CrowdDB: Answering queries with Crowdsourcing

Michael Franklin et al., SIGMOD’11

Presentation by Parijat Mazumdar
CrowdDB: Motivation

Two fundamental problems with present RDBMSs:

- Closed World Assumption
- Extremely literal

```
SELECT market_capitalization FROM company
WHERE name = "I.B.M"
```

Entity Resolution: easy for humans, difficult for computers

Get the best of both sides:

- Human power for subjective compare and finding new data
- Traditional RDBMs for heavy lifting data manipulations
Design considerations for CrowdDB

- **Performance and variability**: Humans are slow, inaccurate, costly, variable
- **Task design and ambiguity**: Natural language is inherently ambiguous
- **Affinity and learning**: Workers develop relationships with requesters, skills for certain kind of HITs
- **Open world**: Possibly unbounded number of answers
CrowdDB: Overview

- Query database using CrowdSQL: slight extension of standard SQL
- Automatic UI generation
- Automatic interaction with crowdsourcing marketplace
- Automatically store results obtained from crowd into database for future use.
Amazon Mechanical Turk (AMT)

- HIT: smallest entity of work a worker can accept to do
- Assignment: Replicas of same HIT for majority voting
- HIT Group: Similar HITs put together

AMT API:
- `createHIT(HIT parameters) → HitID`
- `getAssignmentsForHIT(HitID) → list(asnId,workerId,answer)`
- `approveAssignment(asnID) / rejectAssignment(asnID)`
- `forceExpireHIT(HitID)`
CrowdSQL : Incomplete data

special keyword : CROWD

Crowdsourced Columns
CREATE TABLE Department ( university STRING, name STRING, url CROWD STRING, phone STRING, primary key (university, name) );

Crowdsourced Tables
CREATE CROWD TABLE Professors ( name STRING PRIMARY KEY, email STRING UNIQUE, university STRING, department STRING, FOREIGN KEY (university, department) REF Department(university, name) );

Crowd equivalent of NULL
• CNULL values get filled during queries

special keyword : CNULL

SELECT url from Department WHERE name = “math”
CrowdSQL: Comparisons

CROWDEQUAL

SELECT name FROM Professor
WHERE department ~= "CS"

CROWDORDER

SELECT p from picture
WHERE subject = "Golden Gate Bridge"
ORDER BY CROWDORDER(p,
"Which picture visualises better %subject")
Basic UI generation

- UI Templates are created at compile time
- Instantiated at run-time for each tuple
- Can be edited by application developers
Multi-relational UI generation

• Foreign key references a non-crowdsourced table:
  • Drop-down box
  • Ajax-based “suggest” function

• Foreign key references a crowdsourced table:
  • Normalised interface with “suggest” function
  • Denormalized interface
Query Processing

Extended operators called crowd operators
- **CrowdProbe** collects missing information in CROWD columns/tables
- **CrowdJoin** collects new tuples from inner relation
- **CrowdCompare** implements CROWDEQUAL, CROWDORDER

Overall task of crowd operators
- Create HITs and HIT groups
- Collect results from AMT
- Quality control through majority voting

Rule-based optimiser
- Batch size, number of assignments and price per HIT fixed
- Predicate push-down, join ordering, delete optimisation
- Better alternative: Cost-based optimiser
Query Processing: Example

Figure 3: CrowdSQL Query Plan Generation

(a) PeopleSQL query
(b) Logical plan before optimization
(c) Logical plan after optimization
(d) Physical plan
Results: Micro-tasks

HITs Group size vs Responsiveness

Tradeoff between HITs completed per unit time and % completion of HITs
Results: Micro-tasks

Reward vs Responsiveness

% of HITs fully completed

% of HITs with at least 1 assignment done
Results: Micro-tasks

- Skewed distribution: Community of fan turkers cultivated
- No variation in error rate between fan turkers and others
Results: Complex queries

Entity resolution on company names

- 4 non-uniform company names match to 100 companies (10 per HIT)
- Majority vote produces correct result always

Ordering pictures in terms of relevance

- Majority voting based ranking matches expert ranking

Joining Professors and Departments

- Method 1: first professor details collected, then department details
- Method 2: denormalized version, professor and department details collected together
- Method 1 outperforms Method 2 in accuracy
- Unclear instructions, turkers submitted professors’ phone numbers

```
SELECT p.name, p.email, d.name, d.phone
FROM   Professor p, Department d
WHERE  p.department = d.name AND
       p.university = d.university AND
       p.name = "[name of a professor]"
```
Other observations

• Relationship of requester with turkers is long-term
  - Keep workers happy
  - Implement less stringent approval standards

• Good interface design and precise instructions matter
  - Simple design changes like adding “None of the above” option can improve quality dramatically.
Related Works

Related work in database systems:
- Oracle forms: Automatic interface generation from metadata
- Dealing with open-world nature using operation limiting, top N optimisations
- Dealing with performance volatility using adaptive query processing

Related work in CrowdSourcing:
- Qurk: SQL plus UDF (Appeared in CIDR’11)
- Crowdsourced database using datalog (Appeared in CIDR’11)
- TurKit, toolkit for programming iterative algorithms in crowd
- Quality control and response time optimisation in CrowdSearch
Thank You!