Programming with People: A Literature Survey

Ryan Compton
UC-Santa Cruz
rcompton@ucsc.edu

ABSTRACT

In this paper, the literature on social computing is discussed. First social computing and its applications will be shown. Then a survey of existing social computing systems are examined. Finally a description of the Dog programming language.

Categories and Subject Descriptors

D.3.3 [**Programming Languages**]

General Terms

Human Factors and Languages.

Keywords

Social/Human Computing, Crowdsourcing, Literature Survey.

1. INTRODUCTION

Human or Social Computing uses humans natural ability to solve ordinary tasks that are too difficult for computers to solve. This method of computing has become popular in applications toward complex problems. The problem is split into micro-tasks and dispatched to people and finding a common answer among the crowd.

Human computation has been used in a wide range of goals; one task is classifying galaxies from images taken by the Hubble telescope. The Galaxy Zoo project has been using crowdsourcing to assist in the classification needed of large numbers of galaxies [4]. The project has been so successful it has moved on to doing even more complex problems to be solved with its crowdsource approach. Other possibilities that can arise from using people as your source of data and data extraction are story generation. *Amazing but True Cat Stories* is a novel created from the collective stories of people [3]

Another working task takes advantage of the large amounts of effort that people use to solve computer games. Foldit is a multiplayer online game that encourages gamers to solve hard predication problems with 3-D protein structures [2]. Locating the formation of a protein is a heavy computational challenge due to the vast size of search space. Human computation was successfully applied to the problem and was more successful with structure prediction than typical machine computation.

This leads into the first issue of human computation, motivation. Foldit uses the desire for people to play video games; this motivational tactic is the foundation for "Games with a purpose" [1]. These games to be applied to a vast amount of problems across disciplines to have human computation as a means to solve them. Gaming was shown to be a strong tool of recruitment for crowdsourcing, however there still remains issues with the current system of human computation.

Typical systems that are created to use human computation have a very simplistic approach; build a system that can answer a difficult question. However once these systems are built they can only be applied to that single system, there is no generalization that allows for a cross over.

In this paper, some current systems of human computation and crowdsourcing are presented, with a follow up about systems that address this issue of generalization. The main section of this paper will introduce the Jabberwocky framework which uses the Dog Programming Language. The motivation behind this paper is to encourage further work on human computation system in order to enhance its applications towards more complex problems as well as discuss the Dog Programming Language.

2. CURRENT SYSTEMS OF SOCIAL COMPUTING

There have been a various amount of approaches toward social computing. Some systems are simply an interface into other currently existing crowdsourcing software. This interface allows an extension of other methods of creating crowdsourcing tasks than the main system itself. Other systems have used the approach of treating humans as a database, thus using the principles associated with declarative query languages.

2.1 Programming frameworks for Human Computation

2.1.1 Crowdforge

Crowdforge is a framework and toolkit for crowdsourcing complex work [10]. It takes a MapReduce approach to crowdsourcing, since typical crowdsourcing tasks are of simple tasks and parallel in nature [6]. People are recruited as "Mappers" and "Reducers" which correspond to the responsibilities as in the MapReduce algorithm. Crowdforge attempts to remove the designer from the system as much as possible. Each person in the crowd can act as a "Mapper" in which they can break a problem into sub-problems for other people to solve. Once these sub-problems are solved, then people recruited as a "Reducer" will find the best response to the problem.

This framework has specific goals in order to transition crowdsourcing into MapReduce. One key feature is dynamic partitioning. It is used so that workers can act as the Mappers and decide for themselves how a task can be partitioned. Their results thus will create new subtasks for other workers. This is opposed to the typical system of the task designer needing to specify partitions beforehand which allows for the crowd to decide how a problem can be broken down. There also exist multi-level partitions, so a task can be broken up by more than one partition. Due to this multi-level petitioning, complex flows are a focus of Crowdforge.

Quality control becomes an issue with a system reliant on people who have little experience with a specific task. To address this Crowdforge uses voting, verification, or merging items to find the best response from the crowd, these are the Reducers. One subtask can be for people to rate the response from another person, therefore giving that response a vote. Responses with the highest vote will be used as a result. Other subtasks can be to verify if one response is a valid response to the task, also the act of merging more than one response together.

In Summary, Crowdforge breaks the structure of distributed computing into three types of subtasks. One, the partition tasks, which is where the Mappers break down the larger tasks into discrete subtasks. Secondly, the Mappers specify which task is to be processed by one or more workers. And third, the Reducers reduce the results of multiple workers' tasks into a merged single output.

Currently there are three limitations to Crowdforge. The system does not support iteration or recursive tasks within the task flow. This system is also assuming that the complex tasks given can be broken up into smaller sub-problems. Lastly it is assumed that each sub-task is assumed to be independent. These limitations are very specific to the type of problem Crowdforge is tasked to, but they are issues that occur in many common parallel practices that must be addressed.

*2.1.2 **Turkit***

Turkit is a toolkit used for prototyping and exploring algorithmic human computation, while keeping a simple imperative programming style [11]. Turkit is an extension of JavaScript that introduces functions for interacting with Amazon's Mechanical Turks (MTurk), which is a flexible platform that supports various kinds of human computation. Within MTurk, requesters post small intelligence tasks called HITS for workers to conduct. These workers get paid a small amount of compensation for completing the HIT of their choice. MTurk currently remains exclusive to the tasks that the requester posts.

Turkit's model allows for the exploration of iterative workflows as well as multi-phase task decomposition, features that would be difficult for a requester to conduct simply using MTurk. Other features included address the issue of reliability with conducting complex tasks. MTurk tasks take time to complete, which makes programming workflows more difficulty. Turkit uses a Crash and Rerun Programming style that allows for a script to be re-executed without re-running costly side-effecting functions. A program in Turkit has specific calls to when a command should be rerun or not. Command and data history is stored throughout run time in order to reduce the necessity for a program to be rerun if it crashes. The flow of this system is as such: a script is executed until it crashes, thus a script will be intended to crash once it is finished. Each successfully executed line will be stored in a database. Once a crash occurs that was not intended, the program will automatically rerun from the start. A line can be marked as a non-rerun line by the programmer, thus skipping lines that have a cost to rerunning. In this case the result of this line will be looked up in the database.

There are some issues with determinism using the crash and rerun programming model, but it does address the issue of high cost with running programs toward using people as the primary means of computation, since people cost money.

*2.1.3 **Soylent***

Soylent is a word processing interface between MTurk and Microsoft Word[12]. This interface enables writers to call on MTurk workers to shorten, proofread, and conduct further edits on parts of their document on demand. Soylent's model to improve worker quality was to use a Find-Fix-Verify crowd programming pattern. This will split tasks into a series of stages, generations and review.

Soylent is comprised of three main components: 1) Shortn, a shortening service that will reduce the size of a selected text to 85% of the original length. It is able to accomplish this task without changing the intended meaning of the text and without creating any new errors. 2) Crowdproof, which uses the crowd to find spelling and grammatical errors and provide fixes to these errors. 3) Human Macro is an interface that offloads the tasks of arbitrary word processing. One example of this is formatting citations or finding appropriate figures.

The key importance to Soylent is its interactive user interface. This embeds paid crowd workers to the interface to allow for them to be available to solve complex cognitive and manipulation tasks on demand.

Soylent also addresses another common issue with crowdsourcing behavior. Roughly 30% of the results to such an open-ended task are considered to be a poor result. This is where the Find-Fix-Verify structure comes into play. Find is where the Turkers identify which part of a user's work needs more attention. Soylent will move the parts where at least 20% of the workers agree that it needs work. Fix is where the work is done of the identified patch. Here Turkers produce a fix for the patch. Lastly Verify addresses the issue of poor results. The Turkers are asked to vote on which fix was best for the patch. The highest voted will be presented to the user as a fix to their error.

Overall performance of Soylent was impressive considering some of the potential issues. Latency is higher for using Soylent than an automatic grammar and spell checker, however Soylent would complete a fix in about 2 minutes. When tested against a grammar checker, Soylent was about to catch 67% of errors it was presented, while the grammar checker was only able to catch 30% of those errors. However with both systems combined, they were able to catch 82%, which supports a combined system to address errors.

2.2 Declarative query languages for Human Computation

*2.2.1 **hQuery***

hQuery treats human computation as a database[7]. Through a declarative command, the programmer is able to specify what must be accomplished through the crowd. The system will transparently optimize and manage the evaluation details.

Issues of latency and quality also make this approach sufficiently complex just as the previously mentioned approaches to crowdsourcing. Specifically here, this complexity requires a redesign to the query processor, providing a new ground for innovation in data management research.

hQuery is also susceptible to the challenges of uncertainty in answers and monetary costs that come from using a human computational technique. This motivates for further work into the hQuery system.

2.2.2 CrowdDB

CrowdDB is another approach similar to that of hQuery, however is an extension of SQL[8]. CrowdDB simply uses human input provided by crowdsourcing to process a query that neither a database nor search engine can satisfy. SQL is used as the language for posting complex queries and further is used as a way to model the data return from the queries.

CrowdDB had a major change in the traditional closed-world assumption for query processing not holding for human input. Implementation perspectives gives need to solicit, integrate, and cleanse crowd sourced data, which all are address

Just as previous systems described above, CrowdDB also takes advantage of the existing crowdsource tool MTurk. CrowdDB thus acts as a middle man between the user and crowdsourcing, using a declarative approach.

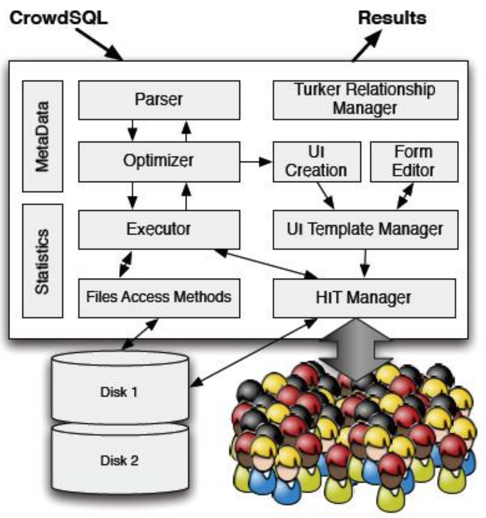

Figure 1: CrowdDB Architecture

Figure 1 shows the layout of CrowdDB. A request is issues from an application through the use of the extension of SQL, CrowdSQL. An application can be built in a normal way and the handlers for the crowd are kept within CrowdDB. As seen in the left side of the figure, the traditional query compilation, optimization, and executors are present. These are extended to incorporate human generated input provided by the MTurk system.

2.2.3 Qurk

Qurk is another system that uses MTurk and a declarative query approach to human computation [9]. Qurk differs in that it uses an asynchronous query executor, an MTurk-aware optimizer that can consider monetary costs and result accuracy, and lastly an adaptive approach to query process to address the not predictability of operator selection.

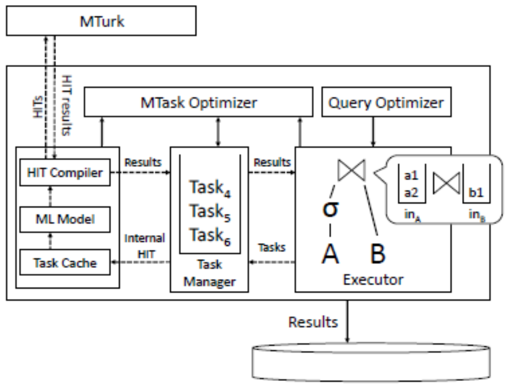

Figure 2: System diagram of Qurk.

As seen in this diagram, Qurk is very similar to the CrowdDB system in the sense that common query executors are used and further extensions are made to incorporate MTurk as an output and input.

3. Programming Environments with Focus on Modularity and Reuse

3.1 HP

Human Processing (HP) is an environment that builds upon previous environments targeted toward human computation, in which it accomplishes this through abstraction [13]. The HP environment works as follows:

1. A Programmer writes a typical program. The programmer then uses implementations of human drivers, tasks, marketplace drivers, or recruiters.
    a. A Human driver is a program that manages the interactions with humans, similar to how a device drivers manages a physical device within an operating system.
    b. A Marketplace driver provides the direct interaction with a marketplace such as MTurk.
2. The programmer will reuse or define certain structures which are referred to as human task descriptions. The task consists of an input, output, human driver, web form, and other meta data that may be necessary. A human task description can be instantiated will one or more instances of a human task.
3. If the programmer wishes to avoid using any type of marketplace, they can instead program a recruiter to serve as the interface to one or more marketplace driver.

The point of the HP environment is to maximize the amount of code reuse. Therefore, limiting the amount of programming needed for a new problem. That is why the environment includes a library of algorithms to be implemented by humans to encourage reuse between different problems.

4. Jabberwocky Programming Environment

To further enhance applications of human computation to more complex problems, systems need to involve the real identities, social structure, and expertise modeling that exists within crowdsourcing. This has been done through the use of question-answering with Aardvark [5]. To address this issue along with the issue of stand-alone systems with rigid structure and requirements in existing crowdsourcing solutions, Jabberwocky was created.

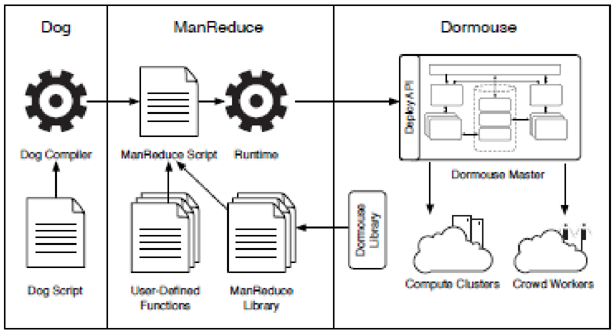

Figure 3: Overview of Jabberwocky

Jabberwocky is a social computing stack that takes advantage of both human and machine computation [14]. It has three components: Dormouse, ManReduce, and Dog. Dormouse is the "virtual machine" for Jabberwocky. It is at the lowest level of software libraries that interact with both people and traditional computing machines. Dormouse is what maintains the real identities, user profiles, and social structure that are within the cowdsourcing solutions. Finally programmers can naturally interact with both control flows of human and machine computation through the communication protocols within Dormouse.

ManReduce is an extension of MapReduce, a parallel programming framework that not only uses machine computation (MapReduce), but extends this framework to human computation as well. This is achieved through crowdsourcing's natural parallel nature. ManReduce sits on top of Dormouse in the Jabberwocky stack.

Dog is a high level scripting language on top of ManReduce. Dog was designed with three goals in mind. First make Dog a highly expressive language, in that a person with little knowledge of programming could understand and write a Dog program. This is a good design goal since the fundamentals of human computation is speaking to people. The second goal is reusability, similar to previous system's goal, to take advantage of previously used programs using human computation and apply them to new ones. And third, to keep the power and flexibility of ManReduce, while obtaining these two previous goals.

The first two goals were achieved by defining a number of library functions that conduct common human and machine functions. Such human functions are Vote, Label, Compare, Extract, and Answer. Such machine functions are Histogram, Filter, Median, and Sort. These functions are specific to which form of computation would perform best at it.

The last goal was achieved by allowing programmers to write their own libraries of functions for humans and machines inside of ManReduce.

4.1 Dog Language Specifications

Dog uses a compiler that has a recursive descent parser, in which parses Dog programs and then creates ManReduce code. Dog functions are wrappers around the existing functions inside ManReduce, which contains mappers and reducers.

Dog has four high-level primitives, PEOPLE, ASK, FIND, COMPUTE.

PEOPLE - a command that returns a specification of people in which can be used inside the ASK and FIND commands. This command requires a FROM clause that gives the Dormouse community or other crowdsourcing service where the people will be selected, such as the command below.

```
workers = PEOPLE FROM facebook
```

This command specifies that the predicate "workers" are people from the community "facebook". Furthermore people can be specified to have certain features. This can be conducted with a WHERE clause.

```
workers = PEOPLE FROM gates WHERE expertise
    CONTAINS 'theory' AND advisor='don knuth'
```

Here, the workers are now people from the community "gates" where their expertise contains "theory" and their advisor is "don knuth". This also gives a good example how easy it is to translate a line of Dog code into an English sentence.

ASK – a command that executes a human function that takes in a specification of people as an argument.

```
labels = ASK workers TO Label ON data
```

The command above is asking the workers to perform an action of labeling on the data. Each human function has some default parameters which can be changed if the programmer wishes to. This is done with adding the USING clause.

```
labels = ASK workers TO Label ON image_data USING
    layout='game'
```

The workers were now asked to label on the data using a specified layout 'game'. Other human functions that can be used are: Vote, Label, Compare, and Answer, as well as programmers own created functions.

FIND – a command that will instantiate a specification of people outside of the ASK function. If a programmer wants to examine features of a community, they can begin by instantiating the community using this command and use additional clauses.

```
workers = PEOPLE FROM gates WHERE expertise
    CONTAINS 'machine learning'
ids = FIND workers
```

This Dog program will instantiate those workers that are specified in the first line. Furthermore FIND can be used to return people who have successfully performed a task.

COMPUTE – a command that executes a machine function. This command will take in a machine function for an argument as well as a set of data to perform the function on. This is conducted in the same manner as with the ASK function however with a different set of library functions.

```
tag_cloud = COMPUTE TagCloud ON words USING
    color_scheme = 'random'
```

4.2 Data Model

Dog is intended to be used on large-scale data and support sequential transformations on such data. Human or machine functions in parallel will conduct the transformations. Dog has two data types: people specifications and data maps. A people specification, created by the PEOPLE command and stored within the predicate to a line using this command. A data map is essentially a key-value store typically used in parallel frameworks. Key-value stores work well inside sequential and parallel frameworks and thus become almost natural to use inside a crowdsourcing framework, due to the parallel work of many people and the sequential work of an individual person. Lastly this model supports the tracking of who is producing which segment of data.

4.3 Routing Tasks

A SUCH THAT clause is used to enable routing tasks based on expertise, demographic, or social structure.

```
validated_reviews = ASK advisors TO Validate ON
    reviews SUCH THAT review.reviewer = advisor.
```

In the example above, advisors will only validate on reviews where the reviewer was the advisor.

4.4 Other Features

Dog as function libraries as a default, this gives a start to a Dog programmer instead of having to create them their self. If they wish to create their own library, they can do so using a directory with the .doghouse extension. Within that directory there must be the appropriate Dormouse and ManReduce files which define the new human and machine functions. Importing these libraries only takes one line.

```
REQUIRE "/dog/lib/statistics.doghouse" AS stats
```

Importing existing data sets is also possible, using the same syntax as REQUIRE however replacing the command to be IMPORT.

There are a number of commands currently implemented in Dog. MERGE, SHIFT, UNSHIFT, and CROSS are all commands that involve the act of joining or manipulating communities or data sets. There are also PARAMETERS clause that can be used inside of ASK to encapsulate the parameters returned from ASK.

Dog also has the feature of composable primitives. This allows more compact and sometimes readable code.

5. DISCUSSION

Interfacing with current crowdsourcing systems is a good way to take advantage of what has been done before. There is no need to recreate an already working system that has been shown to produce quality results. These interfaces are good steps forward to implementing social computing toward higher complexity problem sets.

Declarative query approaches are also good since they are using a paradigm that has already been in use for some time. It was only natural to treat a new data source as such.

Jabberwocky provides an interesting approach to not only crowdsourcing generalization, modularity, and reuse, but also provides a more human approach to writing a crowdsourcing solution. This allows for further use of such tools outside of the programming community and could even serve as a good introductory tool for programming.

Dog's programming style provides a step in a more human readable direction that has been lacking from a field that has been focus on human solutions. Programming languages such as Dog were the next step in taking advantage of social computing systems. Even when the problem of being able to generalize common social computing themes, being able to generalize a line of code into a language outside of the niche for a programmer expert provides a good alternative to a niche of expertise plagued skill sets.

6. REFERENCES

[1] L. von Ahn. *Games with a Purpose*. Computer, June 2006.

[2] S Cooper, F. Khatib, A. Treuille, J. Barbero, J. Lee, M. Beenen, A. Leaver-Fay, D. Baker, and Z. Popovic. *Predicting protein structures with a multiplayer online game.* Nature, June 2010.

[3] B. Hartmann. *Amazing but True Cat Stories*. http://bjoern.org/projects/catbook/, April 2009.

[4] S Bamford and et al. Galaxy Zoo

[5] D. Horowitz and S.D. Kamvar. The anatomy of a large-scale social search engine. In Proc. WWW (2010).

[6] J. Dean and S. Ghemawat. *MapReduce: simplified data processing on large clusters*. Communications ACM, January 2008.

[7] A. Parameswaran and N Polyzotis. *Answering Queries using Humans, Algorithms and Databases.* Technical report.

[8] Michael J. Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. *CrowdDB: answering queries with crowdsourcing.* In Proc. SIGMOD (2011), pages 61–72.

[9] A. Marcus, E. Wu, S. Madden, and R. C. Miller. *Crowdsourced Databases: Query Processing with People*. In Proc. CIDR (2011).

[10] A. Kittur, B. Smus, and R. E. Kraut. *CrowdForge: Crowdsourcing Complex Work*.

[11] G. Little, L. Chilton, M. Goldman, and R. Miller. *TurKit: Human computation Algorithms on MTurk*.

[12] M. Bernstein, G. Little, R.. Miller, B. Hartmann, M. Ackerman, D. Karger, D. Crowell, and K. Panovich. *Soylent: a word processor with a crowd inside.* In Proc. UIST (2010).

[13] P. Heymann and H. Garcia-Molina. Human processing. Technical report.

[14] S. Ahmad, A. Battle, Z. Malkani, and S. Kamvar. *The Jabberwocky Programming Environment for Structured Social Computing.* Proceedings of the Twenty-Fourth Symposium on User Interface Software and Technology, October, 2011.