Mass Collaboration Systems on the World-Wide Web

AnHai Doan[1], **Raghu Ramakrishnan**[2], **Alon Y. Halevy**[3]

[1]University of Wisconsin, [2]Yahoo! Research, [3]Google Inc.

The Age-Old Practice of Mass Collaboration is Transforming the Web and Giving Rise to a New Field

Mass collaboration systems enlist a multitude of humans to help solve a wide variety of problems. Over the past decade, numerous such systems have appeared on the World-Wide Web. Prime examples include Wikipedia, Linux, Yahoo! Answers, Amazon's Mechanical Turk, and much effort is being directed at developing many more.

As is typical for an emerging area, this effort has appeared under many names, including peer production, user-powered systems, user-generated content, collaborative systems, community systems, social systems, social search, social media, collective intelligence, wikinomics, crowd wisdom, smart mobs, crowd-sourcing, and human computation. The topic has been discussed extensively in books, popular press, and academia (e.g., [31, 32, 25, 2, 36, 17, 3, 7]). But this body of work has considered mostly efforts in the physical world (e.g., [31, 32, 25]). Some do consider mass collaboration systems on the Web, but only certain system types (e.g., [34, 30]) or challenges (e.g., how to evaluate users [14]).

This survey attempts to provide a global picture of mass collaboration systems on the Web. We define and classify such systems, then describe a broad sample of systems. The sample ranges from relatively simple well-established systems such as reviewing books to complex emerging systems that build structured knowledge bases to systems that "piggy back" on other popular systems. We then discuss fundamental challenges such as how to recruit and evaluate users, and to merge their contributions. Finally, we discuss future directions. Given the space limitation, we do not attempt to be exhaustive. Rather, we sketch only the most important aspects of the global picture, using real-world examples. The goal is to further our collective understanding – both conceptual and practical – of this important emerging topic.

1. MASS COLLABORATION SYSTEMS

We define mass collaboration (MC) systems, then discuss how to classify them using a set of dimensions.

1.1 Defining MC Systems

Defining MC systems turns out to be surprisingly tricky. Since many view Wikipedia and Linux as well-known MC examples, as a natural starting point, we can say that an MC system enlists a mass of users to *explicitly* collaborate to build a long-lasting *artifact* that is beneficial to the whole community.

This definition however appears too restricted. It excludes for example the ESP game [33], where users *implicitly* collaborate to label images as a side effect while playing the game. ESP clearly benefits from a mass of users. More importantly, it faces the same human-centric challenges of Wikipedia and Linux, such as how to recruit and evaluate users, and to combine their contributions. Given this, it seems unsatisfactory to consider only explicit collaborations; we ought to allow implicit ones as well.

The definition also excludes, for example, an Amazon's Mechanical Turk-based system that enlists users to find a missing boat in thousands of satellite images [20]. Here, users do not build any artifact, arguably nothing is long lasting, and no community exists either (just users coming together for this particular task). And yet, like ESP, this system clearly benefits from users, and faces similar human-centric challenges. Given this, it ought to be considered an MC system, and the goal of building artifacts ought to be relaxed into the more general goal of solving problems. Indeed, it appears that in principle *any* non-trivial problem *can* benefit from mass collaboration: we can describe the problem on the Web, solicit user inputs, then examine the inputs to develop a solution. This system may not be practical (and better systems may exist), but it can arguably be considered a primitive MC system.

Consequently, we do not restrict the type of collaboration nor the target problem. Rather, we view MC as a general-purpose problem solving method. We say that a system is an MC system if it *enlists a mass of humans to help solve a problem defined by the system owners*, and if in doing so, it addresses the following four fundamental challenges: *(1) How to recruit and retain users? (2) What contributions can users make? (3) How to combine user contributions to solve the target problem?* and *(4) How to evaluate users and their contributions?*

Not all human-centric systems address these challenges. Consider a system that manages car traffic in Madison, Wisconsin. Its goal is to coordinate the behaviors of a mass of human drivers (that already exist *within* the system), to minimize traffic jams, say. Clearly, this system does not want to recruit more human drivers (in fact, it wants far fewer of them). We call such systems *mass management (MM) systems.* MM techniques (a.k.a., "crowd coordination" [31]) can be relevant to MC contexts. But the two system classes are clearly distinct.

In this survey we focus on MC systems that leverage the Web to solve the above four challenges (or a significant subset of them). The Web is unique in that it can help recruit a large number of users, enable a high degree of automation, and provide a large set of social software (e.g., email, wiki, discussion group, blogging, and tagging) that MC systems can use to manage their users. As such, compared to the physical world, the Web can dramatically improve existing MC systems and give birth to novel system types.

1.2 Classifying MC systems

MC systems can be classified along many dimensions. In what follows we discuss nine dimensions that we consider most important. The two that immediately come to mind are the *nature of collaboration* and *type of target problem*. As discussed in Section 1.1, collaboration can be explicit or implicit, and the target problem can be any problem defined by the system owners (e.g., building temporary or permanent artifacts, executing tasks).

The next four dimensions refer respectively to how an MC system solves the four fundamental challenges described in Section 1.1: *how to recruit and retain users, what can users do, how to combine their inputs*, and *how to evaluate them?* In Section 3 we will discuss these challenges and the corresponding dimensions in detail. In the rest of this section we discuss instead the remaining three dimensions: degree of manual effort, role of human users, and stand-alone versus piggy-back architectures.

Degree of manual effort: When building an MC system, we must decide how much manual effort is required to solve each of the four MC challenges. This can range from relatively little (e.g., combining ratings) to substantial (e.g., combining code), and clearly also depends on how much the system is automated. Then we must decide how to divide the manual effort between the users and the system owners. Some systems ask the users to do relatively little and the owners a lot. For example, to detect malicious users, the users may simply click a button to report suspicious behaviors, whereas the owners must carefully examine all relevant evidence to determine if a user is indeed malicious. Some systems do the reverse. For example, most of the manual burden of merging Wikipedia edits fall on the users (who are currently editing), not the owners.

Role of human users: We consider four basic roles of humans in an MC system. (a) Slaves: humans help solve the problem in a divide-and-conquer fashion, to minimize the resources (e.g., time, effort) of the owners. Examples are ESP and finding a missing boat in satellite images using Mechanical Turk. (b) Perspective providers: humans contribute different perspectives, which when combined often produce a better solution (than with a single human). Examples are reviewing books and aggregating user bets to make predictions [31]. (c) Content providers: humans contribute self-generated content (e.g., videos on YouTube, images on Flickr). (d) Component providers: humans function as components in the target artifact, such as a social network, or simply just a community of users (so that the owner can sell ads, say). Humans often play multiple roles within a single MC system (e.g., slaves, perspective providers, and content providers in Wikipedia). It is important to know these roles because that may determine how to recruit. For example, to use humans as perspective providers, it is important to recruit a diverse mass where each human can make independent decisions, to avoid "group think" [31].

Stand-alone versus piggy-back: When building an MC system, we may decide to "piggy back" on a well-established system, by exploiting "traces" that users leave in that system to solve our target problem. For example, Google's "Did you mean" and Yahoo's Search Assist utilize the search log and user clicks of a search engine to correct spelling mistakes. Another system may exploit user purchases in an online bookstore (e.g., Amazon) to recommend books. Unlike "stand-alone" systems, such "piggy-back" systems do not have to solve the challenges of recruiting users and deciding what they can do. But they still have to decide on how to evaluate users and their inputs (i.e., traces in this case), and to combine such inputs to solve the target problem.

2. SAMPLE MC SYSTEMS ON THE WEB

Building on the above discussion of MC dimensions, we now discuss MC systems on the Web. We first describe a set of basic system types, then show how deployed MC systems often combine multiple such types.

2.1 Basic System Types

Table 1 shows a set of basic MC system types. The set is not meant to be exhaustive; it shows only those types that have received most attention. From left to right, it is organized by collaboration, architecture, the need to recruit users, and then by the actions users can take. We now discuss the set, starting with explicit systems.

Explicit Systems: These stand-alone systems let users collaborate explicitly. In particular, users can evaluate, share, network, build artifacts, and execute tasks. We discuss these systems in turn.

1. Evaluating: These systems let users evaluate "items" (e.g., books, movies, Web pages, other users) using textual comments, numeric scores, or tags (e.g., [12]).

2. Sharing: These systems let users share "items" such as products, services, textual knowledge, and structured knowledge. Systems that share products and services include Napster, YouTube, CPAN, and the site programmableweb.com (for sharing files, videos, software, and mashups, respectively). Systems that share textual knowledge include mailing lists, Twitter, how-to repositories (e.g., ehow.com, which lets users contribute and search how-to articles), Q&A Web sites (e.g., Yahoo! Answers [5]), online customer support systems (e.g., QUIQ [24], which powered Ask Jeeves' AnswerPoint, a Yahoo! Answers-like site; QUIQ was probably the first to use the term "mass collaboration" in discussing online communities [23]). Systems that share structured knowledge (e.g., relational, XML, RDF data) include Swivel, Many Eyes, Google Fusion Tables,

Nature of Collabo-ration	Architec-ture	Must recruit users?	What users do?	Examples	Target Problems	Comments
Explicit	Stand-alone	Yes	Evaluating • review, vote, tag	• reviewing & voting at Amazon, tagging Web pages at del.ici.ous.com and Google Co-op	Evaluating a collection of items (e.g., products, users)	Humans as perspective providers. No or loose combination of inputs.
			Sharing • items • textual knowledge • structured knowledge	• Napster, YouTube, Flickr, CPAN, programmableweb.com • mailing lists, Yahoo! Answers, QUIQ, ehow.com • Swivel, Many Eyes, Google Fusion Tables, Google Base, bmrb.wisc.edu, galaxyzoo, Piazza, Orchestra	Building a (distributed or central) collection of items that can be shared among users.	Humans as content providers. No or loose combination of inputs.
			Networking	• LinkedIn, MySpace, Facebook	Building social networks	Humans as component providers. Loose combination of inputs.
			Building artifacts • software • textual knowledge bases • structured knowledge bases • systems • others	• Linux, Apache, Hadoop • Wikipedia, openmind, Intellipedia, ecolicommunity • Wikipedia infoboxes/DBpedia, IWP, Google Fusion Tables, YAGO-NAGA,, Cimple/DBLife • Wikia Search, mahalo, Freebase, Eurekster • newspaper at Digg.com, Second Life	Building physical artifacts	Humans can play all roles. Typically tight combination of inputs. Some systems ask both humans and machines to contribute.
			Task execution	• Finding extra-terrestials, elections, finding people	Possibly any problem	
Implicit	Stand-alone	Yes	• play games with a purpose • bet on prediction markets • use private accounts • solve captchas • buy/sell/auction, play massive multiplayer games	• ESP • intrade.com, Iowa Electronic Markets • IMDB private accounts • recaptcha.net • eBay, World of Warcraft	• labeling images • predicting events • rating movies • digitizing written text • building a user community (for purposes such as charging fees, advertising)	Humans can play all roles. Input combination can be loose or tight.
	Piggy back on another system	No	• keyword search • buy products • browse Web sites	• Google, Microsoft, Yahoo • recommendation feature of Amazon • adaptive Websites (e.g., Yahoo! front page)	• spelling correction, epidemic prediction • recommending products • reorganizing a Website for better access	Humans can play all roles. Input combination can be loose or tight.

Figure 1: A sample of basic MC system types on the World-Wide Web

Google Base, many e-science Web sites (e.g., bmrb.wisc.edu, galaxyzoo.org), and many peer-to-peer systems developed in the Semantic Web, database, AI, and IR communities (e.g., Orchestra [10], [29]). Swivel for example bills itself as the "YouTube of structured data", which lets users share, query, and visualize census- and voting data, among others. In general, sharing systems can be central (e.g., YouTube, ehow, Google Fusion Tables, Swivel) or distributed, in a peer-to-peer fashion (e.g., Napster, Orchestra).

3. ***Networking:*** These systems let users collaboratively construct a large social network graph, by adding nodes and edges over time (i.e., homepages, friendships). Then they exploit the graph to provide services (e.g., friend updates, ads, etc). To a lesser degree, blogging systems are also networking systems in that bloggers often link to other bloggers.

A key distinguishing aspect of systems that evaluate, share, or network is that they do not merge user inputs, or do so automatically in relatively simple fashions. For example, evaluation systems typically do not merge textual user reviews. They often merge user inputs such as movie ratings, but do so automatically using some formulas. Similarly, networking systems automatically "merge" user inputs by adding them as nodes and edges to a social network graph. As a result, users of such systems do not need (and in fact often are not allowed) to edit other users' input.

4. ***Building Artifacts:*** In contrast, systems that let users build artifacts such as Wikipedia often merge user inputs "tightly", and require users to edit and merge one another's inputs. A well-known artifact is software (e.g., Apache, Linux, Hadoop). Another popular artifact is textual knowledge bases (KBs). To build such KBs (e.g., Wikipedia), users contribute data such as sentences, paragraphs, Web pages, then edit and merge one another's contributions. The knowledge capture (k-cap.org) and AI communities have studied building such KBs for over a decade. A well-known early attempt is openmind [30], which enlists volunteers to build a KB of commonsense facts (e.g., "the sky is blue"). Recently, the success of Wikipedia has inspired many "community wikipedias", such as Intellipedia (for the US intelligence community) and EcoliHub (at ecolicommunity.org, to capture all information about the E. Coli bacterium).

Yet another popular target artifact is structured KBs.

For example, the set of all Wikipedia infoboxes (i.e., attribute-value pairs such as city-name = Madison, state = WI) can be viewed as a structured KB collaboratively created by Wikipedia users. Indeed, this KB has recently been extracted as DBpedia and used in several applications (see dbpedia.org). Freebase.com builds an open structured database, where users can create and populate schemas to describe topics of interest, then build collections of interlinked topics using a flexible graph model of data. As yet another example, Google Fusion Tables (*tables.googlelabs.com*) lets users upload tabular data and collaborate on it by merging tables from different sources, commenting on data items, and sharing visualizations on the Web.

Several recent academic projects have also studied building structured KBs in an MC fashion. The IWP project [36] extracts structured data from the textual pages of Wikipedia, then asks users to verify the extraction accuracy. The Cimple/DBLife project [1, 7] lets users correct the extracted structured data, exposed in wiki pages, then add even more textual and structured data. Thus, it builds structured "community wikipedias", whose wiki pages mix textual data with structured data (that comes from an underlying structured KB). Other related works include YAGO-NAGA [11], BioPortal [19] and many recent projects in the Web, Semantic Web, and AI communities [3, 18, 4].

In general, building a structured KB often requires selecting a set of data sources, extracting structured data from them, then integrating the data (e.g., matching and merging "David Smith" and "D. M. Smith"). Users can help these steps in two ways. First, they can improve the automatic algorithms of the steps (if any), by editing their code, creating more training data [17], answering their questions [14, 15], or providing feedback on their output [36, 14]. Second, users can manually participate in the steps. For example, they can manually add or remove data sources, extract or integrate structured data, or add even more structured data, data not available in the current sources but judged relevant [7]. In addition, an MC system may perform inferences over its KB to infer more structured data. To help this step, users can contribute inference rules and domain knowledge [27]. During all such activities, users can naturally cross-edit and merge one another's contributions, just like in those systems that build textual KBs.

Another interesting target problem is building and improving systems running on the Web. The project Wikia Search (search.wikia.com) lets users build an open-source search engine, by contributing code, suggesting URLs to crawl, and editing search result pages (e.g., promoting or demoting URLs). Wikia Search was recently disbanded, but similar features (e.g., editing search pages) appear in other search engines (e.g., Google, mahalo.com). Freebase lets users create custom browsing and search systems (deployed at Freebase), using the community-curated data and a suite of development tools (such as the Metaweb query language and a hosted development environment). Eurekster.com lets users collaboratively build vertical search engines called *swickis*, by customizing a generic search engine (e.g., specifying all URLs that the system should crawl). Finally, MOBS, an academic project [15, 14], studies how to collaboratively build data integration systems, those that provide a uniform query interface to a set of data sources. MOBS enlists users to create a crucial system component, namely the semantic mappings (e.g., "location" = "address") between the data sources.

In general, users can help build and improve a system running on the Web in several ways. First, they can edit the system's code. Second, the system typically contains a set of internal components (e.g., URLs to crawl, semantic mappings), and users can help improve these, without even touching the system's code (e.g., adding new URLs, correcting mappings). Third, users can edit system inputs and outputs. In the case of a search engine, for instance, users can suggest that if someone queries for "home equity loan for seniors", the system should also suggest querying for "reverse mortgage". Users can also edit search result pages (e.g., promoting and demoting URLs, as mentioned earlier). Finally, users can monitor the running system and provide feedback.

We note that besides software, KBs, and systems, many other target artifacts have also been considered. Examples include community newspapers built by asking users to contribute and evaluate articles (e.g., Digg) and massive multi-player games that build virtual artifacts (e.g., Second Life, a 3D virtual world partly built and maintained by users).

5. Executing Tasks: The last type of explicit systems that we consider is those that execute tasks. Examples include finding extra-terrestials, mining for gold, searching for missing people [2, 31, 32, 25], and cooperative debugging (cs.wisc.edu/cbi, early work of this project received the ACM Doctoral Dissertation Award in 2005). As a recent well-known example, in the 2008 election the Obama team ran a large online MC operation that asked numerous volunteers to help mobilize voters. To apply MC to a task, we must find task parts that can be "crowd-sourced", such that each user can make a contribution and the contributions in turn can be combined to solve the parts. Finding such parts and combining user contributions are often task specific. "Crowd-sourcing" the parts however can be fairly general, and systems have been developed to assist that process. For example, Amazon's Mechanical Turk can help distribute pieces of a task to a mass of users (and several recent interesting toolkits have even been developed for using Mechanical Turk [13]). It was used recently to search for Jim Gray, a database researcher lost at sea, by asking volunteers to examine pieces of satellite images for any sign of Jim Gray's boat [20].

Implicit Systems: As discussed earlier, such systems let users collaborate implicitly to solve a problem of the system owners. They fall into two groups: stand-alone and piggy-back.

A stand-alone system provides a service such that when using it users implicitly collaborate (as a side effect) to solve a problem. Many such systems exist, and Table 1 lists a few representative examples. The

ESP game [33] lets users play a game of guessing common words that describe images (shown independently to each user), then uses those words to label images. Google Image Labeler builds on this game, and many other "games with a purpose" exist [34]. Prediction markets [25, 31] let users bet on events (e.g., elections, sport events), then aggregate the bets to make predictions. The intuition is that the "collective wisdom" is often accurate (under certain conditions, see [31]), and that this helps incorporate "inside" information that users have. The Internet Movie Database (IMDB) lets users import movies into private accounts (hosted by IMDB). It designed the accounts such that users are strongly motivated to rate the imported movies, as doing so bring many private benefits (e.g., they can query to find all imported action movies rated at least 7/10, or the system can recommend action movies highly rated by people with similar taste). IMDB then aggregates all private ratings to obtain a public rating for each movie, for the benefit of the public. reCAPTCHA asks users to solve captchas to prove they are humans (to gain access to a site), then leverages the results for digitizing written text [35]. Finally, it can be argued that the *target problem* of many systems (that provide user services) is simply to *grow a large community of users*, for various reasons (e.g., personal satisfaction, charging subscription fees, selling ads, selling the systems to other companies). Buy/sell/auction websites (e.g., eBay) and massive multi-player games (e.g., World of Warcraft) for instance fit this description. Here, by simply joining the system, users can be viewed as implicitly collaborating to solve the target problem (of growing user communities).

The second kind of implicit system that we consider is piggy-back systems. Such a system exploits the user traces of yet another system (thus making the users of this latter system implicitly collaborate) to solve a problem. For example, over time many piggy-back MC systems have been built on top of major search engines (e.g., Google, Yahoo!, Microsoft). These systems exploit the traces of search engine users (e.g., search logs, user clicks) for a wide range of tasks (e.g., spelling correction, finding synonyms, flu epidemic prediction, keyword generation for ads [8]). Other examples include exploiting user purchases to recommend products [28], and exploiting click logs to improve the presentation of a Web site [21].

2.2 MC Systems on the Web

We now build on basic system types to discuss deployed MC systems on the Web. Founded on static HTML pages, the Web soon offered many interactive services. Some services serve machines (e.g., DNS servers, Google Map API server), but most serve humans. Many such services do not need to recruit users (in the sense that the more the better). Examples include pay-parking-ticket services (for city residents) and room-reservation services. (We call these mass management systems in Section 1.1.) Many services however face MC challenges, including the need to grow large user bases. For example, online stores such as Amazon want a growing user base for their services, to maximize profits, and startups such as epinions.com grow their user bases for advertising. They started out as primitive MC systems, but quickly improved over time with additional MC features (e.g., reviewing, rating, networking). Then around 2003, aided by the proliferation of social software (e.g., discussion groups, wiki, blog), many "full-fledged" MC systems (e.g., Wikipedia, Flickr, YouTube, Facebook, MySpace) appeared, marking the arrival of "Web 2.0". This Web is growing rapidly, with many new MC systems being developed and non-MC systems adding MC features.

These MC systems often combine multiple basic MC features. For example, Wikipedia primarily builds a textual KB. But it also builds a structured KB (via infoboxes) and hosts many knowledge sharing forums (i.e., discussion groups). YouTube lets users both share and evaluate videos. Community portals often combine all MC features discussed so far. Finally, we note that the Semantic Web, an ambitious attempt to add structure to the Web, can be viewed as an MC attempt to share structured data, and to integrate such data to build a Web-scale structured KB. The World-Wide Web itself is perhaps the largest MC system of all, encompassing everything that we have discussed.

3. CHALLENGES AND SOLUTIONS

We now discuss the key challenges of MC systems: how to recruit users, what contributions they can make, how to combine the contributions, and how to evaluate users and contributions.

1. How to Recruit and Retain Users? Recruiting users is one of the most important MC challenges, for which five major solutions exist. First, we can *require users* to make contributions if we have the authority to do so (e.g., a manager may require 100 employees to help build a company-wide system). Second, we can *pay users.* Mechanical Turk for example provides a way to pay users on the Web to help with a task. Third, we can *ask for volunteers.* This solution is free and easy to execute, and hence is most popular. Most current MC systems on the Web (e.g., Wikipedia, YouTube) use this solution. The downside of volunteering is that it is hard to predict how many users we can recruit for a particular application.

The fourth solution is *to make users pay for service.* The basic idea is to require the users of a system A to "pay" for using A, by contributing to an MC system B. Consider for example a blog website (i.e., system A), where a user U can leave a comment only after solving a puzzle (called a captcha) to prove that U is a human. As a part of the puzzle, we can ask U to retype a word that an OCR program has failed to recognize (i.e., the "payment"), thereby contributing to an MC effort on digitizing written text (i.e., system B). This is the key idea behind the reCAPTCHA project [35]. The MOBS project [14, 15] employs the same solution. In particular, it ran experiments where a user U can access a Web site (e.g., a class homepage) only after answering a relatively simple question (e.g., is string

"1960" in "born in 1960" a birth date?). MOBS then leverages the answers to help build a data integration system. This solution works best when the "payment" is unintrusive or cognitively simple, to avoid deterring users from using system A.

The fifth solution is to *piggy back on the user traces* of a well-established system (e.g., building a spelling correction system by exploiting user traces of a search engine, see Section 1.2). This gives us a steady stream of users. But we must solve the difficult challenge of determining how the traces can be exploited for our purpose.

Once we have selected a recruitment strategy, we should consider how to further encourage and retain users. Many *encouragement and retention (E&R) schemes* exist. We briefly discuss the most popular ones. First, we can provide *instant gratification*, by immediately showing a user how his or her contribution makes a difference [16]. Second, we can provide an *enjoyable experience or a necessary service*, such as game playing (while making a contribution) [33]. Third, we can provide ways to *establish, measure, and show fame/trust/reputation* (e.g., [9, 15, 27, 26]). Fourth, we can set up *competitions*, such as showing top rated users. Finally, we can provide *ownership situations*, where a user may feel he or she "owns" a part of the system, and thus is compelled to "cultivate" that part. For example, zillow.com displays houses and estimates their market prices. It provides a way for a house owner to "claim" his or her house and provide the correct data (e.g., number of beds), which in turn helps improve the price estimation.

The above E&R schemes apply naturally to volunteering, but can also work well for other recruitment solutions. For example, after *requiring* a set of users to contribute, we can still provide instant gratification, enjoyable experience, fame management, etc. to maximize user participation. Finally, we note that deployed MC systems often employ a mixture of recruitment methods (e.g., bootstrapping with "requirement" or "paying", then switching to "volunteering" once the system is sufficiently "mature").

2. What Contributions Can Users Make? In many MC systems the kinds of contributions users can make are somewhat limited. For example, to evaluate, users review, rate, or tag; to share, users add items to a central Web site; to network, users link to other users; to find a missing boat in satellite images, users examine those images.

In more complex MC systems, however, users often can make a far wider range of contributions, from simple "low-hanging fruit" to cognitively complex ones. For example, when building a structured KB, users can add a URL, flag incorrect data, and supply attribute-value pairs (as low-hanging fruit) [6, 7]. But they can also supply inference rules, resolve controversial issues, and merge conflicting inputs (as cognitively complex contributions) [27]. The challenge then is to define this range of possible contributions (and design the system such that it can gather a critical mass of such contributions).

Toward this goal, we should consider four important factors. First, how *cognitively demanding* are the contributions? An MC system often has a way to classify users into groups, such as guests, regulars, editors, admins, and "dictators". We should take care to design cognitively appropriate contribution types for different user groups. Low-ranking users (e.g., guests, regulars) often want to make only "easy" contributions (e.g., answering a simple question, editing 1-2 sentences, flagging an incorrect data piece). If the cognitive load is high, they may be reluctant to participate. High-ranking users (e.g., editors, admins) are more willing to make "hard" contributions (e.g., resolving controversial issues).

Second, what should be the *impact* of a contribution? We can measure the potential impact by considering how the contribution potentially affects the MC system. For example, editing a sentence in a Wikipedia page largely affects only that page, whereas revising an edit policy may potentially affect million of pages. As another example, when building a structured KB, flagging an incorrect data piece typically has less potential impact than supplying an inference rule, which may be used in many "parts" of the MC system. Quantifying the potential impact of a contribution type in a complex MC system may be difficult [14, 15]. But it is important to do so, because we typically have far fewer high-ranking users such as editors and admins (than regulars, say). To maximize the total contribution of these few users, we should ask them to make potentially-high-impact contributions whenever possible.

Third, what about *machine contributions*? If an MC system employs an algorithm for a task, then we want human users to make contributions that are easy for humans, but *difficult for machines*. For example, examining textual and image descriptions to decide if two products match is relatively easy for humans but very difficult for machines. In short, the MC work should be distributed between human users and machines according to what each of them is best at, in a complementary and synergistic fashion.

Finally, the *user interface* should make it easy for users to contribute. This is highly non-trivial. For example, how can users easily enter domain knowledge such as "no current living person was born before 1850" (which can be used in a KB to detect incorrect birth dates, say)? A natural language format (e.g., in openmind.org) is easy for users, but hard for machines to understand and use, and a formal language format has the reverse problem. As another example, when building a structured KB, contributing attribute-value pairs is relatively easy (as Wikipedia infoboxes and Freebase demonstrate). But contributing more complex structured data pieces can be quite difficult for naive users, as this often requires them to learn the KB schema, among others [7].

3. How to Combine User Contributions? Many MC systems do not combine contributions, or do so in a "loose" fashion. For example, current evaluation systems do not combine reviews, and combine numeric ratings using relatively simple formulas. Networking systems simply link contributions (homepages and friendships) to form a social network graph. More complex MC systems, however, such as those that build software, KBs, systems, and games, combine contributions more "tightly". Exactly how this happens is application dependent. Wikipedia for example lets users manually merge edits, while ESP does so automatically, by waiting until two users agree on a common word.

No matter how contributions are combined, a key problem is to decide what to do if users differ, such as when three users assert "A" and two users "not A". Both automatic and manual solutions have been developed for this problem. Current automatic solutions typically combine contributions weighted by some user scores. The work [14, 15] for example lets users vote on the correctness of system components (the semantic mappings of a data integration systems in this case [22]), then combines the votes weighted by the trustworthiness of each user. The work [27] lets users contribute structured KB fragments, then combines them into a coherent probabilistic KB, by computing the probabilities that each user is correct, then weighting contributed fragments by these probabilities.

Manual dispute management solutions typically let users fight and settle among themselves. Unresolved issues then percolate up the user hierarchy. Systems such as Wikipedia and Linux employ such methods. Automatic solutions are more efficient. But they work only for relatively simple forms of contributions (e.g., voting), or forms that are complex but amenable to algorithmic manipulation (e.g., structured KB fragments). Manual solutions are still the currently preferred way to combine "messy" conflicting contributions.

To further complicate the matter, sometimes not just human users, but machines also make contributions. Combining such contributions is difficult. To see why, suppose we employ a machine M to help create Wikipedia infoboxes (as proposed in [36]). Suppose on Day 1 M asserts population = 5500 in a city infobox. On Day 2, a user U may correct this into population = 7500, based on his or her knowledge. On Day 3, however, M may have managed to process more Web data, and obtained higher confidence that population = 5500 is indeed correct. Should M override U's assertion? And if so, how can M explain its reasoning to U? The main problem here is that it is difficult for a machine to enter into a manual dispute with a human user. The currently preferred method is for M to alert U, and then leave it up to U to decide what to do. But this method clearly will not scale with the number of conflicting contributions.

4. How to Evaluate Users and Contributions? MC systems often must manage malicious users. To do so, we can use a combination of techniques on blocking, detection, and deterrence. First, we can block many malicious users by limiting who can make what kinds of contributions. Many e-science MC systems for example allow anyone to submit data, but only certain domain scientists to clean and merge this data into the central database.

Second, we can detect malicious users and contributions using a variety of techniques. Manual techniques include monitoring the system by the owners, distributing the monitoring workload among a set of trusted users, and enlisting ordinary users (e.g., flagging bad contributions on message boards). Automatic methods typically involve some tests. For example, a system can ask users questions for which it already knows the answers, then use the answers of the users to compute their reliability scores [15, 35]. Many other schemes to compute users' reliability/trust/fame/reputation have been proposed (e.g., [9, 26]).

Finally, we can deter malicious users with threats of "punishment". A common punishment is banning. A newer, more controversial form of punishment is "public shaming", where a user U judged malicious is publicly branded as a malicious or "crazy" user for the rest of the community (possibly without U's knowledge). For example, a chat room may allow users to rate other users. If the (hidden) score of a user U goes below a threshold, other users will only see a mechanically garbled version of U's comments, whereas U continues to see his or her comments exactly as written.

No matter how well we manage malicious users, malicious contributions often still seep into the system. If so, the MC system must find a way to undo those. If the system does not combine contributions (e.g., reviews) or does so only in a "loose" fashion (e.g., ratings), undo-ing is relatively easy. If the system combines contributions "tightly", but keeps them "localized", then we can still undo with relatively simple logging. For example, user edits in Wikipedia can be combined extensively within a single page, but kept "localized" to that page (not propagated to other pages). Consequently, we can undo with page-level logging, as Wikipedia does. If the contributions however are "pushed deep" into the system, then undo-ing can be very difficult. For example, suppose an inference rule R is contributed to a KB on Day 1. We then use R to infer many facts, apply other rules to these facts and other facts in the KB to infer more facts, let users edit the facts extensively, and so on. Then on Day 3, should R be found incorrect, it would be very difficult to remove R without reverting the KB to its state on Day 1, thereby losing all good contributions made between Day 1 and Day 3.

At the other end of the user spectrum, many MC systems also identify and leverage influential users, using both manual and automatic techniques. For example, productive users in Wikipedia can be manually identified (e.g., recommended by a user), promoted, and given more responsibilities. As another example, certain users of social networks are influential in that they influence buy/sell decisions of other users. Consequently, some work has examined how to automatically identify these users, then leverage them in viral marketing within a user community [26].

4. CONCLUDING REMARKS

We have discussed MC systems on the World-Wide Web. Our discussion shows that mass collaboration can be applied to a wide variety of problems, and that it raises numerous interesting technical and social challenges. Given the success of current MC systems, we expect that this emerging field will grow rapidly. In the near future, we foresee three major directions: more generic platforms, more applications and structure, and more users and complex contributions.

First, the various systems built in the past decade have clearly demonstrated the value of mass collaboration. The race is now on to move beyond building individual systems, toward building general MC platforms that can be used to develop such systems quickly.

Second, we expect that mass collaboration will be applied to ever more classes of applications. Many of these applications will be formal and structured in some sense, making it easier to employ automatic techniques and to coordinate them with human users. In particular, a large chunk of the Web is about data and services. Consequently, we expect that mass collaboration to build structured databases and structured services (e.g., Web services with formalized input and output) will receive increasing attention.

Finally, we expect that many techniques will be developed to engage an ever broader range of users in mass collaboration, and to enable them, especially naive users, to make increasingly complex contributions, such as creating software programs and building mashups (without writing any code), and specifying complex structured data pieces (without knowing any structured query languages).

5. REFERENCES

[1] The Cimple/DBLife project. `http://pages.cs.wisc.edu/~anhai/projects/cimple`.

[2] Person of the year: You, 2006. Special issue, http://www.time.com/time/magazine/article/0,9171,1569514,00.html.

[3] Wikipedia and artificial intelligence: An evolving synergy, 2008. AAAI-08 Workshop.

[4] Workshop on collaborative construction, management and linking of structured knowledge (CK 2009), 2009. http://users.ecs.soton.ac.uk/gc3/iswc-workshop.

[5] L. A. Adamic, J. Zhang, E. Bakshy, and M. S. Ackerman. Knowledge sharing and yahoo answers: Everyone knows something. In *WWW*, 2008.

[6] X. Chai, B. Vuong, A. Doan, and J. F. Naughton. Efficiently incorporating user feedback into information extraction and integration programs. In *SIGMOD*, 2009.

[7] P. DeRose, X. Chai, B. J. Gao, W. Shen, A. Doan, P. Bohannon, and X. Zhu. Building community wikipedias: A machine-human partnership approach. In *ICDE*, 2008.

[8] A. Fuxman, P. Tsaparas, K. Achan, and R. Agrawal. Using the wisdom of the crowds for keyword generation. In *WWW*, 2008.

[9] J. Golbeck. Computing and applying trust in web-based social network, 2005. Ph.D. Dissertation, University of Maryland.

[10] Z. G. Ives, N. Khandelwal, A. Kapur, and M. Cakir. Orchestra: Rapid, collaborative sharing of dynamic data. In *CIDR*, 2005.

[11] G. Kasneci, M. Ramanath, F. Suchanek, and G. Weikum. The yago-naga approach to knowledge discovery. *SIGMOD Record*, 37(4):41–47, 2008.

[12] G. Koutrika, B. Bercovitz, F. Kaliszan, H. Liou, and H. Garcia-Molina. Courserank: A closed-community social system through the magnifying glass. In *The 3rd Int'l AAAI Conference on Weblogs and Social Media (ICWSM)*, 2009.

[13] G. Little, L. B. Chilton, R. C. Miller, and M. Goldman. Turkit: Tools for iterative tasks on mechanical turk, 2009. Technical Report. Available from glittle.org.

[14] R. McCann, A. Doan, V. Varadarajan, and A. Kramnik. Building data integration systems: A mass collaboration approach. In *WebDB*, 2003.

[15] R. McCann, W. Shen, and A. Doan. Matching schemas in online communities: A web 2.0 approach. In *ICDE*, 2008.

[16] L. McDowell, O. Etzioni, S. D. Gribble, A. Y. Halevy, H. M. Levy, W. Pentney, D. Verma, and S. Vlasseva. Mangrove: Enticing ordinary people onto the semantic web via instant gratification. In *ISWC*, 2003.

[17] R. Mihalcea and T. Chklovski. Building sense tagged corpora with volunteer contributions over the web. In *RANLP*, 2003.

[18] N. F. Noy, A. Chugh, and H. Alani. The CKC challenge: Exploring tools for collaborative knowledge construction. *IEEE Intelligent Systems*, 23(1):64–68, 2008.

[19] N. F. Noy, N. Griffith, and M. A. Munsen. Collecting community-based mappings in an ontology repository. In *ISWC*, 2008.

[20] M. Olson. The amateur search. *SIGMOD Record*, 37(2):21–24, 2008.

[21] M. Perkowitz and O. Etzioni. Adaptive web sites. *CACM*, 43(8), 2000.

[22] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.

[23] R. Ramakrishnan. Collaboration and data mining, 2001. Keynote talk, KDD.

[24] R. Ramakrishnan, A. Baptist, V. Ercegovac, M. Hanselman, N. Kabra, A. Marathe, and U. Shaft. Mass collaboration: A case study. In *IDEAS*, 2004.

[25] H. Rheingold. *Smart Mobs*. Perseus Publishing, 2003.

[26] M. Richardson and P. Domingos. Mining knowledge-sharing sites for viral marketing. In *KDD*, 2002.

[27] M. Richardson and P. Domingos. Building large knowledge bases by mass collaboration. In *K-CAP*, 2003.

[28] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW*, 2001.

[29] R. Steinmetz and K. Wehrle, editors. *Peer-to-Peer Systems and Applications*, volume 3485 of *Lecture Notes in Computer Science*. Springer, 2005.

[30] D. G. Stork. Using open data collection for intelligent software. *IEEE Computer*, 33(10):104–106, 2000.

[31] J. Surowiecki. *The Wisdom of Crowds*. Anchor Books, 2005.

[32] D. Tapscott and A. D. Williams. *Wikinomics*. Portfolio, 2006.

[33] L. von Ahn and L. Dabbish. Labeling images with a computer game. In *Proc. of CHI*, 2004.

[34] L. von Ahn and L. Dabbish. Designing games with a purpose. *Communications of the ACM*, 51(8):58–67, 2008.

[35] L. von Ahn, B. Maurer, C. McMillen, D. Abraham, and M. Blum. recaptcha: Human-based character recognition via web security measures. *Science*, 321(5895):1465–1468, 2008.

[36] D. S. Weld, F. Wu, E. Adar, S. Amershi, J. Fogarty, R. Hoffmann, K. Patel, and M. Skinner. Intelligence in wikipedia. In *AAAI*, 2008.