Crowd Computing

Robert C. Miller
MIT CSAIL
rcm@mit.edu

ABSTRACT
Crowd computing harnesses the power of people out in the Web to do tasks that are hard for individual users or computers to do alone. Like cloud computing, crowd computing offers elastic, on-demand human resources that can drive new applications and new ways of thinking about technology. This paper describes several prototype crowd-computing systems we have built, including Soylent, a Word plugin that crowdsources text editing tasks; VizWiz, an app that helps blind people see using a crowd's eyes; Adrenaline, a camera shutter driven by crowd perception; and Caesar, a system for code reviewing by a crowd of programmers. Crowd computing raises new challenges at the intersection of computer systems and human-computer interaction, including improving quality of work, minimizing latency, and providing the right incentives to the crowd. We discuss the design space and the techniques we have developed to address some of these problems.

INTRODUCTION
Crowd computing is the coordination of a large group of people connected over the Web, each making a small contribution toward the solution of a problem that can't currently be solved by software or one user alone. Other names exist for this approach. One such name, *crowdsourcing* (Howe 2006), coined by analogy to outsourcing, emphasizes how the approach challenges traditional employment models by recruiting people for very short-term work, usually with incentives other than money. Another name, *human computation* (Law & von Ahn 2011), emphasizes the use of people as components of a computing system, computing functions that require human perception, cognition, or knowledge that is currently beyond the reach of artificial intelligence. In contrast, the *crowd computing* name is instead inspired by *cloud computing* (Hamdaqa & Tahvildari 2012), which refers to a system in which highly-available, elastic computational and storage resources are available over a network. Crowd computing systems obtain the same benefits with *human* resources: a networked crowd that can grow and shrink and is available when needed.

Recent years have seen a number of successful crowd computing systems, with a variety of incentives drawing the crowd together, including volunteering, fun, social, and pay. Prominent examples of volunteer-driven systems include Wikipedia[1] and citizen science projects like Galaxy Zoo.[2] For crowds looking for fun, *games with a purpose* (von Ahn & Dabbish 2008) manage to do useful work as a side-effect of playing the game, like tagging images with keywords in the ESP Game (von Ahn & Dabbish 2004) or discovering how to fold proteins in FoldIt (Cooper et al. 2010). Crowds connected on social networks like Facebook and Twitter can also produce useful work, even from small contributions from each person. For example, the tagging of people's faces in Facebook photos enables Facebook to train and improve automatic face recognition.

Crowds can also be paid to do work, as on sites like Amazon Mechanical Turk[3], or MTurk. Named after an 18th century chess-playing automaton, which was actually a hoax because a human chess master was hidden inside it making the actual moves, MTurk is a web service in which people, not computers, do the actual work. On MTurk, a requester can post a job, called a Human Intelligence Task (HIT), typically paying 5-50 cents and taking seconds or minutes to do. A worker accepts the task, completes it in the web browser, and submits their work to the requester for payment.

MTurk is an increasingly important resource for social science experimentation (Mason & Suri 2012), and its demographic characteristics (Iperoitis 2010) and ethical implications (Silberman et al. 2010; Bederson & Quinn 2011) have been investigated. Other online labor marketplaces have also arisen, including CrowdFlower[4], MobileWorks[5], and oDesk[6]. But MTurk continues to be unique in that *all* of the requester's interaction can be automated with an application programming interface (API). MTurk can therefore be integrated into a system that otherwise consists entirely of software. MTurk is thus the first example of a paid crowd computing *utility* – a resource of human intelligence that is highly-available, elastic, and programmable, and that can be a building block in a system. Even for systems that will eventually be powered by other crowd motivators – volunteerism, fun, or social interaction – MTurk provides a way to prototype and develop the system itself, without having to simultaneously undertake

[1] http://wikipedia.org

[2] http://www.galaxyzoo.org

[3] http://www.mturk.com

[4] http://crowdflower.com

[5] http://www.mobileworks.com

[6] http://www.odesk.com

the substantial challenge of building and managing an online community (Kraut & Resnick 2012).

This paper describes some recent explorations into crowd computing systems, mostly but not all prototyped on top of MTurk. The systems explore new approaches to three key metrics of crowd work: the *quality* of the work; the *latency*, or time required to get work back; and the *incentives* involved in motivating people to contribute. The systems range over a variety of application domains, including handwriting transcription, document editing, assistive technology for the blind, and help for students learning programming, illustrating that crowd computing is a broadly-applicable approach in many domains. We conclude with a discussion of the design space for crowd computing, looking toward a future in which crowds are a common component of the toolkit used by software system designers.

IMPROVING QUALITY

Work obtained from human crowds can be unreliable. People can misunderstand instructions and make mistakes, and some people provide maliciously wrong. Removing this noise from a crowd computing system is generally done in one of four ways:

(a) *redundancy*, where multiple people are asked to do the same task, and their answers are aggregated automatically, using averaging, majority vote, or machine learning (Sheng, Provost, & Ipeirotis 2008);

(b) *rating*, where one person does the work and a different group of people is asked to rate or vote on the quality of the answer;

(c) *gold-standard tasks*, where the system includes tasks with already-known answers among the work a person is asked to do, and uses the person's performance on those tasks to decide whether to accept or reject their other work (Oleson et al. 2011);

(d) *behavioral measures*, where the system observes secondary metrics about how the work was done, such as how much time was spent working or the amount of scrolling in the web browser, to distinguish helpful workers from unhelpful ones (Rzeszotarski & Kittur 2011).

Our own work has studied ways to incorporate quality-control measures like these into a workflow for a complex cognitive task. For example, the Improve-and-Vote workflow (Little et al. 2010) is an iterative process in which one person tries to improve an artifact and another group of people votes on whether the change actually was an improvement. Figure 1 shows how Improve-and-Vote can be used to transcribe a messy handwriting sample collaboratively. We have also applied the idea to brainstorming and image captioning (Little et al. 2010). Although Improve-and-Vote can be very effective for producing a quality result, it can be slow and costly to run. A decision-theoretic control approach has been proposed to optimize it (Dai, Mausam & Weld 2010), which maintains belief estimates of artifact quality and worker ability and uses those estimates to automatically decide when the system should ask for a vote, ask for an improvement, or stop the process. One cost-saving rule of thumb from these decision-control experiments is that the first few iterations can usually be run with no voting at all, since successful improvement is very likely in those iterations.

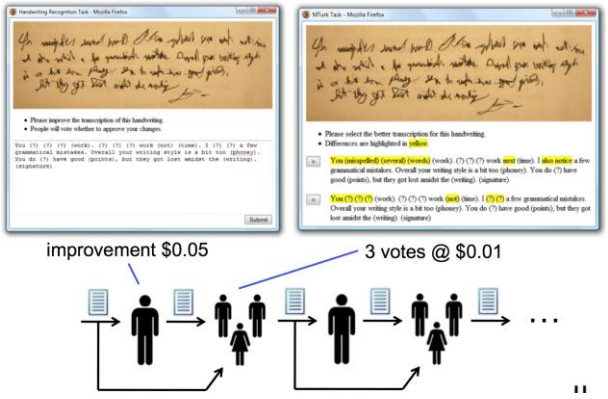

Figure 1: Improve-and-Vote workflow applied to handwriting transcription.

We have also developed a workflow for document editing, called *Find-Fix-Verify* (Bernstein et al. 2010). The workflow is illustrated in Figure 2. In the Find step, redundant workers identify problems in the text, and only problems identified by at least two workers are kept. In the Fix step, other workers propose edits to fix the problems, and the Verify step uses rating to keep only the best edits. We incorporated this workflow into a Microsoft Word plugin, called Soylent, to implement two features, proofreading and text-shortening. Soylent demonstrates that crowd work can be integrated as a component of an interactive system, and used in a way that feels like a feature of the user interface.

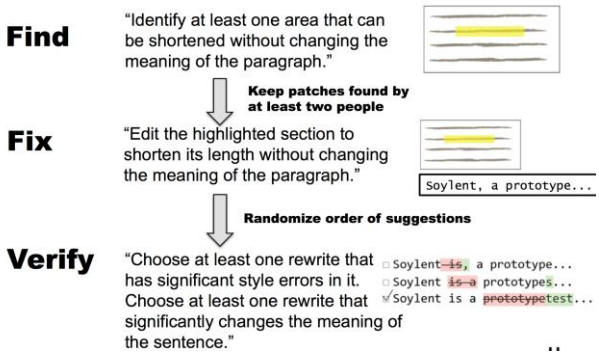

Figure 2: Find-Fix-Verify workflow used to trim unnecessary text from a document.

IMPROVING LATENCY

Our recent work has focused on *realtime crowdsourcing*, in which the crowd's help is needed within seconds in order to support an interactive application. Examples of interactive applications we have built that depend on realtime crowds include VizWiz (Bigham et al. 2010), a smartphone app that allows a blind user to take a picture, ask a question about it, and get answers from a crowd in less than a minute (Figure 3); and Adrenaline (Bernstein et al. 2011), a camera app with a "crowd-controlled shutter", which captures a

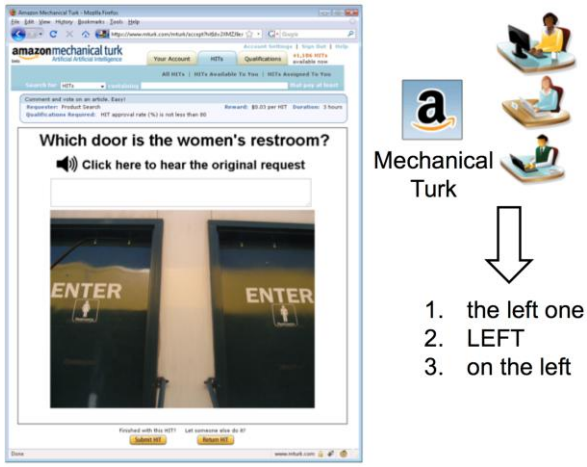

Figure 3: The VizWiz system allows a blind person with a smartphone to take a photo and speak a question about it, and get the question answered by members of a crowd.

short video and sends it to a crowd to choose the best frame to keep, getting the answer back in seconds.

Over the past year, our work in realtime crowdsourcing has produced several findings. First, we have developed a new technique, the *retainer model*, that has crowd members ready to help on demand by recruiting them in advance and paying them a retainer to wait for a short time. Our experiments with the retainer model on Mechanical Turk found that when workers were put on retainer and then recalled in five minutes, 50% responded within two seconds, and 75% within three seconds, making this approach feasible for an interactive application (Figure 5). Using these empirical results, we developed a theoretical model of retainer pools that would allow a system designer to predict the size of the pool required to obtain a desired low response time and low probability of failing to have

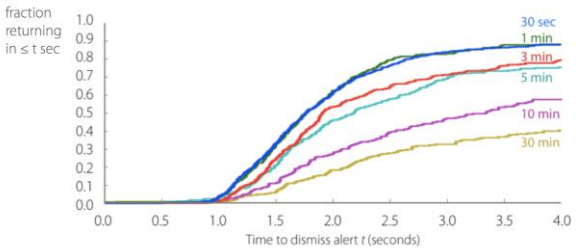

Figure 4: The retainer model recruits crowd workers before they are needed. Experiments on Mechanical Turk show that with a short retainer interval of 5 minutes, roughly half the retained workers are ready to work within 2 seconds after being called on.

enough workers ready. This model also enabled us to devise and test a new technique, *precruitment*, which recalls the workers a few seconds before the request even arrives, in order to mask the 2-second latency of recall (Bernstein et al. 2012).

A second finding is an improvement in quality control for realtime crowdsourcing. Normally, quality control (e.g. voting or looking for agreement between crowd members) adds extra latency to the crowd's work, which slows down the response. To counter this, we developed the rapid refinement algorithm, which guides a crowd to agree on a point in a continuous parameter space (e.g. the best time point in a video) and to do so very quickly. In the Adrenaline camera app, we found that rapid refinement took 3-5 workers a median time of 11 seconds to agree on the best frame of a video, which was several seconds faster than just taking the first answer (no quality control at all), and more than three times faster than voting. We are currently exploring how to apply the ideas of rapid refinement to other domains, such as question answering, web searching, and audio selection.

CROWDS WITH OTHER INCENTIVES

A final consideration in crowd computing system design is the set of incentives that power the system. In the previous systems – Soylent, VizWiz, Adrenaline – the crowd is drawn from Mechanical Turk and motivated by pay. In reality, however, crowd computing is liable to be deployed on a crowd driven by a variety of incentives, including altruism, social reciprocity, and entertainment.

Some of these issues have come up in a crowd-computing system we have developed for classroom use at MIT. *Caesar* is a code reviewing system that allows a mixed crowd of students, alumni, and teaching staff to collaborate on reviewing student programming assignments. Each of these groups has different incentives. Students are motivated extrinsically by grades and intrinsically by a desire to learn. Teaching staff are motivated extrinsically by pay and intrinsically by a desire to impart knowledge. Alumni are motivated intrinsically by altruism, and extrinsically (in our experience) by a desire to meet students and recruit them for summer internships and full-time jobs. When these varying incentives come together in a system, it increases the complexity of the design problem.

The Caesar system has been deployed in several semesters of an MIT software engineering course, 6.005 Software Construction. So far, Caesar has been used to review 13 problem sets, comprising roughly 2500 student submissions by roughly 390 undergraduate students. Counting those students, plus alums and teaching staff, over 500 people have done reviewing, and together they have made more than 21,000 comments on student work.

With the new system, students received written comments about their programs within 3 days after submitting them, considerably faster than the several weeks it often takes for

graders. This fast turnaround time enabled us to institute a "returnin" policy that allows students to revise and resubmit their programs in response to the comments, in order to improve their grade. The average program was reviewed by 10 different reviewers and received 9.6 comments.

An analysis of a sample of comments showed evidence that the crowd interaction promoted learning, particularly from weaker students reviewing stronger students' solutions. For example: "Student: This is interesting. Why do you store all the messages you send/receive in a log? Code author: For debugging. The log adds time stamps, which help a lot for debugging concurrency problems."

We found that most comments in the sample were useful critiques (bugs, clarity, performance, simplicity, or style); some were evidence of learning on the part of the reviewer (as mentioned above) or positive reinforcement of something specific that was good about the code. A fraction of comments (roughly 15%) were some form of "looks good to me," which may have been true, but meant that neither the reviewer nor the code author learned anything from that interaction. Turning this around, however, the remaining 85% of the comments in the system did indicate some degree of learning opportunity had been created by Caesar that didn't exist before, which we take as highly positive, with room for improvement.

One lesson learned is the importance of choosing the right code to review. Programming assignments typically have a lot of uninteresting code – staff-provided code, test cases, tiny exception classes – and an automatic code reviewing system must be smart about what is worth assigning to code reviewers. Caesar had a few early bugs in its reviewing assignment algorithm that led to boring reviewing for the reviewers and unhelpful feedback for the code authors. If the crowd had consisted of paid workers, this may have been less of a problem, but for a crowd with varying incentives, the value of the work can suffer when those incentives are not met.

THE CROWD COMPUTING DESIGN SPACE

Software system designers already have a variety of tools in their design toolbox, including programming languages, platforms, frameworks, libraries, and design patterns. These software tools, in combination with human end-users operating and interacting with the resulting system, have had enormous impact over the last 50 years of the computing era, affecting virtually every sphere of human activity (NRC 1995, NRC 2012). Crowd computing introduces a new kind of component to this toolbox: a crowd of people making small contributions at the system's behest, and coordinated by automatic algorithms. The human intelligence embodied in a crowd has the potential to change how we build and deploy software systems in significant ways.

One such change is the notion of *deployable Wizard-of-Oz prototyping*. Wizard-of-Oz prototyping is a tried-and-true technique for experimenting with ideas in artificial intelligence or human-computer interaction that are currently hard or impossible to build. Essentially, a human simulates the system, doing manually what software will eventually do automatically, acting like the "man behind the curtain" (hence the term Wizard of Oz). Wizard-of-Oz prototyping was used in early experiments with speech recognition (Gould, Conti & Hovanyecz 1982), and has been widely used in user interface design with techniques like paper prototyping (Snyder 2003). In the past, Wizard-of-Oz prototyping was limited to laboratory use, since the wizard had to be present in order to simulate. With crowd computing, however, the *crowd* can take the role of the wizard, using their human intelligence for problems that we don't know how to solve with software yet. Since the crowd is networked and highly available, a Wizard-of-Oz system that uses a crowd can escape the laboratory, and be deployed for real-world use by real users. The VizWiz system is one example. Thousands of blind users have installed it on their phones, and over 40,000 questions have been asked, shedding important light on the kinds of information needs that blind people have and how they ask their questions (Brady et al. 2013). Recent web startups have also used this technique of bootstrapping a system with crowd work (Yoskovitz 2011). Using a crowd-driven Wizard of Oz prototype has two benefits. First, it allows the system to be deployed much sooner, in order to learn whether users actually *need* it, and help it evolve faster to meet user needs. Second, it allows the system to start building a corpus of data – such as photos and questions asked by blind users, along with answers given by crowd workers – which are essential for training machine learning algorithms. After enough data has been collected, artificial intelligence can be introduced into the system to handle tasks that computers can do. This shifts work away from the crowd, reducing the cost of crowd labor, while still keeping the crowd available for the hard tasks that we don't know how to do with AI.

In our experience of developing crowd-powered systems, we have made many mistakes and learned a few lessons. A design handbook for crowd computing has yet to be written, but a few principles are known. First, it helps to divide work into chunks that are as small as possible (for parallelism and fault-tolerance), but not so small that the worker loses necessary context. Second, a system designer should expect noise (poor quality work), even from the highest-quality crowd, and design for it, for example using workflows like Find-Fix-Verify that incorporate quality control mechanisms. Third, a designer must keep in mind that crowds are powered by a variety of incentives, and make sure that the system aligns with and supports those incentives, or the crowd may drain away.

Finally, when deciding whether a particular system would benefit from the crowd component in the toolbox, it's important to think about what the crowd brings to the system. Most systems already have human users

interacting with them or operating them, so human intelligence is *already* part of the system, broadly construed. So what benefit does the crowd bring? One benefit is *diversity*: the crowd has diverse skills, perceptions, and opinions, and even the different user interface presented to the crowd (typically small bits of work) may enable them to see things and do things that the primary end-users of the system do not. Another benefit is *different competence*: the crowd may have abilities that the end-users of the system do not. The VizWiz system demonstrates this property most strongly, since the crowd has vision, but the blind end-users do not. Other kinds of different competence may include language skill, technical expertise, or even physical location in the world. If diversity or different competence are important to the human intelligence needs of your system, then a crowd may be the right tool for the job.

CONCLUSION

Crowd computing draws on the power of people on the Web to do tasks that are hard for individual users or computers to do alone. This paper has presented several examples of crowd computing systems that we have built, in domains ranging from handwriting transcription, to document editing, to assitive technology for the blind, to classroom code reviewing. We have used these example systems to illustrate some of the challenges of crowd computing, including quality control, latency, and incentive management. One of the exciting aspects of crowd computing as a field is its potential for injecting human intelligence into a variety of software systems, and using it as a springboard to artificial intelligence.

ACKNOWLEDGMENTS

Many students and collaborators contributed to this work, including Greg Little, Lydia Chilton, Max Goldman, Jeff Bigham, Michael Bernstein, David Karger, Mark Ackerman, Björn Hartmann, Joel Brandt, Mason Tang, Elena Tatarchenko, Mason Glidden, Kiran Bhattaram, Chris Graves, Juho Kim, Jones Yu, Adam Marcus, Haoqi Zhang, and Joey Rafidi. This work is supported in part by Quanta Computer as part of the Qmulus project, by Xerox Corporation, by the Ford-MIT Alliance, and by NSF under award SOCS-1111124. We are also grateful to over 20,000 Mechanical Turk workers who contributed to the systems we have built over the years.

REFERENCES

Bederson, B. B., and Quinn, A. J. (2011) Web Workers Unite! Addressing Challenges of Online Laborers. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems Extended Abstracts (CHI 2011), pp. 97-106.

Bernstein, M., Karger, D., Miller, R., and Brandt, J. Analytic Methods for Optimizing Realtime Crowdsourcing. In Proceedings of Collective Intelligence 2012.

Bernstein, M.S et al. (2010) Soylent: A Word Processor with a Crowd Inside. In Proceedings of the 23rd annual ACM symposium on User interface software and technology (UIST 2010), pp. 313-322.

Bernstein, M.S, Brandt, J., Miller, R.C. and Karger, D.R. (2011) Crowds in Two Seconds: Enabling Realtime Crowd-Powered Interfaces. In Proceedings of the 24th annual ACM symposium on User interface software and technology (UIST 2011), pp. 33-42.

Bigham, J. et al. (2010) VizWiz: Nearly Real-Time Answers to Visual Questions. In Proceedings of the 23rd annual ACM symposium on User interface software and technology (UIST 2010), pp. 333-342.

Brady, E., Morris, M.R., Zhong, Y., and Bigham, J.P. (2013) Visual Challenges in the Everyday Lives of Blind People. In Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI 2013), pp. 2117-2126.

Cooper, S. et al. (2010) The challenge of designing scientific discovery games. Proceedings of the Fifth International Conference on the Foundations of Digital Games (FDG 2010), pp. 40-47.

Dai, P., Mausam, and Weld, D. S. (2010) Decision-theoretic control of crowd-sourced workflows. In Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2010).

Gould, J.D., Conti, J., and Hovanyecz, T. (1982) Composing letters with a simulated listening typewriter. In Proceedings of the 1982 Conference on Human Factors in Computing Systems (CHI 1982), pp. 367-370.

Hamdaqa, M. and Tahvildari, L. (2012) Cloud computing uncovered: a research landscape. Advances in Computers, v86, pp 41-84.

Howe, J. (2006) The rise of crowdsourcing. Wired. v14, n6, June 2006.

Ipeirotis, P. (2010) Demographics of mechanical turk. Technical report, March 2010. http://www.behind-the-enemy-lines.com/2010/03/new-demographics-of-mechanical-turk.html

Kraut, R.E. and Resnick, P. (2012) Building Successful Online Communities: Evidence-Based Social Design. Cambridge: MIT Press, 2012.

Law, E. and von Ahn, L. (2011) Human Computation. Morgan & Claypool, 2011.

Little, G., Chilton, L.B., Goldman, M., and Miller, R.C. (2010) Exploring iterative and parallel human computation processes. Proceedings of the ACM SIGKDD Workshop on Human Computation (HCOMP 2010), pp. 68-76.

Mason, W. and Suri, S. (2012) Conducting behavioral research on Amazon's Mechanical Turk. Behavioral Research Methods, v44 n1, March 2012, pp. 1-23.

National Research Council (1995). Evolving the High Performance Computing and Communications Initiative to Support the Nation's Information Infrastructure. Washington: National Academies Press, 1995.

National Research Council (2012). Continuing Innovation in Information Technology. Washington: National Academies Press, 2012.

Oleson, D., Sorokin, A., Laughlin, G., Hester, V., Le, J., and Biewald, L. (2011) Programmatic Gold: Targeted and Scalable Quality Assurance in Crowdsourcing. Proceedings of the AAAI Workshop on Human Computation (HCOMP 2011).

Rzeszotarski, J. and Kittur, A. Instrumenting the crowd: using implicit behavioral measures to predict task performance. In Proceedings of the 24th annual ACM symposium on User interface software and technology (UIST 2011), pp. 13-22.

Sheng, V.S, Provost, F., and Ipeirotis, P.G. (2008) Get another label? improving data quality and data mining using multiple, noisy labelers. In Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '08), pp. 614-622.

Silberman, M. et al. (2010) Sellers' problems in human computation markets. Proceedings of the ACM SIGKDD Workshop on Human Computation (HCOMP 2010), pp. 18-21.

Snyder, C. (2003) Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces. Morgan Kaufmann, 2003.

von Ahn, L. and Dabbish, L. (2004) Labeling images with a computer game. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI 2004), pp. 319-326.

von Ahn, L. and Dabbish, L. (2008) Designing games with a purpose. Communications of the ACM, v51 n8, August 2008, pp 58-67.

Yoskovitz, B. (2011) Don't Code What You Can Mechanical Turk. http://www.instigatorblog.com/dont-code/2011/05/06/

.