



Cost-Effective Data-Collection Systems for Citizen Science

Sudarshan S. Chawathe

University of Maine

Motivation

- ▶ *Citizen-science* projects often include data collection by non-experts.
- ▶ They need a data-collection and management system that
 - ▷ can be set up quickly and easily
 - ▷ has low cost
 - ▷ scales easily as usage grows
 - ▷ is easily maintained over long periods of time
- ▶ Few solutions satisfy all these needs.



Web services: opportunities and challenges

- ▶ Popular *DynamoDB* database service: *NoSQL*, *schema-less*.
- ▶ Rudimentary but useful query language.
- ▶ Large variation in service (financial) cost of equivalent queries.
- ▶ Query optimization for cost or other metrics is left to programmer.
- ▶ Query costs are difficult to predict.
 - ▷ complicated cost model.
 - ▷ lack of histograms and other statistical data.



Contributions

- ▶ Formulation of a query-rewriting problem for DynamoDB.
 - ▷ Motivated in part by observations from practitioners (Roussel and Branson, 2017).
- ▶ Preliminary results.
 - ▷ Illustration of the benefits of such query-rewriting.
 - ▷ Method outline.
 - ▶ Algebraic expression of DynamoDB queries.
 - ▶ Enumerations of rewritings.
 - ▶ Cost-estimation of queries using histograms.
 - ▶ Estimation of histograms by probing.
- ▶ Ongoing work.
 - ▷ Generalized algebraic query language.
 - ▷ Translation to DynamoDB queries.
 - ▷ System implementation and experimental evaluation.
 - ▷ Updates.



Cost-estimation problem statement

- ▶ Given
 - ▷ a DynamoDB database scheme
 - ▷ Table definition: partition key, sort key, ...
 - ▷ Secondary indexes: local, global, ...
 - ▷ a query Q (*select-project* on a single table).
- ▶ Find
 - ▷ a rewritten query that is equivalent to Q and has lowest cost.
- ▶ An API-level query optimization problem.
 - ▷ No access to system information.
 - ▷ No system-maintained statistics and histograms.



Sample *DynamoDB* table

- ▶ Table: $R(a, b, c, d, e, f)$
- ▶ Types: $a \dots e$: *number*; f : *string*.
- ▶ *Partition key* a ; *sort key* b .
- ▶ Local Secondary Indexes (LSIs) on c, d, e
 - ▷ c_idx, d_idx, e_idx .



DQL queries

- ▶ DynamoDB Query Language (Arcangeli, 2018).
- ▶ Syntactic sugar on native DynamoDB queries.
- ▶ Conveniences, but no real SQL-like constructs.
 - ▷ Still no multi-table queries, subqueries, ...
- ▶ *DQL-like* queries

```
select *
from R
where a = 1 and b <= 101 and c >= 1001 and
      (d <= 10001 or e >= 100001)
[with table|foo_idx ... ]
```



Sample database statistics

- ▶ 10^7 items of same size.
- ▶ 10^6 with $a = 1$ and of these
 - ▷ 10^5 with $b \leq 101$
 - ▷ 10^4 with $c \geq 1001$
 - ▷ 10^3 with $(b \leq 101 \wedge c \geq 1001)$
- ▶ 10 with $d \leq 10001$ (all happen to have $a = 1$)
- ▶ 1 with $e \geq 100001$ (which also happens to have $a = 1$)



Costs of rewritten queries

- ▶ *scan* of table R . cost: 10^7 .
- ▶ *query* R with only partition-key $a=1$. cost: 10^6 .
- ▶ *query* using index on b . cost: 10^5 .
- ▶ *query* using index on c . cost: 10^3 .
- ▶ *query* using indexes on both d and e by mapping disjunction to union. cost: $10 + 1 = 11$.

```
(select *
 from R
 where a = 1 and b <= 101 and c >= 1001 and d <= 10001
 using d_idx)
union
(select *
 from R
 where a = 1 and b <= 101 and c >= 1001 and e >= 100001
 using e_idx)
```



Summary

- ▶ *Web services* enable low-overhead, low-cost, and scalable systems for data-collection.
- ▶ Careful design is needed to avoid cost overruns.
- ▶ Ongoing work in automating some design aspects:
 - ▷ Automated mapping of database schemes.
 - ▷ Cost-based query rewriting.
 - ▷ Implementation and experiments.



Acknowledgments

- ▶ Supported in part by the US National Science Foundation grants 1027960, 1142007, and 1848747,



References

- Arcangeli, S. (2018). DQL—DynamoDB query language: A simple, SQL-ish language for DynamoDB. System documentation.
- Roussel, A. and Branson, R. (2017). The million dollar engineering problem. segment blog.

