Platform Considerations in Human Computation

Adam Marcus, Eugene Wu,
David R. Karger, Samuel Madden, Robert C. Miller
MIT CSAIL
32 Vassar St., Cambridge MA
{marcua, sirrice, karger, madden, rcm}@csail.mit.edu

ABSTRACT

With the recent growth of interest in human computation, the number of crowdsourcing platforms and corresponding workflows has been growing rapidly. This presents problems for both platform developers, who reimplement the same building blocks with each new platform, and human computation workflow developers, who must cope with the increasing complexity of tasks that may span multiple platforms. In this paper, we describe two systems designed to alleviate these pain points. Qurk is a system that lets developers describe workflows in a high-level declarative language, and which automatically optimizes the workflow across multiple platforms. Djurk is an open source human computation platform that is both usable out-of-the-box, and provides a feature-rich starting point for building new crowdsourcing platforms.

Author Keywords

Human Computation, MTurk, Mechanical, Turk, Databases

AUTHORS' EXPERIENCE

The authors are currently building a tool called Qurk [12], a declarative workflow management system that allows human computation over relational databases. Qurk makes human computation a first-class citizen in data processing, bringing with it the benefits of decades of research by the database community in workflow execution and optimization.

The authors are also in the planning stages of a system called Djurk, a generalized crowdsourcing platform in which systems such as Mechanical Turk or ODesk could be implemented. The first platform we will build using Djurk is designed for the data journalism community, but we hope that it can be the starting point for other platform developers.

In this paper, we describe our vision for human computation platforms, combining numerous competing platforms built from a pluggable set of building blocks, and a declarative workflow system which can span these platforms to make data processing using humans easier.

INTRODUCTION

Today's human computation tasks can be sent to multiple varied platforms including MTurk, ODesk, and reCAPTCHA. Each platform is geared toward different tasks, provides crowd workers with different skills, and has different notions of crowd worker reputation and incentives. With choice comes complexity, complicating the already difficult task of optimizing human computation workflows for parameters such as worker compensation, accuracy, and time to task completion. We first present a system called Qurk that can automatically optimize these choices without extra work on behalf of the workflow developer. We then propose a system called Djurk, aimed at providing human computation platform developers an easier starting point for building new crowdsourcing platforms.

DECLARATIVE WORKFLOW MANAGEMENT

Patterns such as Find-Fix-Verify [2] show that interesting crowdsourced tasks are a multi-stage workflow of related tasks, rather than carefully-crafted one-off tasks. We have found that even with tools such as TurkIt [10], building such workflows on a single platform is a non-trivial process. A large portion of human computation workflow code is devoted to retrieving from and storing data in a storage system, verifying human results, and hand-optimizing parameters such as task price or the number of workers per task. As the diversity of human computing platforms increase, implementing workflows that run across all of them will only become more difficult.

As database researchers, we have noticed that these problems parallel the traditional data management problems that databases alleviate through the use of declarative languages. Declarative queries allow users to describe *what* they want, rather than specify *how* the computation should be performed. We believe that by bringing human computation workflows into the data processing regime provided by traditional database systems, the complexities of task and workflow specification can be reduced.

To explore the integration of human computation and database principles, we have built Qurk [12]. Qurk users build complex workflows in a declarative language (currently SQL) that allows the user to query for data from human computation platforms much in the same way they can query a traditional database table. Qurk compiles and runs the human computation tasks on the ideal platform for the task (at the moment, we support MTurk). This approach has two key benefits. First, because programmers do not specify the details of query implementation, there is an opportunity for automatic optimizations which affect the latency, cost, or quality of the results. Second, the approach eases the developer's burden of implementing low-level optimization details.

While there are a number of interesting systems details, we refer the reader to [12] and focus the rest of the section on exciting optimization opportunities.

Optimizations

In addition to traditional database optimization parameters such as resource consumption and query latency, Qurk provides parameters for monetary cost and result accuracy. The optimizations that we outline fall into three categories: 1) Task Performance, 2) Task Avoidance, and 3) Platform Selection.

Task Performance

Task performance optimizations are concerned with improving the latency, quality, or reducing the cost of running tasks on human computation platforms.

Runtime Pricing. If it appears that one type of task takes longer to complete than others, Qurk can offer more money for it in order to get a faster result. We can borrow from techniques such as adaptive query processing [7] to accomplish this goal.

Input Sampling. For large tables that could lead to many HITs on systems like MTurk, Qurk attempts to sample the input tables to generate Qurk jobs that uniformly cover the input space. This is similar to issues that arise in online query processing [9].

Batch Predicates. When a query contains two filters on the same table, we can combine them into a single HIT, decreasing cost. For example, a query with predicates `imgColor(image) == 'Red'` and `imgLabel(image) == 'Car'` could present crowd workers with the `imgColor` and `imgLabel` tasks at the same time.

Batch Records. Similarly, if tasks such as image filtering are simple for workers to perform, we can reduce cost in exchange for increased worker latency if we send multiple images to filter with each task.

Avoid Worker Fatigue. Workers may be tasked with needle-in-a-haystack tasks, such as identifying the location of a boat lost at sea from satellite imagery. Workers may experience fatigue in such cases, missing the needle after a series of haystack examples. We can remove skew from the data by injecting known matches into the stream of items workers see to keep them on their toes for potential matches.

Join Heuristics. The space of comparisons required for join predicates for having turkers combine data from two tables can be reduced with a preprocessing step identifying necessary join conditions. A naïve join implementation would require quadratic comparisons between all records of each table. We can avoid some of these expensive operations by having workers cluster like items together, and only perform quadratic join comparisons within clusters.

Task Avoidance

The following optimizations reduce the number of tasks sent to human computation platforms.

Task Result Cache. Once a HIT has run, its results might be relevant in the future. For example, if a `products` table has already been ranked in one query, and another query wishes to rank all red `products`, the result of the old task output can be used. Additionally, as explored in TurKit [10], queries might crash midway, or might be iteratively developed. In such scenarios, a cache of completed HITs can improve response time and decrease costs.

Model Training. Recent work by Wais et al. [13] suggests that there are cases where simple machine learning models might be superior to crowdworker results for certain budgets and tasks. Given that learning models and crowd workers excel in different scenarios, we see two optimization opportunities. First, where possible, we can use gold standard data or worker output to train models, which, if they achieve a certain accuracy, may eventually replace workers on future tasks. Second, the decision to send a task to the crowd or to a learning model is an active learning problem of its own.

Platform Selection

The future of human computation features a multitude of platforms and crowds which are ideal for solving different tasks. If Qurk is aware of the various platforms' configurations, it can split workflows into tasks to be assigned to different platforms. For example, when compiling a task to find similar images of flowers from two tables of plant images, Qurk can first send images to be labelled by an image labelling game, and then send pictures of flowers to MTurk for similarity comparisons. As users change their preferences between money, accuracy, and task completion time, Qurk can adjust the platform that receives each task.

NEW CROWDSOURCING PLATFORMS

Qurk makes the process of building human computation workflows on existing crowdsourcing platforms simpler and more efficient for requesters. There are cases, however, where existing platforms do not meet the needs of requesters. We are designing an extensible open source crowdsourcing platform called Djurk that will allow platform designers to easily implement platforms with characteristics optimized for different worker tasks.

There are several motivations for Djurk. First, many organizations are already implementing custom human computation platforms (e.g., The Guardian's MP expenses platform[1]), and re-implement the same building blocks as other crowdsourced platforms. An open implementation of these blocks can allow organizations to innovate toward features unique to their tasks. Second, a hosted public service may

[1]`http://mps-expenses.guardian.co.uk/`

simply not be usable, due to data sensitivity or expertise requirements. For example, the New York Times was limited in how much of the WikiLeaks Iraq dataset it could deduplicate by hand before a deadline, but the data was too sensitive to be placed on Mechanical Turk[2]. Our first goal with Djurk is to build a data journalism-oriented crowdsourcing platform that is easily deployable behind a firewall.

In the process of designing Djurk, we compiled an (incomplete) list of important platform characteristics:

Identity. Requesters are currently identified by name, whereas workers are given an opaque unique identifier. In order to assist with platform portability, both sides of the table should be provided with identities through protocols like OpenID [3]. These identities would also be connected to reputation and provenance profiles, which would help requesters and workers better decide which tasks to perform.

Reputation. Requesters desire some knowledge of the quality of work performed by potential workers. Workers similarly wish to work for requesters with a history of fair treatment and fast payment turnaround. MTurk currently offers requesters a coarse-grained notion of turker quality, and turkers have created their own message boards to identify requesters of note.

Requesters and workers can benefit from better reputation systems. For example, a platform can provide more detail into task types at which a worker excelled. To assist workers in traversing the various specialized human computation platforms, worker reputation profiles should be portable across human computation platforms so that they can establish themselves as dependable quickly on a platform geared toward a specific task.

Provenance. There are many crowdsourced tasks for which accountability is a key factor. For example, outsourced insurance form processing requires a provenance trail identifying which workers had access to sensitive information. In these situations, the rich body of research into database provenance can assist requesters in digging into the output of a crowd-powered workflow. Provenance would help identify and reverse problems such a faulty task description or worker when they surface.

Incentive. MTurk incentivises workers through monetary compensation, while other systems incentivize task completion with gameplay. Chandler and Kapelner identified that incentives other than money, such as context, can motivate turkers [5]. Given the multitude of incentive systems, human computation platforms should offer a number and combination of incentives to workers.

Worker Interface. MTurk relies on a task completion interface which is form-based, and generally relies on workers being at a PC to complete tasks. txteagle [8] shows that this interface could be mobile, and outsourced programming tasks on ODesk would ideally provide an interface which is suited for program development. Djurk could provide specialized interfaces and contexts in which workers can complete tasks.

Requester Interface. Task generation can start in many forms, ranging from uploading a csv file in MTurk, to using a phone to ask a question on ChaCha. For task definition to move to the mainstream, we envision a community-maintained collection of templates for various tasks, providing *crowdsourcing by example* functionality to requesters.

Requester/Worker Coordination. As we reconsider worker and requester interfaces, we can also reconsider their interactions. Rather than transactional request-reponse tasks, a crowdsourcing platform should provide iterative refinement of task definition and worker output. Crowdsourcing platforms which result in creative output, such as ODesk already provide such mechanisms. Additionally, providing workers with the ability to discuss task completion with other workers might facilitate larger-scale task definition and coordination.

Both forms of coordination might be synchronous or asynchronous. For example, a logo design crowdsourcing platform might allow asynchronous out-of-band communication for iterative design cycles. A crowdsourced development platform mgiht allow workers to coordinate through synchronous chat for pair-programming sessions.

Requester/Worker API. One complaint we often hear is that MTurk's interface does not empower requesters or workers enough to define, find, or perform tasks most effectively. To ensure that Djurk's interface does not hinder either party, a rich API should be available to support task creation, task search through novel interfaces.

Finally, Qurk can also perform more powerful optimizations if the platforms expose the above characteristics, as well as statistics about its worker population, such as expertise, latency and costs. For example, by know the expertise of workers on different platforms, Qurk can better route tasks across these platforms.

CONCLUSION

As the number of human computation platforms and crowdsourced data processing needs increase, the need for systems to simplify workflow creation will continue to grow. To this end, we presented two systems to alleviate the problems for both platform and workflow developers. On the platform side, we introduced Djurk, an open source platform that can easily be extended for specialized needs. We also presented Qurk, a system for declaratively specifying data processing human computation workflows, which optimizes such workflows on behalf of the developer.

AUTHOR BIOGRAPHIES

Adam Marcus and Eugene Wu are graduate students at MIT's CSAIL. Adam's focus is on the intersection between

[2] `http://www.cjr.org/the_news_frontier/visualizing_the_iraq_war_logs.php?page=all`
[3] `http://openid.net/`

social computing and database systems. His most relevant publications include TwitInfo [11], a system for crowdsourcing news from microblogs, FeedMe [3], a system for friendsourcing web-based content sharing, and DataPress [1], a system for facilitating data- and vizualization-oriented discussions in blog entries. He recently presented Qurk at the Conference on Innovative Database Research [12], and has published more database performance-oriented papers in VLDB and WWW. Eugene studies database storage and query performance, as well as provenance in diverse data workflows. He has published WebTables [4], a system for extracting data from from human-generated tables, and has built systems such as SASE [14] for stream data processing, TrajStore [6] for location-based data, and Shinobi [15] for high-skew workloads.

REFERENCES

1. E. Benson, A. Marcus, F. Howahl, and D. R. Karger. Talking about data: Sharing richly structured information through blogs and wikis. In *International Semantic Web Conference (1)*, pages 48–63, 2010.
2. M. S. Bernstein et al. Soylent: a word processor with a crowd inside. In *UIST '10: Proceedings of the 23nd annual ACM symposium on User interface software and technology*, pages 313–322, New York, NY, USA, 2010.
3. M. S. Bernstein, A. Marcus, D. R. Karger, and R. C. Miller. Enhancing directed content sharing on the web. In *CHI*, pages 971–980, 2010.
4. M. J. Cafarella, A. Y. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. Webtables: exploring the power of tables on the web. *PVLDB*, 1(1):538–549, 2008.
5. D. Chandler and A. Kapelner. Breaking monotony with meaning: Motivation in crowdsourcing markets. Technical report, University of Chicago, 2010.
6. P. Cudré-Mauroux, E. Wu, and S. Madden. Trajstore: An adaptive storage system for very large trajectory data sets. In *ICDE*, 2010.
7. A. Deshpande, Z. G. Ives, and V. Raman. Adaptive query processing. *Foundations and Trends in Databases*, 1(1):1–140, 2007.
8. N. Eagle. txteagle: Mobile crowdsourcing. In *Internationalization, Design and Global Development*. Springer, 2009.
9. P. J. Haas et al. Selectivity and cost estimation for joins based on random sampling. *J. Comput. Syst. Sci.*, 52(3):550–569, 1996.
10. G. Little et al. Turkit: human computation algorithms on mechanical turk. In *UIST '10: Proceedings of the 23nd annual ACM symposium on User interface software and technology*, pages 57–66, 2010.
11. A. Marcus, M. S. Bernstein, O. Badar, D. R. Karger, R. C. Miller, and S. Madden. Twitinfo: Aggregating and visualizing microblogs for event exploration. In *CHI*, pages 971–980, 2011.
12. A. Marcus, E. Wu, D. R. Karger, S. Madden, and R. C. Miller. Crowdsourced databases: Query processing with people. In *CIDR 2011*.
13. P. Wais, S. Lingamneni, D. Cook, J. Fennell, B. Goldenberg, D. Lubarov, and D. Martin. Towards building a high-quality workforce with mechanical turk. In *Computational Social Science and the Wisdom of Crowds Workshop, NIPS 2010*.
14. E. Wu, Y. Diao, and S. Rizvi. High-performance complex event processing over streams. In *SIGMOD Conference*, pages 407–418, 2006.
15. E. Wu and S. Madden. Partitioning techniques for fine-grained indexing. In *ICDE*, 2011.