Crowdsourcing Database Systems: Overview and Challenges

Chengliang Chai*, Ju Fan†, Guoliang Li*, Jiannan Wang#, Yudian Zheng‡
*Tsinghua University †Renmin University #Simon Fraser University ‡Twitter
chaicl15@mails.tsinghua.edu.cn, fanj@ruc.edu.cn, liguoliang@tsinghua.edu.cn, jnwang@sfu.ca, yudianz@twitter.com

***Abstract*—Many data management and analytics tasks, such as entity resolution, cannot be solely addressed by automated processes. Crowdsourcing is an effective way to harness the human cognitive ability to process these computer-hard tasks. Thanks to public crowdsourcing platforms, e.g., Amazon Mechanical Turk and CrowdFlower, we can easily involve hundreds of thousands of ordinary workers (i.e., the crowd) to address these computer-hard tasks. However it is rather inconvenient to interact with the crowdsourcing platforms, because the platforms require one to set parameters and even write codes. Inspired by traditional DBMS, crowdsourcing database systems have been proposed and widely studied to encapsulate the complexities of interacting with the crowd. In this tutorial, we will survey and synthesize a wide spectrum of existing studies on crowdsourcing database systems. We first give an overview of crowdsourcing, and then summarize the fundamental techniques in designing crowdsourcing databases, including task design, truth inference, task assignment, answer reasoning and latency reduction. Next we review the techniques on designing crowdsourced operators, including selection, join, sort, top-k, max/min, count, collect, and fill. Finally, we discuss the emerging challenges.**

I. Introduction

Many data management and analytics tasks cannot be effectively solved by existing computer-only algorithms, such as entity resolution [24], [25], [6], [4]. Fortunately, crowdsourcing has been emerged as an effective way to address such tasks by utilizing hundreds of thousands of ordinary workers (i.e., the crowd). Thanks to the public crowdsourcing platforms, e.g., Amazon Mechanical Turk (AMT) and CrowdFlower, the access to the crowd becomes easier.

However, it is rather inconvenient to interact with crowdsourcing platforms, as the platforms require one to set various parameters and even write codes. Inspired by traditional DBMS, crowdsourcing database systems, such as CrowdDB [10], Qurk [18], Deco [20] CrowdOP [9] and CDB [13], [14], have been recently developed. On one hand, these systems provide declarative programming interfaces and allow requesters to use a SQL-like language for posing queries that involve crowdsourced operations. On the other hand, they leverage crowd-powered operations (aka operators) to encapsulate the complexities of interacting with the crowd. Under these design principles, given a SQL-like query from a requester, the systems first parse the query into a query plan with crowd-powered operations, then generate tasks to be published in crowdsourcing platforms, and finally collect the crowd's inputs for producing the result.

Crowdsourcing databases have two main differences from traditional databases. Firstly, traditional databases use "close-world" model, which processes queries based on the data inside database only; while crowdsourcing databases use the "open-world" model, which can utilize the crowd to crowdsource data, i.e., collecting a tuple/table or filling an attribute. Secondly, crowdsourcing databases can utilize the crowd to support operations, e.g., comparing two objects, ranking multiple objects, and rating an object. These two main differences are attributed to involving the crowd to process database operations. In this paper, we review existing works on crowdsourcing database system from the following aspects.

Crowdsourcing Overview. Suppose a requester (e.g., Amazon) has a set of computer-hard tasks (e.g., entity resolution tasks that find the objects referring to the same entity). The requester first designs the tasks. Then the requester publishes her tasks on a crowdsourcing platform, e.g., AMT. Workers who are willing to perform such tasks accept the tasks, answer them and submit the answers back to the platform. The platform collects the answers and reports them to the requester. If a worker has accomplished a task, the requester who publishes the task can approve or disapprove the worker's answers. The approved workers will get paid from the requester.

Crowdsourcing Database Design Techniques. The crowd has some different characteristics from machines. (1) *Not Free*. Workers need to be paid for answering a task, and it is important to control the *cost*. (2) *Error Prone*. Workers may return noisy results, and we need to tolerate the noises and improve the *quality*. (3) *Diverse*. Workers have various background knowledge, leading to different accuracies to answer different tasks. We should capture workers' characteristics to achieve high *quality*. (4) *Dynamic*. Workers are not always online to answer tasks and we need to control the *latency*. Many techniques have recently been proposed to handle these features to redesign database operators and optimization techniques. (i) *Task Design.* We can design different task types to support a crowdsourced operator. For example, in crowdsourced sort, we can ask the crowd to either compare two objects or rank multiple objects. We can select different task design techniques to optimize crowdsourced operators. (ii) *Truth Inference.* To tolerate the noisy results, we can assign each task to multiple workers, model the workers' quality, and infer the results by aggregating the answers. (iii) *Task Assignment.* We assign appropriate tasks to workers and make full use of workers' unique talents. (iv) *Answer Reasoning.* We can model the tasks and deduce the answers of tasks based on those of other tasks. For example, in crowdsourced join, if we get crowd's answers that "US = United States" and "US = America", we can deduce that "United States = America".

[1]Guoliang Li is the corresponding author.

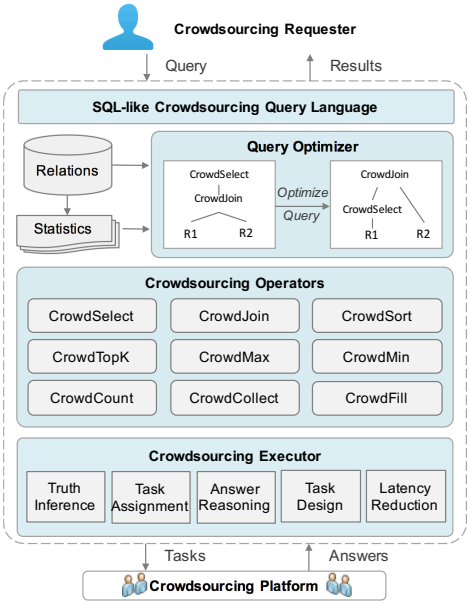

Fig. 1: Architecture of Crowdsourcing DB Systems.

		CrowdDB	Qurk	Deco	CrowdOP	CDB
Crowd Powered Operators	CrowdSelect	√	√	√	√	√
	CrowdJoin	√	√	√	√	√
	CrowdSort	√	√	×	×	√
	CrowdTopK	√	√	×	×	√
	CrowdMax	√	√	×	×	√
	CrowdMin	√	√	×	×	√
	CrowdCount	×	×	×	×	√
	CrowdCollect	√	×	√	×	√
	CrowdFill	√	×	√	√	√
Optimization Objectives	Cost	√	√	√	√	√
	Latency	×	×	×	√	√
	Quality	√	√	√	√	√
Design Techniques	Truth Inference	√	√	√	√	√
	Task Assignment	×	×	×	×	√
	Answer Reasoning	×	×	×	×	√
	Task Design	√	√	√	√	√
	Latency Reduction	×	×	×	×	√

Fig. 2: Comparison of Crowdsourcing DB Systems.

(v) *Latency Reduction.* We can model workers' behavior and design effective models to reduce the latency.

Crowdsourcing Systems & Operators. Using the aforementioned techniques, recent efforts have been made to develop crowdsourcing database systems, such as CDB [13], CrowdDB [10], Qurk [18], Deco [20], and CrowdOP [9]. For achieving high crowdsourcing query processing performance, the systems focus on optimizing cost (cheap), latency (fast) and quality (good). Moreover, there are also techniques that focus on designing individual crowdsourced operators, including selection [22], [26], join [24], [25], top-k/sort [7], [17], aggregation [16], [12], and collect [23], [21], [2].

• **Tutorial Structure.** We can do both 1.5 and 3 hours tutorial but prefer the 3 hours' one. The 3 hours' tutorial is split into 2 sections. In the first section (1.5 hours), we first give an overview of crowdsourcing (20 min), including motivation of crowdsourcing, basic concepts (e.g., workers), crowdsourcing platforms, crowdsourcing workflow, and crowdsourcing applications. Then we talk about an overview of crowdsourcing database systems (20 min, see Section II), and fundamental techniques in designing crowdsourced operators, including task design (10 min), truth inference (10 min), task assignment (10 min), answer reasoning (10 min), and latency reduction (10 min). In the second section (1.5 hours), we first discuss different crowdsourced operators (60 min), e.g., *selection, join, topk, sort, max/min, count, collect, fill*. Finally we provide emerging challenges (15 min). We leave 15 min for Q&A to interact with the tutorial audience. If we have to do the 1.5 hours' one, we tend to remove the section about operators(60 min), reduce the time of challenges and Q&A to 10 min in total and remove the latency reduction(10 min).

• **Tutorial Audience.** The intended audience includes all ICDE attendees from research and industry communities. We will not require any prior background knowledge and a basic understanding of database (e.g., selection, join) will be helpful.

• **Differences from Existing Tutorials.** There are existing crowdsourcing tutorials (e.g., in KDD'18 [3], VLDB'16 [1], VLDB'15 [11], ICDE'15 [5], VLDB'12 [8], SIGMOD'17 [15]). VLDB'16 [1] investigates human factors involved in task assignment and completion. VLDB'15 [11] focuses on truth inference in quality control. ICDE'15 [5] reviews some crowdsourcing operators, crowdsourced data mining and social applications. VLDB'12 [8] introduces crowdsourcing platforms and discusses general design principles for crowdsourced data management. SIGMOD'17 [15] focuses on quality, cost, and latency control for crowdsourced data management. KDD'18 [3] focuses on different applications and operations in crowd-powered data mining. Compared with these tutorials, we focus on the fundamental techniques for building a practical crowdsourced database system. Moreover, we systemically review crowdsourcing operators and optimization techniques that are proposed in recent five years.

II. System Design Overview

Several crowdsourcing database systems [10], [20], [18], [9], [13] are recently proposed to encapsulate the complexities of leveraging the crowd for query processing. In this part, we introduce an overview of the design of these systems.

• **Data model.** Existing crowdsourcing database systems are built on top of the traditional *relational* data model, where data is specified as a schema that consists of relations and each relation has a set of attributes. The difference is that crowdsourcing database systems employ an *open-world* assumption that either some attributes of a tuple or even an entire tuple can be crowdsourced based on queries from the requester.

• **Query language.** Most crowdsourcing query languages follow the standard SQL syntax and semantics, and extend SQL by adding features that support crowdsourced operations, e.g., asking the crowd to perform data processing operations.

• **Architecture.** The architecture of a typical crowdsourcing database system is illustrated in Figure 1. A SQL-like query is issued by a crowdsourcing requester and is first processed by a Query Optimizer. Like traditional databases, the Query Optimizer parses the query into a tree-structure query plan, and then applies optimization strategies to produce an *optimized* query plan. However, the key difference is that the tree nodes in a query plan are *crowd-powered operators*. Typically, a crowd-powered operator abstracts a specific type

of operation that can be processed by the crowd. Figure 2 shows how operators are supported by the existing systems.

Crowd-powered operators are then executed by CROWDSOURCING EXECUTOR to generate human-intelligent tasks (HITs) and publish the HITs on crowdsourcing platforms (e.g., AMT). Next, after collecting answers from the crowd, the executor evaluates the query plan and returns the final result to the requester. To this end, the executor employs several crowdsourcing data processing techniques, e.g., truth inference, task assignment, answer reasoning, task design, and latency reduction. Figure 2 illustrates how the systems implement these techniques, with the details in Section III.

• **Optimization.** Query optimization is indispensable in crowdsourcing database systems, as the difference of various query plans may be several orders of magnitude. It is worth noting that crowdsourcing optimization is more challenging than that of traditional databases, because it needs to optimize multiple objectives, including quality control, cost control, and latency control. It is desirable for a system to support "multi-objective" optimization, as any single optimization may not satisfy requester's needs. Figure 2 compares the existing systems regarding their capabilities of supporting optimization.

III. CROWDSOURCING OPERATORS

A. Design Techniques

• **Truth Inference.** Crowdsourcing may yield relatively low-quality results or even noise and *Truth Inference* aims to infer the correct answer (called truth) of each task given multiple workers' answers. Existing studies first build worker model to estimate workers' quality, and then infer the truth and workers' quality based on the following intuitions: (1) a worker is of high quality if her answer is close to the truth; (2) a task's answer is highly probable to be the truth if the answer is given by a high quality worker [27].

• **Task Assignment.** Workers have diverse qualities on different tasks, and *Task Assignment* aims to wisely assign tasks to workers within a given cost budget. When a worker requests for tasks, existing works will estimate the gain of assigning each task to the worker (first estimating the worker answer to this task based on the collected answers and then computing the quality improvement), and assign the task with the highest gain of improvement [27].

• **Answer Reasoning.** In many cases, the tasks generated by crowdsourced operators have inherent relationships, *Answer Reasoning* aims to deduce the answers of some tasks (without needing to crowdsource these tasks) based on answers of crowdsourced tasks. For example, suppose a crowdsourced join operator generates three tasks: (A, B), (B, C), and (A, C). If we have already known that A is equal to B, and B is equal to C, then we can deduce that A is equal to C based on transitivity, thereby avoiding the cost for checking (A, C).

• **Task Design.** *Task Design* focuses on designing effective task types to optimize crowdsourced operators. For example, [24] proposes two task types to optimize crowdsourced join: (1) pair-based task asks workers to identify whether two given objects refer to the same real-world entity; (2) cluster-based task asks workers to group entities into different clusters.

TABLE I: Crowdsourced Operators

Crowdsourced Operators		Techniques
CrowdSelect	Filtering [19]	Truth Inference, Task Assignment Latency Reduction
	Find [22]	Truth Inference, Task Assignment Latency Reduction
	Search [26]	Truth Inference, Task Assignment Latency Reduction
CrowdJoin	CrowdER [24]	Truth Inference, Task Assignment Answer Reasoning, Task Design
	Transitivity [25]	Truth Inference, Task Assignment Answer Reasoning
CrowdSort/CrowdTopk	Heuristics [12]	Truth Inference, Task Assignment Answer Reasoning
	Heap [7]	Truth Inference, Task Assignment Answer Reasoning
	Hybrid [17]	Truth Inference, Task Assignment Answer Reasoning
CrowdMax/CrowdMin	[12]	Truth Inference, Task Assignment Answer Reasoning
CrowdCollect	[23]	Truth Inference
CrowdFill	[21]	Truth Inference, Task Assignment Latency Reduction, Task Design
CrowdCount	[16]	Truth Inference, Task Assignment Task Design

• **Latency Reduction.** In many cases, requesters have latency requirement and *Latency Reduction* aims to reduce the latency. There are several ways in latency reduction: (1) setting each HIT with a higher price to reduce the latency in collecting answers; (2) leveraging the round/statistical model to capture the latency in workers' answering tasks; and (3) devising strategies (e.g., dynamically maintaining a pool of fast workers) to improve the latency.

B. Operator Design

Existing works focus on using the above design techniques to implement crowd-powered operators to optimize cost, quality, and latency. Table I summarizes how the crowdsourced operators are implemented based on various techniques.

• **CrowdSelect.** Given a set of items, crowdsourced selection identifies items that satisfy a set of constraints, e.g., selecting images that have both mountains and humans. Existing works can be classified into three categories: (1) Crowd Filtering [19] (or All-Selection) returns all items that satisfy the given constraints; (2) Crowd Find [22] (or k-Selection) returns k items that satisfy the given constraints; (3) Crowd Search [26] (or 1-Selection) returns only one item that satisfies the given constraints. They focus on finding the answers within a cost or latency constraint.

• **CrowdJoin.** Existing crowdsourcing works mainly focus on Equi-Join. Given a table (or two tables), a crowdsourced Equi-Join is to find all record pairs in the table (or between two tables) that refer to the same entity. It is rather expensive to enumerate every pair to ask the crowd, and existing crowdsourcing works focus on designing user-friendly interfaces [24] or leveraging transitivity relations [25] to reduce the cost while keeping high quality.

• **CrowdSort/Topk.** Given a set of items which are comparable but are hard to be compared by machines, CrowdTopk (or Sort) aims to find top-k items (or a ranking list) based on a certain criterion, e.g., understanding difficulty of sentences, clarity of images. The challenges include tolerating the comparison error and reducing the cost.

• **CrowdMax (CrowdMin).** Crowdsourced Max [12] is a special case of CrowdTopk where $k = 1$, which finds the

max item in a dataset, e.g., finding the most beautiful picture about Great Wall.

• **CrowdCount.** Crowdsourced Count [16] is to count the number of items in a dataset that satisfy a given constraint, e.g., counting the number of birds in a picture. Existing works focus on designing effective task types and devising unbiased sampling estimator.

• **CrowdCollect.** Different from the above query operators which perform queries on a given set of known items, CrowdCollect [23] tries to collect the *unknown* items from the crowd, e.g., enumerating the top-100 universities in US. It focuses on improving the coverage of the collected items.

• **CrowdFill.** CrowdFill [21] focuses on asking the crowd to fill the cells in a table. For example, given a table that shows the statistics of football players, it asks workers to fill missing cells (e.g., the position of *Messi*). It focuses on achieving high quality, while without taking large cost and latency.

IV. Crowd System Challenges

Query Optimization. A SQL query often corresponds to multiple query plans and it relies on a query optimizer to select the best plan. Existing optimizer estimates the computation cost of each query plan and chooses the one with the minimum estimated cost. However, this process turns to be quite challenging in a crowdsourcing environment because (1) there are three optimization objectives (result quality, monetary cost, and latency) that need to be considered and (2) humans are much more unpredictable than machines.

Benchmark. A large variety of TPC benchmarks standardize performance comparisons for database systems and promote the development of database research. Although there are some open datasets (http://dbgroup.cs.tsinghua.edu.cn/ligl/crowddata), there is still lack of standardized benchmarks available. In order to better explore the research topic, it is important to study how to develop evaluation methodologies and benchmarks for crowd db systems.

Crowdsourcing Indexing. Crowdsourcing indexing will be useful for many crowdsourced operators, such as join and sort. However, it is non-trivial to build such index, because crowd-powered comparisons are error-prone and building index also incurs cost. Therefore, it calls for techniques to reduce the building cost while preserving the index accuracy.

Acknowledgement. This work was supported by the 973 Program of China (2015CB358700), NSF of China (61632016, 61521002, 61661166012), Huawei, and TAL education.

References

[1] S. Amer-Yahia and S. B. Roy. Human factors in crowdsourcing. *Proceedings of the VLDB Endowment*, 9(13):1615–1618, 2016.

[2] C. Chai, J. Fan, and G. Li. Incentive-based entity collection using crowdsourcing. ICDE, 2018.

[3] C. Chai, J. Fan, G. Li, J. Wang, and Y. Zheng. Crowd-powered data mining. *CoRR*, abs/1806.04968, 2018.

[4] C. Chai, G. Li, J. Li, D. Deng, and J. Feng. A partial-order-based framework for cost-effective crowdsourced entity resolution. *The VLDB Journal*, pages 1–26, 2018.

[5] L. Chen, D. Lee, and T. Milo. Data-driven crowdsourcing: Management, mining, and applications. In *ICDE*, pages 1527–1529. IEEE, 2015.

[6] S. Das, P. S. G. C., A. Doan, J. F. Naughton, G. Krishnan, R. Deep, E. Arcaute, V. Raghavendra, and Y. Park. Falcon: Scaling up hands-off crowdsourced entity matching to build cloud services. In *SIGMOD*, pages 1431–1446, 2017.

[7] S. B. Davidson, S. Khanna, T. Milo, and S. Roy. Using the crowd for top-k and group-by queries. In *ICDT*, pages 225–236, 2013.

[8] A. Doan, M. J. Franklin, D. Kossmann, and T. Kraska. Crowdsourcing applications and platforms: A data management perspective. *Proceedings of the VLDB Endowment*, 4(12):1508–1509, 2011.

[9] J. Fan, M. Zhang, S. Kok, M. Lu, and B. C. Ooi. Crowdop: Query optimization for declarative crowdsourcing systems. *IEEE Trans. Knowl. Data Eng.*, 27(8):2078–2092, 2015.

[10] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. Crowddb: answering queries with crowdsourcing. In *SIGMOD*, pages 61–72, 2011.

[11] J. Gao, Q. Li, B. Zhao, W. Fan, and J. Han. Truth discovery and crowdsourcing aggregation: A unified perspective. *Proceedings of the VLDB Endowment*, 8(12):2048–2049, 2015.

[12] S. Guo, A. G. Parameswaran, and H. Garcia-Molina. So who won?: dynamic max discovery with the crowd. In *SIGMOD*, pages 385–396, 2012.

[13] G. Li, C. Chai, J. Fan, X. Weng, J. Li, Y. Zheng, Y. Li, X. Yu, X. Zhang, and H. Yuan. CDB: optimizing queries with crowd-based selections and joins. In *SIGMOD*, pages 1463–1478, 2017.

[14] G. Li, C. Chai, J. Fan, X. Weng, J. Li, Y. Zheng, Y. Li, X. Yu, X. Zhang, and H. Yuan. CDB: A crowd-powered database system. *PVLDB*, 11(12):1926–1929, 2018.

[15] G. Li, Y. Zheng, J. Fan, J. Wang, and R. Cheng. Crowdsourced data management: Overview and challenges. In *SIGMOD*, 2017.

[16] A. Marcus, D. R. Karger, S. Madden, R. Miller, and S. Oh. Counting with the crowd. *PVLDB*, 6(2):109–120, 2012.

[17] A. Marcus, E. Wu, D. R. Karger, S. Madden, and R. C. Miller. Human-powered sorts and joins. *PVLDB*, 5(1):13–24, 2011.

[18] A. Marcus, E. Wu, S. Madden, and R. C. Miller. Crowdsourced databases: Query processing with people. In *CIDR*, pages 211–214, 2011.

[19] A. G. Parameswaran, H. Garcia-Molina, H. Park, N. Polyzotis, A. Ramesh, and J. Widom. Crowdscreen: algorithms for filtering data with humans. In *SIGMOD*, pages 361–372, 2012.

[20] H. Park, R. Pang, A. G. Parameswaran, H. Garcia-Molina, N. Polyzotis, and J. Widom. Deco: A system for declarative crowdsourcing. *PVLDB*, 2012.

[21] H. Park and J. Widom. Crowdfill: collecting structured data from the crowd. In *SIGMOD*, pages 577–588, 2014.

[22] A. D. Sarma, A. G. Parameswaran, H. Garcia-Molina, and A. Y. Halevy. Crowd-powered find algorithms. In *ICDE*, pages 964–975, 2014.

[23] B. Trushkowsky, T. Kraska, M. J. Franklin, and P. Sarkar. Crowdsourced enumeration queries. In *ICDE*, pages 673–684, 2013.

[24] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. CrowdER: crowdsourcing entity resolution. *PVLDB*, 5(11):1483–1494, 2012.

[25] J. Wang, G. Li, T. Kraska, M. J. Franklin, and J. Feng. Leveraging transitive relations for crowdsourced joins. In *SIGMOD*, 2013.

[26] T. Yan, V. Kumar, and D. Ganesan. Crowdsearch: exploiting crowds for accurate real-time image search on mobile phones. In *MobiSys*.

[27] Y. Zheng, G. Li, Y. Li, C. Shan, and R. Cheng. Truth inference in crowdsourcing: Is the problem solved? *PVLDB*, 2017.

Biography

Chengliang Chai is a PhD student at the Department of Computer Science, Tsinghua University, China. His research interests include data integration and crowdsourcing (especially crowdsourced data integration).

Ju Fan is currently working as an associate professor at the Department of Computer Science in Renmin University, Beijing, China. His main research interests include knowledge base and crowdsourcing (especially crowdsourcing systems).

Guoliang Li is currently working as a professor at the Department of Computer Science, Tsinghua University, Beijing, China. His research interests mainly include data cleaning and integration, and crowdsourcing (especially crowdsourcing database system and optimization).

Jiannan Wang is currently working as an assistant professor at the School of Computing Science in Simon Fraser University, Canada. His main research interests include data cleaning and crowdsourcing (especially crowdsourcing optimization).

Yudian Zheng is now a Machine Learning scientist at Twitter. His research interests include crowdsourced data management (especially truth inference and task assignment).