Reprowd: Crowdsourced Data Processing Made Reproducible

Jiannan Wang
Simon Fraser University
Burnaby, Vancouver, Canada
jnwang@sfu.ca

ABSTRACT

Crowdsourcing is a multidisciplinary research area including disciplines like artificial intelligence, human-computer interaction, database, and social science. To facilitate cooperation across disciplines, *reproducibility* is a crucial factor, but unfortunately it has not gotten enough attention in the crowdsourcing research community.

Imagine a researcher Bob did a crowdsourcing experiment, and another researcher Ally would like to reproduce the experiment (Note that in this paper, we trust the crowdsourced answers collected by Bob, which is a weaker claim of reproducibility than [4].). This reproducibility process could take a lot of time for both Bob and Ally. From the Bob's perspective, he has to spend additional time in modifying the code since the code written for doing the experiment is different from that used for reproducindog the experiment. For example, the former requires to collect answers from crowd workers, but the latter just reuses the cached crowdsourced answers. From the Ally's perspective, once she receives the Bob's code and crowdsourced answers, she might find it hard to examine the experimental result since the code may not be easy to extend or the crowdsourced answers may not contain enough lineage information (e.g., when were the tasks published? which workers did the tasks?).

These issues discourage researchers from sharing experimental results or analyzing others' results to derive new insights, resulting in a significant negative impact on the crowdsourcing research community. While some recently developed tools [3, 1] can be used to mitigate the impact of these issues, they do not fully meet Bob and Ally's requirements. Reprozip [1] is tool for automatically packing an experiment along with the entire programming environment (e.g., dependent libraries or packages). It saves the time for deploying an experiment but not the time that Bob spends in modifying the code or Ally spends in examining the code. Turkit [3] proposes a crash-and-rerun programming model. This programming model can help Bob to solve his issue, but add extra burden to Ally. The reason is that Turkit caches the functions' returned values into a database *in sequence*, thus Ally has to be very careful with the order of these function calls. If she accidentally swapped the order of two functions or added a new function between them, the whole experiment would break.

In this paper, we present REPROWD, a system aiming to address these issues. We identify two requirements for making a crowdsourcing experiment easy to reproduce. 1) Sharable. Once Bob finishes a crowdsourcing experiment, he should be able to directly share the experiment to Ally without any need to change the code. (2). Examinable. The experiment should capture complete lineage information about how crowdsourced answers were collected and allow Ally to extend the code more easily.

We restrict the current focus of the system on the database field only. Most of the crowdsourcing works in the database field are centered around the implementations of crowdsourced data processing operators [2]. That is, how to combine computers and crowds to implement traditional database operators such as join, sort, and max. Despite the restricted focus, REPROWD is actually beneficial to any research field that needs to collect data from the crowd.

A key insight in designing REPROWD is to model a list of steps for doing a crowdsourcing experiment as a sequence of manipulations of a tabular dataset called *CrowdData*. It enables us to leverage existing techniques that were originally developed for data management, such as data recovery and data lineage, to address reproducibility challenges. Specifically, in order to satisfy the "sharable" requirement, the system guarantees that any manipulation of CrowdData is fault recovery. That is, when the program is crashed, rerunning the program is as if it has never crashed. Thus, at any given point, Ally can simply rerun the Bob's code (without any modification) to reproduce his experimental result. In order to satisfy the "examinable" requirement, CrowdData not only contains complete lineage information about crowdsourced answers but also allow other researchers to extend it using the provided APIs. We find that the CrowdData programming model is general enough to be used in re-implementing a large number of existing crowdsourced data processing algorithms in the literature. We have implemented two crowdsourced join algorithms [5, 6] based on CrowdData. A preliminary result suggests that the implementations of high-level operators (e.g., join, sort, max) can easily inherit the sharable and examinable properties from CrowdData. We have open sourced REPROWD at http://sfu-db.github.io/reprowd/, and will continue implement more crowdsourced data processing operators using CrowdData in the future.

1. REFERENCES

[1] F. Chirigati, R. Rampin, D. Shasha, and J. Freire. Reprozip: Computational reproducibility with ease. In *SIGMOD*, pages 2085–2088, 2016.
[2] G. Li, J. Wang, Y. Zheng, and M. J. Franklin. Crowdsourced data management: A survey. *IEEE TKDE*, 28(9):2296–2319, Sept 2016.
[3] G. Little, L. B. Chilton, M. Goldman, and R. C. Miller. Turkit: human computation algorithms on mechanical turk. In *UIST*, pages 57–66, 2010.
[4] P. Paritosh. Human computation must be reproducible. In *CrowdSearch*, pages 20–25, 2012.
[5] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. Crowder: Crowdsourcing entity resolution. *PVLDB*, 5(11):1483–1494, 2012.
[6] J. Wang, G. Li, T. Kraska, M. J. Franklin, and J. Feng. Leveraging transitive relations for crowdsourced joins. In *SIGMOD*, pages 229–240, 2013.