CyLog/Crowd4U: A Declarative Platform for Complex Data-centric Crowdsourcing

Atsuyuki Morishima
University of Tsukuba
mori@slis.tsukuba.ac.jp

Norihide Shinagawa
University of Tsukuba
siena@slis.tsukuba.ac.jp

Tomomi Mitsuishi
University of Tsukuba
tomomi@slis.tsukuba.ac.jp

Hideto Aoki
University of Tsukuba
s1221575@u.tsukuba.ac.jp

Shun Fukusumi
University of Tsukuba
s1220693@u.tsukuba.ac.jp

ABSTRACT

This demo presents a principled approach to the problems of data-centric human/machine computations with Crowd4U, a crowdsourcing platform equipped with a suite of tools for rapid development of crowdsourcing applications. Using the demo, we show that declarative database abstraction can be used as a powerful tool to design, implement, and analyze data-centric crowdsourcing applications. The power of Crowd4U comes from CyLog, a database abstraction that handles complex data-centric human/machine computations. CyLog is a Datalog-like language that incorporates a principled feedback system for humans at the language level so that the semantics of the computation not closed in machines can be defined based on the game theory. We believe that the demo clearly shows that database abstraction can be a promising basis for designing complex data-centric applications requiring human/machine computations.

1. INTRODUCTION

In recent years, the significance of integration of data-centric human/machine computations has increased because of the following reasons. First, it has been found that aggregating the power of machines and humans is a promising approach for achieving some of the computationally difficult goals. Secondly, in many of the currently-used data-centric systems, computation is not necessarily closed in machines.

This demo presents a principled approach to the problems of data-centric human/machine computations with Crowd4U, a crowdsourcing platform that is equipped with a suite of tools for the rapid development of crowdsourcing applications. Crowdsourced data processing has been one of the hottest issues in database research today. Proposed systems include CrowdDB [4], Qurk [11], and Deco [12], which focus on different aspects of data-centric human/machine computations, and the common approach is to design small extensions to SQL so that the crowd can participate in the process of SQL queries. In [3], how Datalog-like codes can be partly evaluated by the crowd in the backend is discussed. In contrast, we argue that the declarative database abstraction can be a promising basis for designing a wider range of data-centric crowdsourcing systems. We show that the database abstraction allows us not only (1) to naturally and concisely describe complex data-centric human/machine computations, but also (2) to analyze the behavior of the code executed in the world that is not closed in the machine. This makes Crowd4U different from existing crowdsourced data processing systems and makes the demo unique compared to those of other systems [4] [11]. For example, in one demo scenario, the main contributor of data extraction is gradually changed from the crowd to the machine, and in another scenario, the crowd participates in the process of identifying manageable small tasks to solve a larger problem.

The power of Crowd4U comes from CyLog [9][10], which is a Datalog-like language developed by the FusionCOMP project. It has three extensions to handle human/machine computations. First, it allows predicates to be *open*, which means that the decision on whether a fact holds is crowdsourced when the fact cannot be derived in the database. For example, to solicit the crowd to give keywords to label a given image `i`, we can write the following rule:

```
Label(i, keyword)/open <- Image(i);
```

Here, the value of the attribute `keyword`, which does not appear in the rule body, is to be crowdsourced. Second, it has a built-in reward system at the language level to give a well-defined semantics even if we have open predicates. Finally, it allows us to dynamically add rules into the code. The proposed set of extensions raises novel challenges and opportunities. The highlights of the demo are as follows:

Database abstraction as a language for complex data-centric crowdsourcing. The demo shows that database abstraction can be a promising basis for designing complex data-centric applications requiring human/machine computations. We believe that the decision to adopt Datalog as its basis was successful for several reasons. First, as recent studies suggest [6], languages based on Datalog allow us to concisely describe data-centric applications. Second, logic-based programming has an affinity towards event-driven executions, which complex data-centric applications often require. And finally, describing the code as a set of rules brings the flexibility to the data-centric crowdsourcing. For example, allowing the code to dynamically add and delete rules naturally achieves the *higher-order crowdsourcing*, in that the crowd not only contributes to data input, but also participates in the program evolution.

Proceedings of the VLDB Endowment, Vol. 5, No. 12

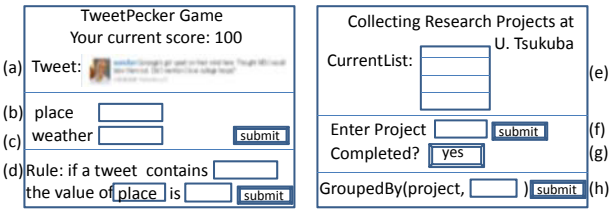

Figure 1: Main Components of UIs for Scenarios

Introducing rational data sources as a means to deal with the world not closed in machines. With open predicates, human factors are incorporated in the program executions. Since people might lie and they need motivation to participate in the computation, it is difficult to predict the execution results. One possible approach is to consider humans as *rational* data sources, who provide data in a way consistent with the expected rewards. The feedback system incorporated in CyLog provides us with components that are designed based on concepts taken from the game theory. Incorporating a feedback system is essential, because then we have a tool to define the semantics of the computation not closed in machines. In CyLog the semantics are defined using the equilibrium of the game (Section 3), which is the state reached by rational workers. This makes CyLog unique since other languages provide no hints on whether users will behave in the expected manner. With the abstraction, tools from the game theory can be used to *predict* behavior. The game theory is known to be useful when discussing not just real "games" but any system that involves incentive structures, such as networks, auctions, and GWAPs [7] [13]. Although the game theory is not a magic wand, this demo will show that this approach is a good starting point.

The remainder of this demo proposal is as follows. Section 2 shows example scenarios requiring complex data-centric human/machine computations. Section 3 briefly sketches the syntax and semantics of CyLog focusing on the three extensions to Datalog. Section 4 explains the architecture of our crowdsourcing platform CyLog/Crowd4U. Section 5 shows the demonstration details.

2. COMPLEX DATA-CENTRIC CROWDSOURCING

Data-centric crowdcsourcing is often more complex than crowdsourced SQL query processing. This section shows two interesting scenarios CyLog makes possible with appropriate incentive design. The point is that the programs, which are executed by both the machines and the crowd, are theoretically guaranteed to work well when the people behave rationally.

Crowdsourced Rule Generation. In the first scenario, we construct tables from a set of tweets. This is a typical scenario that is encountered when constructing "aggregator sites" to collect information on the Web. In our setting, the data extraction process to construct tables is crowdsourced, because we assume that it is not easy to have a single contributor who constantly maintains the tables, nor to immediately develop dedicated software. We design a crowdsourcing application (named TweetPecker) as a GWAP [1].

TweetPecker is similar to the ESP game [1], which crowdsources labeling image files with keywords. The ESP game shows each image to the crowd (players) on line, solicits them to provide appropriate keywords for it, and rewards them when the keywords match. Instead of image files, TweetPecker shows tweets to players, and solicits their help to extract data values from the tweets for attributes to construct tables. Fig. 1 (left) illustrates the main components of TweetPecker's user interface for a player, where a tweet is shown to the player (Fig. 1 (a)). For example, if the tweet shown is "We have good weather in NYC," a worker will input the word "NYC" for the `place` attribute of the tweet using the HTML form (b). A difference from the ESP game is that instead of directly giving a value, a player can give an extraction rule (Fig. 1 (d)) that extracts values as his proxy. An example of the rule is "if the tweet contains `good weather`, the value of attribute `weather` is `sunny`." The value extracted by the rule is adopted when two other players agree that the value is correct. Then, TweetPecker rewards both of the players as well as the player who supplied the rule. When the value extracted by the rule is not adopted, the rule supplier receives a negative score. Interestingly, we can design an incentive structure so that the main contributor of the extraction tasks is gradually changed from the crowd to machines, when the crowds bahave rationally.

Crowdsourced Problem Decomposition. In the second scenario, we solicit the crowd to collect all data values that satisfy a given condition and to decide whether or not the task is completed. Covering all items is important in many applications including Web-based services, such as reviews for restaurants in particular areas. And it is often the case that we need the power of the crowd to collect data items. Fig. 1 (right) shows the simplified interface, with which we collect all of the research projects conducted at a university. The page shows the list of already-collected data values (e) and the HTML form to accept new data (f), which are often found on user interfaces of ordinary Web applications. In addition, the worker clicks on the button (g) to tell that the list is complete. In our scenario, we allow the crowd not only to participate in the process of enumerating items, but also to participate in the process of identifying smaller manageable tasks. For this purpose, we add another HTML form to the interface to allow grouping (h). If a worker specifies that the research projects can be grouped by the laboratory, our problem is decomposed to the smaller tasks of collecting all of the projects in each lab and a task of collecting all of the labs, so that it becomes easier to know what is missing and whether all of the items have been covered. In general, the problem can be further decomposed by recursively grouping the items. From our experience [2], problem decomposition makes the enumeration process more efficient. If we appropriately design the incentive structure, rational workers identify and focus on small tasks instead of continuing to perform the original task.

3. PROGRAMMING IN CYLOG/CROWD4U

CyLog's three important extensions to Datalog are as follows: (1) It allows some predicates to be open, which allows us to concisely write data-centric computations not closed in machines. (2) It has components to define games, in which the behaviors of workers are recorded in a special table and aggregated to compute feedbacks to workers. This gives us a principled framework with which we can define the semantics of the code. (3) It has an explicit table to manage CyLog rules, which allows us to dynamically add and delete rules and makes higher-order crowdsourcing possible.

Crowd4U is a crowdsourcing platform deployed at University of Tsukuba and it provides a set of built-in functions

which can be easily used by the code written in CyLog.

Open Predicates. Below is a fragment of a CyLog program for the TweetPecker game.

```
1. WeatherTweet(t) <- Tweet(t, hashtag:"#weather");
2. WeatherI(t, place)/open <- WeatherTweet(t);
```

Line 1 can be interpreted as a Datalog rule with the closed world assumption, i.e., the `WeatherTweet` table contains those tuples corresponding to tweets with hashtag `#weather`. "x:y" (e.g., `hashtag:"#weather"`) means that y is a value or variable for attribute x. If the variable name is the same as the attribute name, the attribute name can be omitted. In addition, attributes unused in the rule can be omitted.

Line 2 includes an open predicate and states that for each tweet stored in `WeatherTweet`, the value of `place` is to be crowdsourced. The predicate corresponds to the text input field in Fig. 1 (b). Open predicates can have attributes that do not appear in the rule body. Programmers can allow the system to wait for the crowd to answer with HTML forms, or can add rules to explicitly solicit inputs from workers. The rule for the `weather` attribute (c) is similar and is omitted.

With open predicates, the collection of all research projects in the second scenario can be described as follows:

```
1. Project(p)/open;
2. Project:@closed/open;
```

Line 1 is the open predicate that accepts human inputs to collect projects (the input field in Fig. 1 (f)). In Line 2, `Project:` does not denote one tuple but the entire relation, and `@closed` states that the relation is closed, i.e., it already covers all of the projects. Therefore, Line 2 states whether the `Project` relation covers all of the projects is open and should be decided by humans (the button in Fig. 1 (g)).

Path Table and Game Aggregations. CyLog adopts terms and concepts from the game theory to design the appropriate behavior of rational data sources. For example, TweetPecker rewards the workers who gave the same values and adopts the agreed value for an attribute. Fig. 2 shows a part of the payoff matrix for TweetPecker, in which each cell shows not only the payoffs, but the output values. We call such a game with output values a *data game* (only two players and two terms are shown here). In CyLog, a generalized version of the game in Fig. 2 is called `duplicate` game. Then, the semantics of open facts are defined by the *equilibrium* of the game [14], i.e., the state reached by rational players. Note that the game is a typical coordination game [14] in the game theory, in which rational players choose the same value. To compute the value and payoffs, CyLog maintains a special table named *Path* table to record the provenance (corresponding to a *path* in the game theory) to show how players have behaved in the game instance (Fig. 3), and supports for aggregations of the table to compute the value and payoffs. The design of game components gives us a simple and flexible way to incorporate both extensive- and normal-form games [14] in programs.

We extend the code fragment for the TweetPecker game:

```
1. WeatherTweet(t) <- Tweet(t, hashtag:"#weather");
2. TweetPecker(t) <- WeatherTweet(t);
3. game TweetPecker(t) [WeatherI], duplicate {
4.     WeatherI(t, place)/open <- WeatherTweet(t);
5. }
6. Weather(t, place:p) <- TweetPecker(t, value:p);
```

We already explained the rules in Lines 1 and 4. Line 2 starts TweetPecker for each WeatherTweet. Lines 3 to 5

Player A/Player B	Term X	Term Y
Term X	(1,1), Term X	(0,0), NULL
Term Y	(0,0), NULL	(1,1), Term Y

Figure 2: Payoff Matrix for `duplicate` Game

Order	Date	Player	Rel	Action
1	10:10am	Kate	WeatherI	NYC
2	10:11am	Ann	WeatherI	NYC
3	10:12am	Pam	WeatherI	Summit

Figure 3: Path Table

defines the TweetPecker game, whose instance is identified by `TweetPecker(t)`, for which Path table is constructed and game aggregations are executed.

Line 3 states that the changes to the `WeatherI` table is recorded in `Path` table (Fig. 3) and that the game aggregation `duplicate` is executed when the game ends. The programmers can define game aggregations in CyLog by herself [10], or can call built-in aggregations provided by Crowd4U. Line 6 adopts the values computed by `TweetPecker(t)`, in which `value` is an implicit attribute defined for each data game. Payoffs are computed in a similar way and stored in the pre-defined `Payoff` table (omitted).

Dynamic Rule Insertion and Deletion. CyLog rules are managed in a special table, named the `Rule` table. Therefore, the program can extend its own set of rules simply by inserting tuples representing rules into the table. For example, the second scenario solicits the crowd to give possible grouping of research projects so that they can divide and conquer the cover-all problem. A direct way to implement this is to rewrite rules. When the HTML form in Fig.1 (h) accepts "Lab," the original rules to directly solicit the projects can be replaced with the following rules:

```
1. Project(p, lname)/open <- Lab(lname);
2. Project:@closed <- Lab:@closed,
       forall Lab(lname) (Lab(lname, cover_all:true));
```

Line 1 collects the research projects for each lab. Line 2 states that the task is completed when all of the tasks for collecting projects for the labs are completed. We also need to add the rules to collect all of the labs in the same way as we wrote the original set of two rules for collecting projects.

CyLog introduces rule constructors to allow tuples to represent a set of rules. For the cover-all scenario, we can add the new rules by (1) taking a group name from the open predicate corresponding to Fig. 1 (h), (2) constructing the rules using the name (i.e., the new rules for "Labs"), and (3) inserting the tuples representing the rules into `Rule` table.

Codes for the Scenarios. With CyLog/Crowd4U, various applications using complex data-centric human/machine computations can be written in a concise way. We have already explained fragments of the codes for the two scenarios. The code for TweetPecker consists of a few tens of rules and contains the following rules in addition to the explained ones: (1) Accept extraction rules from the crowd (this corresponds to Fig. 1 (d)), (2) extract data values using the extraction rules, (3) when the output of a rule matches with the value given by another worker, both she and the worker who supplied the rule are rewarded, and (4) if a value is adopted as an attribute value of a tweet, workers who gave other values and the workers who gave rules to produce other values for the same tweet all receive negative scores.

A game theoretical analysis [5] of our TweetPecker code proves that rational people give correct rules. Our experiment also revealed that the main contributor to data extraction was gradually changed from workers to the machine.

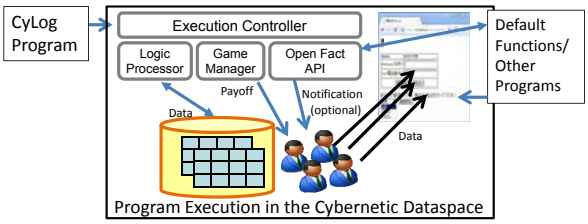

Figure 4: CyLog/Crowd4U Prototype

Similarly, the CyLog code for the cover-all problem consists of a few tens of rules, some of which define the incentive structure for workers to identify and focus on small tasks.

4. CYLOG/CROWD4U ARCHITECTURE

Crowd4U is a data-centric crowdsourcing platform, which is designed to harness the power of people in academia. Currently, it is deployed at University of Tsukuba, and most of the registered workers are students of the university. It is being used for various applications including maintaining academic calendars and covering all of the menu items in the more than 20 restaurants at the university. We explain the architecture of the prototype system (Fig. 4).

Query Processing for Data-centric Complex Crowdsourcing. To execute complex data-centric crowdsourcing applications, Crowd4U supports the execution of CyLog codes by both machines *and* people, interleaved with the executions of programs in general-purpose languages. It provides APIs to invoke events in order to indicate that a new fact holds, which allows us to naturally combine CyLog programs with other codes written in procedural languages.

The CyLog processor adopts a semi-naive event-driven evaluation strategy in which the rules are evaluated in a bottom up way; Crowd4U knows that a fact holds when there is an event indicating that the fact holds. Rules are processed as follows: when all of the facts in a rule body hold, we determine whether the head of the rule holds. When the head is not open, an event for the head fact is always invoked by the logic processor. When the head is open, people are responsible for invoking the event to indicate that the fact holds. Crowd4U provides the default views to interact with people and to call the API to invoke events.

Handling Rational Data Sources. According to the behavior of the users in the program execution, the game manager gives them values that represent payoffs. Currently, the payoff values in Crowd4U are not monetary, but are visible to workers, showing the contribution of each worker in the crowd. However, from our experience, non-monetary values provide sufficient feedback to the workers, informing them of the values of their contributions, and we have found that it is very important to make their contributions visible. Therefore, Crowd4U explicitly acknowledges the contributions of workers in different ways.

Rapid Development Support. Crowd4U provides a suite of built-in functions to support the programming. It has a facility to easily define Web views that are associated to open predicates and event invocations, making it easy to implement interactions with the crowd. Crowd4U also provides functions to implement common game situations including majority votes, duplicate (coordination) games, and other data games with various types of incentive structures.

5. DEMONSTRATION DETAILS

This demo will show the whole workflow on the Crowd4U prototype, from formulating CyLog codes to showing the results. We not only execute the code for the demo scenarios with the workers on Crowd4U, but also encourage the audience to join the scenarios as workers. When we have multiple visitors, we also encourage them to play data games with different incentive structures. The experience helps the audience to get a sense of how essential the incentive structure is for designing data-centric human/machine computations.

We employ mainly the scenarios shown in Section 2. We demonstrate that appropriate incentive structures make the code work well, and that different game designs lead to different results. In the data extraction scenario, we plan to use the hash tag for VLDB2012 (e.g., `#vldb2012`) to collect tweets. We then construct tables on Crowd4U, and demonstrate that the main contributor of the data extraction is gradually changed from the workers to the machine. In the second scenario, we show the power of the crowdsourced program decomposition, by demonstrating that workers identify and focus on small manageable tasks to collect the URIs of research labs related to VLDB2012. We also show that the codes for the scenarios are intuitive and concise. If they want, we explain the proofs to show the codes work well with rational data sources.

Moreover, we will have the audience to try other codes, some of which are given in an ad-hoc style. We try to visually show the inside of the prototype system, allowing the audience to see how the programs are compiled and interpreted, and how the user interfaces are generated.

6. ACKNOWLEDGEMENTS

The authors are grateful to Prof. Sugimoto, Prof. Kitagawa, Prof. Sakaguchi and Prof. Nagamori for their helpful comments. This research was partially supported by PRESTO from the Japan Science and Technology Agency.

7. REFERENCES

[1] L. von Ahn, L. Dabbish: Designing Games with a Purpose. CACM 51(8): 58-67, 2008.
[2] H. Aoki, A. Morishima, N. Shinagawa. Cover-all Operation for Data Crowdsourcing. DEIM Forum 2012, 2012.
[3] A. G. Parameswaran, N. Polyzotis: Answering Queries using Humans, Algorithms and Databases. CIDR 2011: 160-166, 2011.
[4] A. Feng, M. J. Franklin, D. Kossmann, T. Kraska, S. Madden, S. Ramesh, A. Wang, R. Xin: CrowdDB: Query Processing with the VLDB Crowd. PVLDB 4(12): 1387-1390, 2011.
[5] S. Fukusumi, A. Morishima, N. Shinagawa. Data Extraction from Microblogs with a GWAP. DEIM Forum 2012, 2012.
[6] J. M. Hellerstein. The Declarative Imperative: Experiences and Conjectures in Distributed Logic. SIGMOD Record, 39(1), 5-19, 2010.
[7] S. Jain, D. C. Parkes: The Role of Game Theory in Human Computation Systems. KDD Workshop on Human Computation 2009: 58-61, 2009.
[8] M. Keulen, A. Keijzer: Qualitative Effects of Knowledge Rules and User Feedback in Probabilistic Data Integration. VLDB J. 18(5): 1191-1217, 2009.
[9] A. Morishima. A Database Abstraction for Data-centric Social Applications. KJDB2010, 2010.
[10] A. Morishima, N. Shinagawa, S. Mochizuki: The Power of Integrated Abstraction for Data-Centric Human/Machine Computations. VLDS 2011 held at VLDB2011: 5-8, 2011.
[11] A. Marcus, E. Wu, D. Karger, S. Madden, R. C. Miller: Demonstration of Qurk: A Query Processor for Humanoperators. SIGMOD 2011: 1315-1318, 2011.
[12] A. Parameswaran, H. Park, H. Garcia-Molina, N. Polyzotis, J. Widom. Deco: Declarative Crowdsourcing. Technical Report. Stanford, 2011.
[13] Y. Shoham: Computer Science and Game Theory. CACM 51(8): 74-79, 2008.
[14] F. Vega-Redondo. Economics and Theory of Games, Cambridge University Press, 2003.