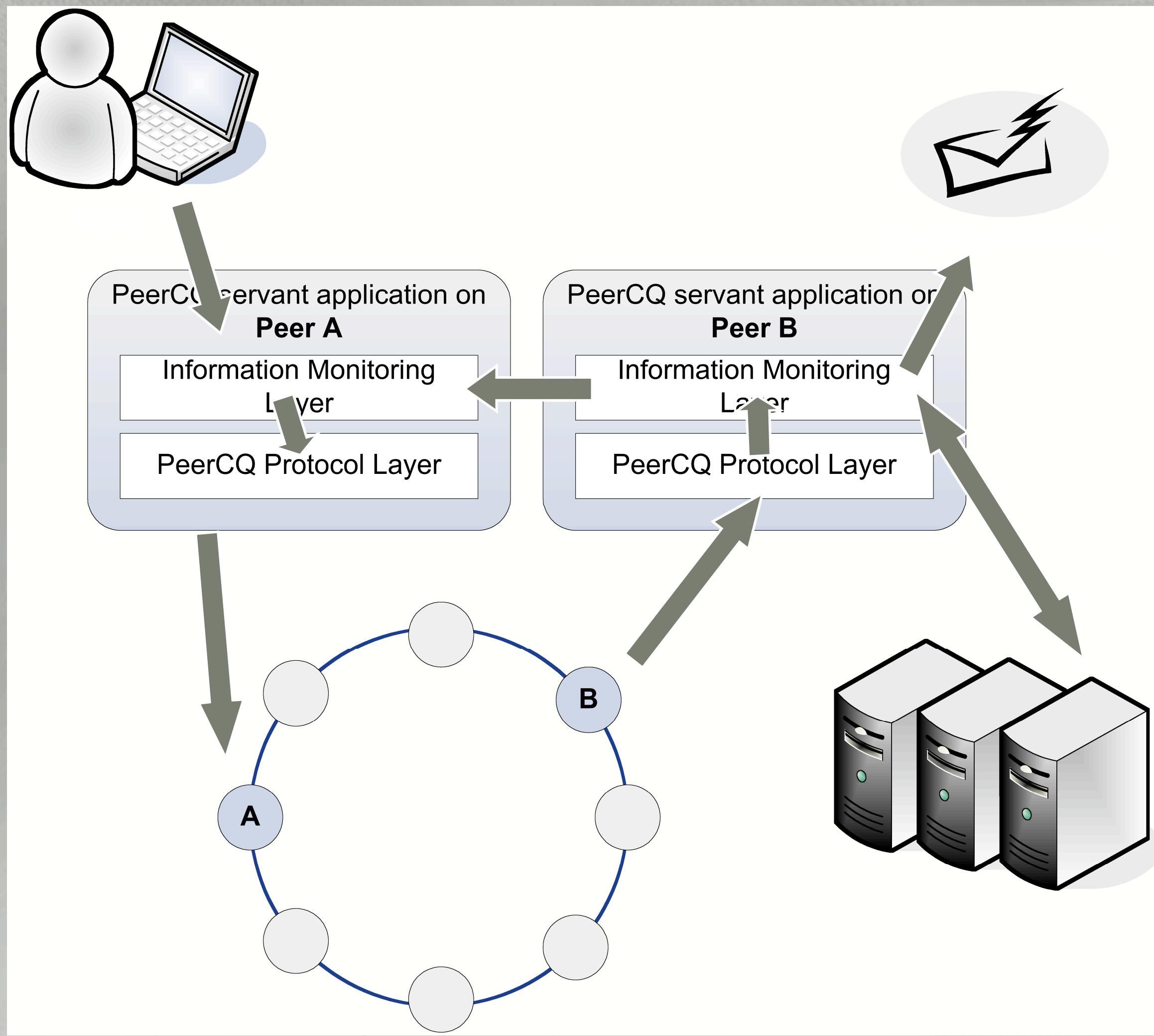
  Trigger: 6:00pm everyday

  Stop: 1 year (by default)

Continual Queries

- **Example 2:**
  - “Notify me in the next six months whenever the total quantity on hand and quantity on order of items drops below their threshold.”
  - Create CQ inventory_monitoring as
  - Query:
    - SELECT item_name, item_no, qty_on_hand, qty_on_order, threshold
    - FROM Item_Inventory;
  - Trigger:
    - qty_on_hand + qty_on_order < threshold;
  - Stop: six months

PeerCQ Overview



PeerCQ Protocol

- Goal:
  - CQ-awareness
    - Similar triggers are grouped
  - Peer-awareness
    - More CQs assigned to higher capability peers
  - Cache-awareness
    - CQs are assigned to peers according to the content of the caches

Strict Matching

- **Follows consistent matching**
  - **Assigns CQ to peer with id closest to cq_id**
  - **Peers with higher capability are assigned with more peer ids**
- **Effective Donation**
  - **Perceived donation of the peer by system**
  - **ED $\in$ [1,C], where 1=min, C=max**
  - **R (resources) = <“cpu”, “hard disk”, “memory”, “network bandwidth”>**
  - **AR (actual resources); PD (peer donation)**

Strict Matching

AR[1] → RP[1]
[0, 400) → 1, old
[400, 800) → 2
[800, 1200) → 3, moderate
[1200, 1600) → 4
[1600, 2000+) → 5, powerful

AR[2] → RP[2]
[0, 15) → 1, small disks
[15, 30) → 2
[30, 45) → 3, moderate disks
[45, 60) → 4
[60, 75+) → 5, large disks

AR[3] → RP[3]
[0, 64) → 1, small mems
[64, 128) → 2
[128, 256) → 3, moderate mems
[256, 512) → 4
[512, 1024+) → 5, large mems

AR[4] → RP[4]
[0, 64) → 1, dial-up
[64, 128) → 2, ISDN
[128, 256) → 3, ISDL / Cable
[256, 512) → 4, ASDL / Cable
[512, 1024+) → 5, Cable / T1

Strict Matching

```
calculateED(P, PD, AR)
   ED = 0
   // i stands for the four types of  resources;
   // cpu, memory, hard disk, network conn.
   for i = 1 to 4
      RP[i] = MF[i](AR[i])
      DP[i] = PD[i] * RP[i]
      ED = ED + RI[i] * DP[i]
      ED = ⌈ P.rel * (C/5) * ED ⌉
      return ED
```

Strict Matching

- Mapping CQs to identifiers
  - CQs are similar if mon_srcs and mon_items are the same
  - CQ ids are composed of 2 hashed values
  - Grouping factor controls the size key space
  - Hotspots may form for popular CQs

Relaxed Matching

- Idea:
  - Take into account data source proximity, caching and load balancing
  - Off-load CQ to neighbour when appropriate
- UtilityF(p,cq)=

  PLF(p.peer_props.load) *

  (CAF(p.peer_props.cache,cq.mon_item) + α * (SDF(p.peer_props.IP,cq.mon_src))
- Shows load-aware & cache-aware

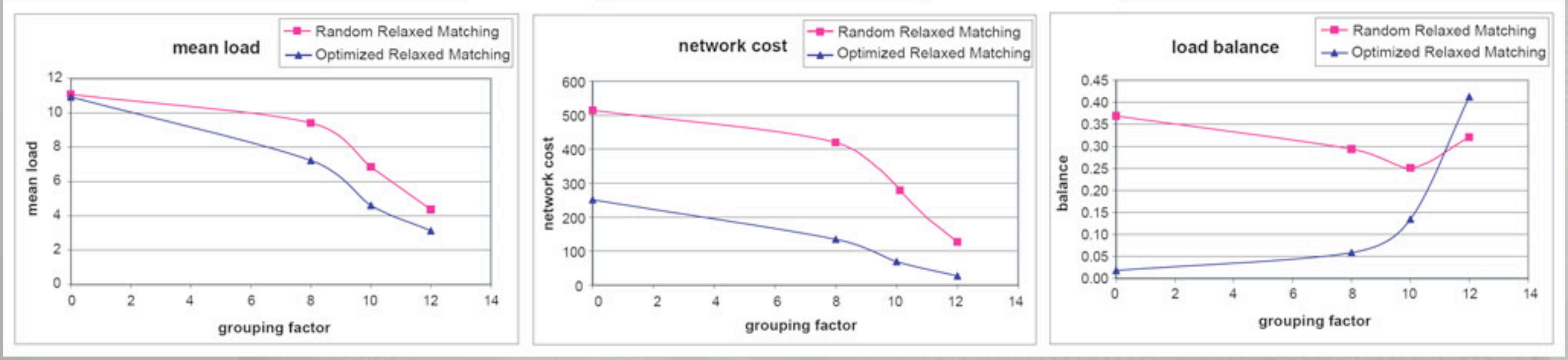
Routing / Membership Mgmt

- Lookup functions similar to Chord
  - Uses routing table and neighbour list
  - Allows bi-directional traversals
- CQs owned by neighbours are migrated
- Concurrent joins / departures are synchronized in neighbour list
- Periodic polling messages detect failures
- Neighbour list repairs failures

Simulation Results

- Effective Donation (ED)
  - Number of CQs assigned to peer is proportional to number of ids it has
- Grouping Factor
  - Increasing grouping factor too much destroys load balancing property
  - Optimized relaxed matching is more effective in grouping CQs

Experimental Results



Conclusions

- PeerCQ distributes CQs over the Internet
- Incorporates CQ-awareness, Peer-awareness and cache-awareness
- "PeerCQ is highly scalable, self-configurable and supports efficient and robust way of processing CQs."

Reference

B. Gedik, L. Liu. PeerCQ: A Decentralized and Self-Configuring Peer-to-Peer Information Monitoring System.

B. Gedik, L. Liu. PeerCQ: A Scalable and Self-Configurable Peer-to-Peer Information Monitoring System.

L. Liu, C. Pu, W. Tang. Continual Queries for Internet Scale Event-Driven Information Delivery.

Comments

Pluses

- Best paper award in ICDCS 2003
- Results from both simulation and real implementation

Minuses

- No discussion of reliability and security
- No incentive to report true capacity, may lead to demise of the system
- No comparison between Chord and PeerCQ
- No latency measurements

Discussions

- How can PeerCQ be made secured?
  - Peers need access to the trigger / query
- Can token-based incentive be useful?
  - Encourages accurate capability data
- Can Chord be used in strict matching?