Human-powered Sorts and Joins
by Adam Marcus et al.

SUMMARY:
This paper gives a description of a new declarative workflow engine Qurk based on MTurk which aims to reduce workflow design burden by involving crowds(human) with tasks to compare items for sorting and joining data. This paper also proposes some optimization solutions to lower the overall cost of running sorts and joins on the crowd, including task batching, replacing pairwise comparisons with numerical ratings, and pre-filtering tables before joining. As shown in an experiment of joining two image sets, Qurk dramatically reduces the overall cost from $67 to $3 with little effect on performance. In another end-to-end experiment, Qurk reduces the cost by a factor of 14.5.

COMMENTS:
I am inspired by how this novel workflow engine reduces the cost for both joins and sorts. The paper proposes smart batching to handle joins, a more productive approach than native batching, combined with QA scheme and feature filtering. Moreover, this economically efficient approach barely undermines the result quality and latency. The sort operation is also improved by batching, and the authors did a good job in combining the ratings and comparisons approach, which proves to be both competent and economically efficient.

However, many issues remain unaccounted in this paper: for instance, how to choose the appropriate batch size for a certain join/sort task, which should be efficient without upsetting the workers. The reputation issue should also be considered when deploying this workflow engine, especially when setting price and workload. After all, the reputation of the employer is vital if it wishes to attract turkers continuously.

QUESTION:
How to eliminate and identify workers who generate spam answers, given that QA is limited to only categorical data?
Executing SQL over Encrypted Data in the Database-Service-Provider Model
by Hakan Hacigumus et al.

SUMMARY:
This paper address the data privacy issue of the "database as a service" model. Specifically, it tries to protect data from service providers that cannot be trusted by executing SQL queries over encrypted data. To achieve this, the SQL query is split into two parts: the service provider's site tries to process as much of the query as possible without decrypting the data; then the client site do the decryption and the remaining query processing. This paper explores an algebraic framework to work out the best split solution and minimize the computation at the client site.

COMMENTS:
This paper involves a novel method "coarse index" enabling partial SQL query to be executed on the provider site without decrypting the data, the result of this partial query is then transmitted to the client for further query. Hence the privacy of such data on the database service provider is protected and the client is only responsible for cooperating in the query execution.

My concerns for this approach as follows: first, compared with formal strategies, this approach brings more communication cost and overhead to the clients of this "database service", which would increase as the queries on the data grows. This might contradict to the original principle of this product, which is to take over the task of data management and data query. The second problem concerns the privacy of the transmission process from the server to the client. If this partial query is intercepted by malicious hackers, it might be exploited by them to attain economic reward, which would potentially undermine the confidentiality of the data.

QUESTION:
Since the increasing number of buckets could lead to the decrease of network cost and client-site query execution time, should the buckets number be set as high as possible?