Matching Schemas in Online Communities: A Web 2.0 Approach

Robert McCann[1], **Warren Shen**[2], **AnHai Doan**[2]

[1]Microsoft, [2]University of Wisconsin-Madison

robert.mccann@microsoft.com, {whshen,anhai}@cs.wisc.edu

***Abstract*— When integrating data from multiple sources, a key task that online communities often face is to match the schemas of the data sources. Today, such matching often incurs a huge workload that overwhelms the relatively small set of volunteer integrators. In such cases, community members may not even volunteer to be integrators, due to the high workload, and consequently no integration systems can be built. To address this problem, we propose to enlist the multitude of users in the community to help match the schemas, in a Web 2.0 fashion. We discuss the challenges of this approach and provide initial solutions. Finally, we describe an extensive set of experiments on both real-world and synthetic data that demonstrate the utility of the approach.**

I. Introduction

The World-Wide Web is teeming with communities, such as those of movie fans, database researchers, bioinformatists, intelligence analysts, and so on. As such communities proliferate, research on their data management challenges has attracted increasing attention [1], [2], [3], [4], [5]. A key challenge is to integrate data from multiple community-related sources. For example, the community of real estate agents in the Great Lakes region may want to build a system that integrates all real-estate sources in that area. As another example, the database community may want to integrate all information about publications, from DBLP, Google Scholar, and researchers' homepages, among others.

Today, integrating such data within a community is largely shouldered by a relatively small set of volunteers, henceforth called *builders*. To integrate the data, a key task that builders often face is to establish semantic correspondences called *matches* across the schemas of the data sources, such as location = address and name = concat(fname,lname). To solve this task, builders often employ a matching tool to find match candidates, then examine and repair the candidates to obtain the correct matches.

Much progress has been made in schema matching, and many matching tools have been proposed (see [6], [7] for recent surveys, and also [8], [9], [10], [11], [12]). However, no robust, highly accurate tool has yet been found. Consequently, schema matching still often incurs a huge workload that overwhelms the small set of builders. Worse, community members may not even volunteer to be builders, because the workload is just too high, and so no integration system can be built. Consequently, to facilitate the widespread deployment of integration systems in online communities, it is crucial to develop solutions that reduce the schema-matching burden of system builders.

In this paper we explore such a solution. Our key idea is to enlist the multitude of community members (i.e., *users*) to help the builders match schemas. Specifically, suppose the builders apply a tool M to match two schemas S and T. Then we can modify M so that during the matching process it can ask users relatively simple questions, then learn from the answers to improve matching accuracy, thereby reducing the matching workload of the builders.

For example, suppose M has predicted that attribute monthly-fee-rate of schema S is of the type DATE (and hence can benefit from a specialized date matcher), but it is not entirely confident in that prediction. Then it can ask the users to verify that prediction, by posing the question "is monthly-fee-rate of the type DATE?". As another example, suppose M determines that it can significantly improve matching accuracy if it knows whether lot-area is always greater than house-size. Then M can ask the users "is lot-area always greater than house-size?". As yet another example, suppose M has produced the match lot-id = ad-id, but is not entirely confident in that match. Then, it can ask users to verify that match, by posing the question "does lot-id match ad-id?".

As described, this Web 2.0 approach to schema matching is promising: it can harness the collective community feedback to help significantly reduce the matching workload of the system builders. Realizing this approach, however, raises several challenges. In the rest of the paper we elaborate on these challenges and provide initial solutions.

We begin by defining the problem of Web 2.0 schema matching for online communities. Specifically, our goal is to leverage community users to significantly reduce *the workload of the builders*, while keeping the average workload per user minimal.

We then consider the challenges of which questions to ask users, and how to pose those questions. Building on the principle of selecting questions that are relatively easy for human users, but difficult for "machines", we consider questions that help a matching tool verify intermediate predictions (e.g., bday is of type DATE), learn simple domain integrity constraints (e.g., lot-area is always greater than house-size), and verify final match predictions (e.g., house-size = sqft). We then show how to generate such questions and pose them to users.

Next, since community users can be malicious or ignorant,

we examine the challenge of evaluating their reliability and combining their answers. We propose a conceptually simple, yet effective solution. This solution classifies users into trusted and untrusted, based on their answers to a set of evaluation questions (with known answers), then combines the answers of the trusted users using a voting scheme. We compare and contrast this solution with more sophisticated ones.

In the next step, we discuss how to solicit user participation. We show that the common scheme of volunteering (employed by most current mass collaboration works) can also work in our community context, then propose a novel scheme in which users "pay" by answering a few questions in order to use certain services.

We then describe experiments with both real-world and synthetic data that demonstrate that the above Web 2.0 approach can leverage minimal workload per user to improve matching accuracy, thereby significantly reducing the workload of the builders. In particular, we describe our experience in building a small data integration system over 10 real-world book store sources using 132 users, with very little work per user and builder.

Finally, for future work, we briefly discuss how the current work can be applied to other application settings, (e.g., enterprise schema matching, best-effort data integration), and other problems (e.g., source discovery, wrapper construction, entity matching, and matching pictures with persons).

II. Related Work

Our work is most related to mass collaboration research, which enlists a multitude of Internet users to build software artifacts (e.g., *Linux*), knowledge bases (e.g., the online encyclopedia *wikipedia.com*, see also [13], [14]), review and technical support websites (e.g., *amazon.com, epinions.com, quiq.com*, [15]). The research has also addressed improving the accuracy of a range of algorithms (e.g., ranking function in search engines, recommender systems [16]).

Our solution also enlists multiple users, but for the goal of improving the accuracy of schema matching tools for online communities. Furthermore, the above mass collaboration applications rely largely on volunteer users. In contrast, we can also utilize users via a "payment" scheme (see Section VI), which can potentially be applied to other mass collaboration applications. Finally, many mass collaboration works have combined user answers in ad-hoc ways. In contrast, we provide a principled solution to this problem.

Mass collaboration approaches to data management have also recently received increasing attention in the database community (e.g., Web 2.0 track at ICDE-08, mass collaboration panel at VLDB-07, see also [17], [18], [19], [20], [21], [22]). Our work here contributes to this emerging direction.

Our work is also related to active learning (e.g., [23]). Active learning however usually involves only a *single* user, whereas our work involves *multiple* users, and thus must address the challenge of combining their noisy answers. Our solution is therefore a form of *collective active learning*.

Within data integration contexts, several works [24], [25] have provided sophisticated example-driven mechanisms to interact with users, for schema and instance-level integration. The interaction however is limited to a *single user* (who is presumably the builder), whereas we consider interacting and learning from multiple users. As far as we know, our work is the first to take such a mass collaboration approach to schema matching, a topic that has received much attention [6].

Finally, we have reported isolated parts of this work in several workshop and poster papers [26], [27]. This paper provides the first in-depth description of our overall approach.

III. Background & Problem Definition

We now provide a brief background on schema matching, then define the Web 2.0 style matching problem considered in this paper. For simplicity, we consider only relational schemas (our solution however generalizes to more complex data representations, e.g., ontologies, see Section VIII). To match such schemas, many tools have been proposed. Most such tools find *1-1 matches*, e.g., location = address. Some recent tools can also find *complex matches*, e.g., name = concat(fname,lname). In this paper we will consider both types of matching tools.

Regardless of the type of matches produced, when applied to two schemas S and T, such a matching tool M often treats one schema, say S, as *the source schema*, and the other schema, T, as the *target schema*. For each attribute of source schema S, e.g., sqft, M then produces a list of candidate matches, ranked in decreasing order of confidence, such as:

sqft = ad-id
sqft = lot-area
sqft = house-size
...

The builder B (henceforth we assume just a single builder, for ease of exposition) then examines such a list in a *top-down* fashion to find the correct match. If this match is not in the list, B may try to provide some feedback to M, and "coax" it into producing a new ranked list. When all else fails, B must manually find the correct match, by painstakingly examining the schemas and any associated documentation[1].

Thus, ideally we would like the matching tool M to be *highly accurate* and *robust*, in that it produces the correct match in the top few matches of most ranked lists, across a broad range of matching scenarios. Unfortunately no such matching tool exists today. The fundamental reason is that matching is an inherently *knowledge-intensive* activity. To match two schemas accurately, we often need a significant amount of knowledge about the attributes and the domain of the schemas. However, such knowledge is often absent from the schemas and data to be matched, causing problems for even the most sophisticated current matching tools. Consequently, schema matching remains a brittle business. When

[1] After finding the semantic matches, B may decide to elaborate them into mappings – which are full-fledged SQL queries that express how to transform data from one schema to the other – using a tool such as Clio [25]. This mapping-creation step however is outside the scope of this paper.

Schema S

address	lot-area	price	house-size	num-beds	num-baths	contact	ad-id	month-posted	year-posted
Atlanta, GA	7,000	360,000	2,500	3	2	Mike Brown	32	June	2004
Raleigh, NC	8,500	430,000	3,000	4	2	Jean Laup	15	September	2004

Schema T

house-loc	approx-sqft	post-price	bdrms	bthrms	tax	agent-name	monthly-fee-rate	lot-size
Denver, CO	3,500	550,000	3	3	0.007	Laura Smith	7	1900
Portland, OR	2,500	370,800	3	2	0.008	Dan Kress	5	2000

Fig. 1. Simplified real-estate schemas that are used in our examples in this paper.

much knowledge is available (e.g., in the schemas, data, and available auxiliary sources), and the matching tool can make use of this knowledge, then it can do well. Otherwise it does badly.

Thus, in a sense our goal is to harness the community users to inject more knowledge into the matching process. We want to do so with minimal workload for each user, otherwise they could be reluctant to participate. We can now state our problem as follows.

Problem Statement: *Let W be the workload of a builder B for the task of finding the correct matches between two given schemas S and T, using a matching tool M. Then our goal is to significantly reduce this workload W. Toward this goal, develop a solution that leverages the population of community users $\mathcal{U}$ to significantly improve the accuracy of matching S and T, while keeping the workload of each user $U \in \mathcal{U}$ minimal.*

Intuitively, we can improve matching accuracy by trying to leverage community users to "move" the correct matches up in the ranked lists of matches, to be the top matches or as close to the top as possible. This way, the builder B does not have to examine deep into the ranked lists, thereby saving labor.

Our solution, described in the next few sections, follows the above strategy. It is also important to note that the goal of that solution is to reduce the *builder workload*, not the *total workload* (of both the builder and users). We discuss this issue further in Section VIII-A.

IV. Generating & Posing Questions

We now describe our solution to the above problem. In this section we describe how to generate and pose questions to users. The next section describes how to handle noisy users. Section VI discusses how to entice users to participate. Section VII then shows how to put all of these together in the final system. For space reasons, we can only discuss the key ideas behind the solution, and refer the reader to [28] for a detailed description.

A. Generating Questions to Ask Users

Intuitively, when extending a tool to learn from users, we want to ask questions that (a) significantly impact the tool's accuracy, and (b) are relatively easy for human users, but difficult for machines (i.e., the tool) to answer. In schema matching contexts, many such question types can be asked. As a first step, in this paper we will consider those that can help a matching tool (a) verify intermediate predictions (made internally), (b) learn useful domain integrity constraints, and (c) verify final match candidate predictions.

In what follows we discuss how to generate questions of the above types, first for the case of 1-1 schema matching and then for complex matching. Since the matching tool must interact with users in *real time*, rather than trying to find the *best* set of such questions, we focus instead on *quickly* finding a *reasonable* set (that help reduce the builder's workload). Throughout the paper, we will use as a running example the scenario of matching the two tiny real-estate schemas S and T in Figure 1.

Is attribute ***monthly-fee-rate*** of type ***MONTH***?

agent-name	monthly-fee-rate	lot-size
Laura Smith	7	7,750
Dan Kress	5	6,500

YES | NO | DON'T KNOW | POSTPONE

Fig. 2. A sample question to verify a type prediction.

Generating Questions for 1-1 Matching

We modify a 1-1 matching tool M to generate questions as follows.

1. Questions to Verify Intermediate Predictions: During the matching process, a tool such as M often makes myriad intermediate predictions. A very common kind of such prediction assigns schema attributes into pre-defined *types* such as person name, date, phone, price, etc. Correct typing can be exploited to boost matching results, but conversely, wrong typing can drastically reduce accuracy [29]. Hence, we leverage users to help verify typing predictions.

Specifically, we first apply M to match S and T, and record the type predictions made by M (e.g., "monthly-fee-rate is of type MONTH"), as well as the number of components of M (e.g., base matchers, combiner, constraint handler, etc., see [30], [31]) that rely on each type prediction.

Next, to ensure a reasonable user workload, we select only the k type predictions (where k is pre-specified) that we believe would make the most impact on matching accuracy. We approximate the impact of a type prediction p by the number of components of M that rely on p.

Finally, we ask users to verify these k type predictions. For example, to verify "monthly-fee-rate is of type MONTH", we generate and pose the question in Figure 2. Users can answer "yes", "no", "don't know", or "postpone". Section V discusses how to merge the "yes" and "no" answers (that in most cases come from different users) to obtain a final answer to the question. Note that if a user answers "postpone", then we ask no further question from that user in that session (see Section VI).

2. Questions to Learn Domain Constraints: During the matching process, knowing whether a certain domain integrity constraint holds (e.g., s-date is always less than e-date) can make a drastic impact on the matching accuracy [31], [29].

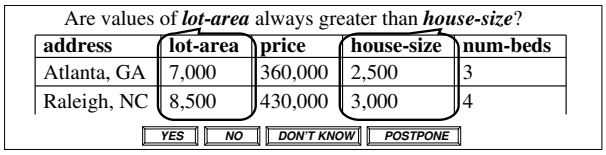


Fig. 3. A sample question to verify a domain constraint.

Hence, after verifying type predictions, we generate questions to help M learn certain domain integrity constraints.

Specifically, we first rerun M on S and T, leveraging all previously verified type predictions. Next, we compile all constraints of certain types. As a first step, in this paper we consider only two types: *comparison* and *uniqueness*, which have been exploited in recent matching work [31], [29]. Comparison constraints claim that the values of an attribute A are always greater than or equal to those of an attribute B (e.g., lot-area vs. house-size). Uniqueness constraints claim that if two attributes of S match a single attribute of T (e.g., house-loc), then they must be the same attribute.

We then select the top v constraints that appear to have the most impact on matching accuracy. For each constraint c, assuming c is true, we rerun M to obtain the new matching result. We then approximate the impact of c as the difference between the new and old matching results (we omit the details of computing this difference for space reasons). Finally we ask users to verify each of the selected constraints. For example, Figure 3 shows a question that asks users to verify the constraint "lot-area is greater than house-size".

3. Questions to Verify Final Match Predictions: In the last step we ask users to verify the matches predicted by M. First, we rerun M on S and T, taking into account the previously verified type predictions and the learned domain constraints. Recall that for each attribute of schema S, M produces a ranked list of matches, in decreasing order of confidence. Figure 4.a shows for example such a ranked list for approx-sqft.

We then ask users to verify the top-ranked match: approx-sqft = ad-id, by posing the question in Figure 5. If users say no, then we ask them to verify the next match: approx-sqft = lot-area, and so on, until users have said yes to a match, or we have reached a pre-specified depth of k matches.

Suppose users have just said yes to the match approx-sqft = house-size. We then revise the ranked list to create a final ranked list that we will output. Specifically, we bring the above match to the top of the current ranked list, then rerank the other matches using any user answers we have collected: if two matches x and y both receive at least v user answers each (with v pre-specified), and the fraction of users saying yes to x is higher than that to y, then we rank x higher than y. Figure 4.b shows a possible revised ranked list for approx-sqft, which we then output as the final ranked list for approx-sqft.

Thus, the intuition is that user feedback will have revised the ranked list such that the likely matches "bubble" to the top of the list (e.g., house-size and lot-area in this case),

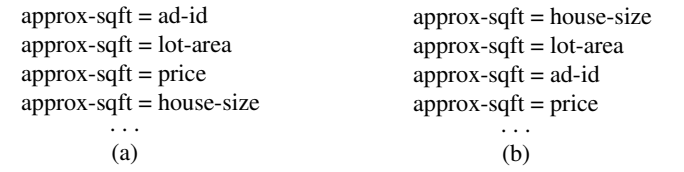


Fig. 4. (a) A sample ranked list, and (b) its revision after soliciting user feedback.

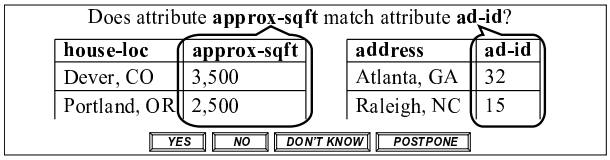


Fig. 5. A sample question to verify a predicted 1-1 match.

later saving the builder from having to examine deep into the ranked list to find the correct match.

Generating Questions for Complex Matching

If M finds complex matches (e.g. [29]), then we modify it as follows. First, we still leverage users to correct wrong typing predictions and to learn domain constraints, in a fashion similar to modifying 1-1 matching tools.

We then try to verify the predicted complex matches. However, a direct verification of complex matches is often difficult. For example, using systems such as [29] we may obtain the ranked list of matches in Figure 6.a for price. Asking users to verify such matches would be cognitively too heavy and time consuming, because they would have to perform, e.g., arithmetic computation, among others. Hence, we allow users to indirectly evaluate the matches as follows.

Step 1: Take the top k matches for price (we set k to 10 in our experiments). Compile the set of all attributes that appear in these matches. For example, if k = 3, then the set of attributes in the top 3 matches for price is {list-price, t-rate, disc-fac, agent-id}.

Step 2: For each attribute in the above set, ask users if a complex match for price can possibly involve that attribute. For example, Figure 7 shows a question that asks users if the complex match for price can involve disc-fac.

Step 3: For price, suppose users have indicated that the attributes that can be in its matches are list-price and t-rate. Then we rerun the matching tools for price, focusing on formulas that combine these two attributes. We then select the top few matches from the output. Suppose we select

price = list-price * (1 + t-rate/100)
price = list-price.

Then now we can place these matches on top of the previous complex matches, to form the new ranked list shown in Figure 6.b.

Step 4: In the last step, we clean the above matches by dropping terms that involve attributes that users have identified *not* to be included, which are disc-fac and agent-id in this case, then merge identical matches. Figure 6.c shows the final revised list of matches to be output.

price = list-price * (1 + t-rate) * disc-fac
price = list-price * t-rate – disc-fac
price = list-price + 0.01 * t-rate * agent-id
price = list-price * (1 + t-rate/100) – disc-fac
• • •

(a)

price = list-price * (1 + t-rate/100)
price = list-price
price = list-price * (1 + t-rate) * disc-fac
price = list-price * t-rate – disc-fac
price = list-price + 0.01 * t-rate * agent-id
• • •

(b)

price = list-price * (1 + t-rate/100)
price = list-price
price = list-price * (1 + t-rate)
price = list-price * t-rate
• • •

(c)

Fig. 6. (a) A sample ranked list of complex matches, and (b)-(c) its revisions after user feedback.

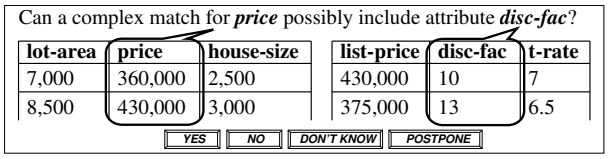


Fig. 7. A sample question to verify the involvement of an attribute in a complex match.

B. Posing Questions to the Users

When posing each question Q (generated as described earlier) to users, we try to provide enough context so that users can quickly grasp the meanings of the attributes mentioned in Q and thus answer Q correctly.

Specifically, for each attribute s mentioned in Q, we first mention s by name in the concrete question posed to the users (at the top of the question window). Next, we select m attributes (currently set to 3) that are closest to s in the schema (including s itself), and show a snapshot of n tuples (currently set to 5) projected onto these attributes, in the form of a table. We then highlight the column for s in the table, highlight mention of s in the question, and visually link the two. All sample questions shown so far in this paper are generated in this format.

For the experiments described in Section VIII, we found that contexts generated as above were helpful, in that using them users were able to answer most questions accurately, thereby improving matching accuracy. In future work we will explore more sophisticated contexts, such as showing users both the original Web query interfaces (if any) and the schemas obtained from them, and both "before" and "after" query results (so that users can gauge the potential impact of their answers).

V. Managing Noisy Users

We have described how to generate and pose questions. Since online communities often contain "noisy" (e.g., malicious or ignorant) users, whose answers to such questions cannot be trusted, developing a way to manage such users is critical. We now describe our solution to this problem.

A. A Spectrum of Solutions

To manage noisy users, a reasonable solution is to develop a model of user reliability, then use the model to combine users' answers (to each question). Many such models can be developed. A basic one can simply classify users as *trusted* or *untrusted*, then consider only the answers of trusted users. Such a model is easy to understand and tune, and takes relatively little training data (e.g., data of the form $\langle$question, correct answer, user answer$\rangle$) to infer the trustworthiness of each user. However, the model cannot utilize the full range of user feedback. Consider for example a user U_{bad} who always answers questions *incorrectly*. The above model would classify U_{bad} as untrusted, and ignore his answers. Intuitively, this is a "waste", because if the model can learn that U_{bad} always gives incorrect answers, then it can make use of that fact in deriving correct answers for the questions.

At the other end of the spectrum, we can develop sophisticated models, such as one that employs a Bayesian network to model all major factors that affect user answers. These models typically estimate real-valued reliability scores for users, then use such scores to combine answers. The models are powerful in that they can utilize the full range of user feedback, e.g., exploit the answers of user U_{bad} described earlier. On the other hand, they are difficult to understand and tune, and often require a large amount of training data for accurate estimation of reliability scores.

In this paper, as a first step, we consider simple trusted/untrusted models, for three reasons. First, such models require relatively little training data (as described earlier), thereby minimizing the workload of each user, an important requirement in our context. Second, it turned out that in practice (based on our experiments) users such as U_{bad} described earlier often behave in a highly unreliable fashion, e.g., switching unpredictably between answering questions incorrectly and correctly. Hence, it is unclear how to effectively model and exploit such users, even with current sophisticated models. Consequently, we opt instead to use a simpler model that ignores such users. Third, again from our experiments, we found that it is critical that the system builder understand how noisy users are managed, and be able to tune the parameters of the management scheme, to adjust to the situation at hand (e.g., tightening the trustworthiness criteria if the user population appears to be largely unreliable).

In the rest of this section, we describe the basic trusted/untrusted model that we currently employ.

B. Classifying Users as Trusted or Untrusted

To classify users as trusted or untrusted, we first create a set of questions with known answers (henceforth called *evaluation questions*). The builder can make up these questions, or obtain them from a small set of matches that he or she has manually verified.

Next, we ask users these evaluation questions (randomly mixed in with real questions, see Section VII). Suppose a user U has answered a evaluation questions, and b out of a correctly. Then we classify U as trusted if $a \geq v_1$ and

$b/a \geq v_2$, for pre-specified v_1 and v_2. Intuitively, we want to make sure U has answered at least v_1 evaluation questions, and at least a sizable fraction v_2 of those correctly, before we can trust U.

C. Combining Answers of Trusted Users

From now on, we will use the term "answer" to refer to an answer from a trusted user, when there is no ambiguity. Consider a "real" question Q (asked by matching tool M, as described in Section IV). As users answer this question, we will in essence see a "stream" of answers to Q, as time progresses.

We then monitor and stop this stream (i.e., no longer solicit answer for Q) as soon as

- we have collected at least v_3 answers and the gap between the majority and minority answer is at least g, or
- we have collected v_4 answers, where $v_4 > v_3$.

In either case, we return the majority answer as the final answer for Q.

Intuitively, we can confidently output the majority answer if it has achieved a statistically significant lead over the minority answer (signified by the gap g, see more below). But in any case, we do not want users to devote more than a fixed limit v_4 of answers to any single question. Note that all answers above refer to those from trusted users.

The following theorem shows that the above algorithm produces correct answers with high probability:

Theorem 1: For any question Q, the above algorithm will halt and return the correct answer with probability at least $1 - \mathcal{E}_1 - \mathcal{E}_2$, *where* $\mathcal{E}_1$ *is* $\sum_{j=\left\lceil \frac{v_3+g}{2} \right\rceil}^{\left\lfloor \frac{v_4+g}{2} \right\rfloor} \binom{2j-g}{j} p^{(j-g)}(1-p)^j +$ $+ \sum_{j=\left\lceil \frac{v_3+g+1}{2} \right\rceil}^{v_3} \binom{v_3}{j} p^{(v_3-j)}(1-p)^j$, $\mathcal{E}_2$ *is* $\sum_{j=\frac{v_4+1}{2}}^{\left\lfloor \frac{v_4+g}{2}-1 \right\rfloor} \binom{v_4}{j} p^{(v_4-j)}(1-p)^j$, v_1, v_2, v_3, g, *and* v_4 *are as defined in Section V-B and Section V-C, and* p *denotes the expectation of the distribution* $P(r) \cdot \sum_{i=v_1 v_2}^{v_1} \binom{v_1}{i} r^i (1-r)^{(v_1-i)}$, *where* $P(r)$ *is the distribution of individual user probabilities of correctness induced by soliciting answers from the user population.* □

Intuitively, Theorem 1 bounds the probability that a given question will converge to the incorrect answer. $\mathcal{E}_1$ bounds the probability that our gap criteria g will be reached with an incorrect majority answer. $\mathcal{E}_2$ bounds the probability of the majority answer being incorrect after reaching our maximum number of answers v_4. The overall quality of user answers is quantified by p, the probability that, through the process of soliciting feedback from a population of distinct users with individual "reliabilities", a given answer will be correct. Specifically, if each user answers correctly with some individual probability r (their "reliability") and our process of soliciting participation produces a stream of user answers that were generated by individual probabilities with distribution $Pr[r]$, then the answers will be correct with probability equal to p as stated in the theorem (evaluating and ignoring untrusted users boost the reliabilities for these answers by the $\sum_{i=v_1 v_2}^{v_1} \binom{v_1}{i} r^i (1-r)^{(v_1-i)}$ factor).

Our experiments show that the algorithm often halts short of reaching the limit v_4 on number of answers per question, thereby minimizing user effort. It also merges user answers correctly with very high probabilities. For example, suppose that user reliability scores are distributed following some Normal distribution with mean 0.75 (recall that a user with reliability score 0.6, say, answers on average 6 questions correctly out of 10 questions). If we set the user evaluation thresholds to $v_1 = 8$ and $v_2 = 0.7$ and the answer convergence criteria to $g = 6$, $v_3 = 8$, and $v_4 = 15$, then the above theorem states that we will output the correct answer for question Q with probability greater than 0.99.

VI. Soliciting User Participation

Our final challenge is to secure user participation. To this end, we discuss two solutions: volunteering and "payment".

Volunteering: Most mass collaboration schemes (see the related work section) have solicited user participation via volunteering. The success of many volunteering-based online collaborative projects suggests that this scheme can also work in our data integration context for online communities, provided that the workload per volunteer remains relatively low.

"Payment": In addition to volunteers, we propose that the builder also obtain user participation via "payment" schemes, whenever appropriate. The basic idea is to ask users to "pay" for certain services by answering questions, then use the answers to help the builder.

For example, suppose the builder has built an integration system A over three data sources. Then when a user poses a query to A, A can ask the user to answer a question before displaying the query result. The builder then uses such answers to help improve A (e.g., by adding a fourth source).

In a slightly more complex scheme, suppose there exists another system B that has "agreed" to help A. Then the users of B can also be asked to "pay" for using B by answering questions. The answers can then be used to maintain and expand A. We call such system B a *helper application*. Section VIII-D describes our experiments that used a database course homepage as a helper application.

To make the "payment" scheme work, we impose four requirements. First, the helper application must either be a *monopoly* (i.e., not offered anywhere else) or have *better quality* than the competitors. This way the users either have to "pay" or are willing to tolerate question answering. Second, the workload of each user must not exceed a certain threshold (e.g., three questions per day). Third, the questions must be asked at a predictable time (preferably only at the start of a user session). Finally, each user must have the ability to *defer* question answering to a more appropriate time (in effect owing the system the answers). Our experiments (Section VIII) suggest the feasibility of "payment" schemes with these requirements in practice.

VII. Putting It All Together: The Overall System

We now briefly describe the overall system, which combines the technologies discussed earlier, and consist of three main

TABLE I: MATCHING TASKS FOR OUR EXPERIMENTS.

Task Types	Domains	Description
1-1 schema matching	Book query interfaces	10 interfaces, total 65 attributes
	Real estate I	2 schemas, with 55-44 attributes
	Company listings	2 taxonomies, with 330-115 attributes
Complex matching	Real estate II	2 schemas, with 19-32 attributes
	Inventory	2 schemas, with 34-49 attrs

modules: question manager, answer solicitor, and user manager (see [28] for a detailed description).

During the matching process, tool M generates questions (as described in Section IV), and sends them to the *question manager*. This module interacts with the answer solicitor and the user manager to decide which users to ask which questions. The module also monitors the answers that each question Q has received. As soon as the answers for Q have converged to a final answer (see Section V), the question manager sends that final answer back to M. Once M has received the answers to all questions it poses, and have finished matching the schemas, the builder verifies and corrects the predicted matches, to obtain the final correct matches.

The *answer solicitor* monitors the users in the environment. Whenever there is an opportunity to ask a user U a question (e.g., through voluntary participation or "payment" scheme, as described in Section VI), it contacts the *user manager*, which keeps track of users' workload, question answering history, and trust/untrusted status. If the workload quota of user U has been reached (e.g., U has answered three questions, the maximum allowed per day), then the user manager does nothing. Otherwise, it checks to see if user U can be *trusted*. If yes then it notifies the question manager to ask U a question; otherwise it evaluates U further, by sending U an evaluation question (i.e., one with known answer), to see if U answers that question correctly.

VIII. EMPIRICAL EVALUATION

We now empirically show that the above Web 2.0 approach can leverage minimal workload per user to improve matching accuracy, thereby significantly reducing the workload of the builder. We show that even when users cannot answer questions perfectly, their answers still contribute toward reducing this workload. We demonstrate that our solution scales up to large and diverse populations. Finally, we describe our experience of building a real-world data integration system over 10 online book stores, using a population of 132 users.

Data Sets, Matching Tasks, and Automatic Tools: Table I describes five schema-matching tasks and associated data sets. Book Query Interfaces finds 1-1 matches between 10 query interfaces and a given global schema (having five attributes) of a data integration system. Real Estate I finds 1-1 matches between two schemas for house listings. Company Listing finds 1-1 matches between two very large taxonomies, with 115-330 attributes. Finally, Real Estate II and Inventory find complex matches between pairs of schemas. We obtained data from $invisibleweb.com$ and from the experiments described in [30], [31], [29], [32]. We match the schemas using variants of recently published 1-1 and complex matching systems (see Column "Automatic Tools" of Table II) [30], [31], [32], [29].

Measuring Accuracy and Workloads: Following [31], [29], we compute the *accuracy* on each task as the fraction of top matches that are correct.

To examine how much our solution reduces the builder workload, we first develop a workload model for the builder. We develop this model based on observing how volunteer builders matched several schemas in Table II, on our own schema matching experience, and on current practice at enterprises [33].

First we describe the workload model for 1-1 schema matching. Suppose for attribute A, matching tool M has produced a ranked list of matches $m_1, m_2, \ldots, m_k$. We model the builder B's effort to find a correct match for A in three stages. First, B sequentially scans the ranked list $m_1, m_2, \ldots$ until either a plausible match has been found, or a pre-specified number p of matches has been examined.

Second, if a plausible match has *not* been found, B constructs one, using all available resources (e.g., the ranked list, the schemas, data instances, documents, etc.). In the third, and final, stage, given a plausible match (either found on the ranked list or constructed manually), B performs a quick "sanity check", by scanning the next q matches in the ranked list, to either convince himself or herself that there is no better match, or modify the current plausible match slightly, taking into account the scanned matches. Thus, the total workload of the builder B is

$$W_b = u * cost_1 + v * cost_2 + q * cost_3,$$

where u is the number of matches actually examined in Stage 1, and v is 1 if a plausible match must be constructed in Stage 2, and 0 otherwise. $cost_1$ and $cost_3$ are the workload incurred examining a single match in Stages 1 and 3 (respectively), and $cost_2$ is the cost of constructing a plausible match.

After measuring the actual time the volunteer builders spent on several 1-1 matching tasks in Table II, we set $p = 10$, $q = 5$, $cost_1 = 4$, $cost_2 = 8$, and $cost_3 = 1$. Intuitively, each sanity check imposes the smallest cost ($cost_3$), since B typically only glances at the proposed match. A "plausibility check", on the other hand, requires significantly more work ($cost_1$) as B must retrieve raw data, materialize a few instances of the match, and determine the correctness of the results. Constructing a plausible match is the most expensive operation ($cost_2$) as B must create the match, as well as verify its plausibility.

We model the *user workload* by assuming the cognitive demand of answering a question is equivalent to the smallest unit of work in the builder model, a single sanity check ($cost_3$).

Our treatment of workload for complex matching is similar, though tailored to fit the nature of that task (e.g., sanity checking a complex match is dependent upon the complexity of the match formula), and is omitted for space reasons. While our model is only approximate, we found that it correlates well with builder and user workloads as actually measured on tasks in Table II. Recent work has also proposed workload models

TABLE II: EXPERIMENTAL SETTINGS, IMPROVEMENTS IN ACCURACY, AND REDUCTIONS IN LABOR WHEN EMPLOYING OUR SOLUTION.

Task Types	Domains	Automatic Tools	Users	Tool Accuracy	Tool + Users Accuracy	User Workload	Builder Workload (before → after)	Builder Savings
1-1 schema matching	Book query interfaces	COMA	132 undergrad students, payment	0.63	0.97	12.5	781 → 277	65%
	Real estate I	LSD	3 volunteers	0.67	0.96	84.3	1062 → 490	54%
	Company listings	GLUE	6 volunteers	0.35	0.44	1473	17685 → 10529	40%
Complex matching	Real estate II	iMAP	5 volunteers	0.58	0.67	38	838 → 512	39%
	Inventory	iMAP	12 volunteers	0.33	0.89	52	1245 → 829	33%

[30], [34], but made the restrictive assumption that the builder only examines the top ranked match of each attribute.

Soliciting User Participation: To recruit users, for the first task (Book Query Interfaces), we employed a "payment" scheme (see Section VIII-D), and for the rest of tasks we asked for volunteers. Our user populations for the tasks range from 3 to 132, and consist of students at a large university.

A. Accuracy and Workload

We begin by examining how much we can improve accuracy over automatic tools. Columns "Tool Accuracy" and "Tool + Users Accuracy" of Table II show the accuracy of the tools alone and the tools with user help, respectively. The results show that our solution significantly improves accuracy across all five matching tasks, by 9-56%, to reach accuracy of 44-97%.

Next we examine the workload of the users, as well as the reduction in workload for the system builder due to the above accuracy improvement. Column "User Workload" shows the average effort that each user spent on the tasks. In terms of actual time, these efforts range from 5 to 15 minutes per task per user. The exception is Company Listing, where we allotted one hour to each user, and they all used up the hour.

Column "Builder Workload" shows the workload reduction for the builder. For example, the first row "781 → 277" states that the builder workload using only an automatic tool is 781, and that the workload using the tool plus users is 277, just 277/781 = 35% of the original workload. This is a savings of 65%, as shown in the "Builder Savings" column. The result shows that we can significantly reduce builder workload across all tasks, by 33-65%.

Thus, Table II suggests that a Web 2.0 style matching approach can significantly improve matching accuracy, at a minimal user workload, thereby significantly reducing the workload of the application builder. This result is achieved with a negligible overhead, which caused no noticeable delay in our experiments.

Total Workload: While our solution reduces the workload of the builder B (the key goal of this paper), it would be interesting to know how it affects the total workload. Let C be the workload of B in the traditional context (when no user is involved), and D be the workload of B plus that of all users in our solution context. Then we observed that in certain cases D is higher than C. Intuitively, this is due to the "built-in" redundancy that is necessary to handle noisy user answers.

However, in certain other cases, we observed that D is much lower than C (e.g., only 70-84% of C, for Real Estate I & II in Table II), resulting in an overall workload reduction. These cases occur when we ask questions for which the users can quickly converge on the correct answers, and then those answers greatly help the matching tool improve its accuracy.

B. User Performance

Question Answering: First, we consider the variability of user answers. For simple questions, such as "Is attribute date-posted of type DATE?", most users answered correctly and quickly, as expected.

For many questions, however, we found that user answers were "scattered", for three reasons. First, some users were simply "too quick" on some questions, and gave wrong answers. Second, for certain questions, the context provided (in the questions) was insufficient for some users to answer correctly. Third, certain questions are inherently subjective, with no objective answer. For example, can "Advertising" match "TV Broadcasting" in Company Listing? Some users said "yes" and some said "no". This third reason accounts for the vast majority of "scattered answers".

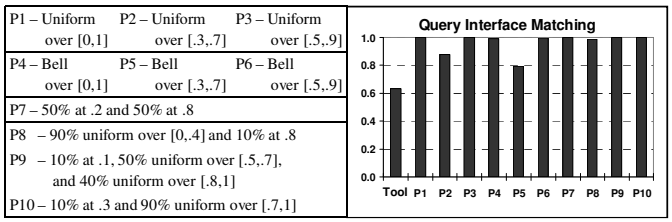
Fortunately, we found that even when a question is subjective, users still tend to converge on the *few plausible answers*. As a result, these answers usually "bubble" to the top of the returned ranked list, albeit in some unpredictable order. Since the correct results are still close to the top, we found that even when hammered by the subjectivity of questions, user answers still helped improve accuracy and helped the system builder quickly find the correct result.

"Payment" Schemes: We have employed helper applications such as course homepages as the "payment" schemes in some of our experiments (see Section VI). Our experience suggests that as long as the "payment" per day is kept to the minimum (e.g., 2-3 answers), users felt comfortable. Users also preferred answering the questions right at the start of using the helper application. This way, they were not interrupted during the day (analogous to paying for a service at the beginning or at the end).

C. Sensitivity Analysis

We also examine how our solution handles a broad range of user populations that we expect to commonly occur in practice, as well as populations of size up to 10,000. Since it is difficult to recruit such a large number of users, we consider simulation experiments.

Simulated User Populations: We generated ten synthetic user populations, as described in Figure 8.a. Each population has 500 users and each user has a single reliability score. A



(a) (b)

Fig. 8. Populations and results of simulation experiments.

reliability score of 0.6 means that on average the user answers 6 questions correctly out of 10. The ten user populations $P_1 - P_{10}$ have their reliability scores generated according to *uniform, bell, bimodal,* and *mixed* distributions. For example, population P_1 with "Uniform over [0,1]" means that user reliability scores are distributed uniformly over the interval [0,1]. The populations model "good", "average", and "bad" populations, based on the fraction of users with high reliability score (e.g., over 0.65) that they contain.

Accuracy and Workload: Figure 8.b shows the accuracy of our solution on the first task in Table I, i.e., Book Query Interfaces. (The results on other tasks are similar.) The first bar shows the accuracy of the automatic tool, duplicated from Table II, for comparison purposes. The next ten bars show the accuracy for user populations $P_1 - P_{10}$, respectively.

The results show that we can significantly improve automatic tools and reach the high accuracy of 98-100% for all populations except P_2 and P_5, where the accuracy is 79-89%. The populations P_2 and P_5 are the "worst" among the ten, in that they contain very few "good" users and that the highest reliability of these users is 0.7. Hence, the convergence criterion used for the experiments in Figure 8 is sufficient for the other eight "more reliable" populations but is inadequate for P_2 and P_5 (to achieve the maximum accuracy).

We also ran our system with different convergence criteria on four populations P_2, P_5, P_8, and P_{10}. The results show that applying a more conservative convergence criterion (which increases the number of evaluation questions, the gap size, and the stopping threshold) improves the accuracy on these populations to near perfect, at the cost of a slight increase of 1.8 in average user workload. The results thus demonstrate that we can achieve high accuracy across a broad range of populations. We also found that for these experiments the average user workload remains quite low at 1.8-6.5 per matching a query interface.

Effects of Population Size: Finally, we experimented with population sizes varying from 50 to 10,000 users, over many population types. The results show that matching accuracy remains stable and that the time it took to manage the collaboration framework was negligible. Furthermore, as expected, the workload required per user to complete the task decreases linearly as the population size increases. This suggests that our solution can scale up to large populations in all important performance aspects, and that in a large population the minimal workload per user will be significantly reduced further.

D. "Hands-Off" Systems

Finally, to explore the potential of our solution for building "hands-off", long-running integration systems where the builder workload is significantly reduced or removed, we attempted to build a simple data integration system on the Web, relying almost exclusively on user effort.

This system integrates online book stores. We created a simple mediated schema (having five attributes), found a large set of potential book store query interfaces from $invisibleweb.com$, then set up our collaborative solution. To recruit users, we used a "payment" scheme by leveraging the homepage of a database course. When a student reached the course homepage, he or she had to answer a simple question before gaining further access. However, each student did not have to answer more than 4 questions per day, and most of them ended up answering no more than 2. (Note that some of these questions were also used for evaluating users.) We used the answers to discover book store query interfaces, then match them with the mediated schema.

After 12 days, we have built a system over 10 book stores, with very little workload from each user (1.5 answers per day on average). Since the system builder did no work, apart from the initial setup as described earlier, we found the system slightly buggy (e.g., certain matches were not correct), but still very useful: one can query it via the mediated schema and obtain good results from the sources.

IX. Conclusion & Future Work

We have proposed a Web 2.0 style schema matching solution and demonstrated the potential of such solutions for integrating the data of online communities. Our work raises several interesting future directions that we plan to pursue:

Exploring More Sophisticated User Models: While our basic user model (Section V) has appeared to work well, we want to explore the tradeoffs as we move to more sophisticated user models. For example, we currently do not evaluate a user continuously, and hence cannot catch cases where a trusted user later becomes unreliable (e.g., start answering questions in a random fashion). Continuous user evaluation can minimize this problem, but would increase user workload. Is there a good balance? As another example, a Bayesian network based model can provide a principled way to exploit *all* users (not just trusted ones), but can significantly increase the amount of training data required (as discussed in Section V). Is there a way to address this problem? And under which conditions would such models become optimal?

Exploring Other Application Contexts: Can we apply such Web 2.0 style solutions to matching enterprise schemas? One may argue that they are unlikely to work well, because the cognitive bar would be just too high for layman users to contribute any useful information, by spending just a few minutes of their time. For example, if understanding say

pcost and ccost would require a careful reading of some documentation (e.g., to decide if they are before- or after-tax costs), then when faced with the question "does pcost match ccost?", a layman user is unlikely to be able to answer correctly.

This is true. However, layman users can already answer many "obvious" questions correctly, with little cognitive effort. For example, they can quickly flag "pcost is of type DATE" and "pcost = sim-id" as incorrect, even without knowing what these attributes mean. As another example, they can quickly supply coarse-grained domain integrity constraints, such as "bdate $<$ 2007". The important point to note is that, during the matching process, supplying even such seemingly "obvious" information can already significantly improve matching accuracy. This is because matches produced by today's matching systems are often highly *interrelated*, in that by correcting a match (or supplying an integrity constraint), that effect may "cascade" and affect many other seemingly unrelated matches [12]. This effect can be even more pronounced in complex matching scenarios.

Thus, it would be interesting to explore to what extent such Web 2.0 solutions would help enterprise schema matching. Another interesting problem in this direction is to explore applying such Web 2.0 solutions to best-effort, approximate data integration or dataspace systems [35], [36].

Exploring Other Problems: Finally, it would be interesting to explore if such solutions can work for other problems, both within data integration (e.g., source discovery, wrapper construction, and entity matching), and beyond that context (e.g., matching pictures with researchers, as we have done in DBLife, a prototype portal for the database research community [1]). We have conducted some preliminary work in this direction [28], and observed that for some of these problems it could take even less cognitive effort for users to answer questions (compared to the schema matching context). Examples include deciding if a Web form is indeed the query form for an online store (as opposed to, say, a subscription form) in source discovery, and deciding if two products match by comparing their pictures. Such problems can be especially amenable to Web 2.0, collaborative style solutions.

Acknowledgment: This work is partially supported by NSF CAREER grant IIS-0347903 and an Alfred P. Sloan fellowship to the third author. We thank the anonymous reviewers for many useful comments on an earlier version of this paper.

REFERENCES

[1] P. DeRose, W. Shen, F. Chen, A. Doan, and R. Ramakrishnan, "Building structured Web community portals: A top-down, compositional, and incremental approach," in *VLDB*, 2007.

[2] A. Doan, R. Ramakrishnan, F. Chen, P. DeRose, Y. Lee, R. McCann, M. Sayyadian, and W. Shen, "Community information management," *IEEE Data Eng. Bull.*, vol. 29, no. 1, pp. 64–72, 2006.

[3] Community Systems Group, "Community systems research at Yahoo!" *SIGMOD Record*, vol. 36, no. 3, pp. 47–54, 2007.

[4] [Online]. Available: http://research.microsoft.com/research/sv/CIM

[5] N. Bansal, F. Chiang, N. Koudas, and F. W. Tompa, "Seeking stable clusters in the blogosphere," in *VLDB*, 2007.

[6] E. Rahm and P. Bernstein, "On matching schemas automatically," *VLDB Journal*, vol. 10, no. 4, 2001.

[7] A. Doan and A. Y. Halevy, "Semantic integration research in the database community: A brief survey," *AI Magazine*, vol. 26, no. 1, pp. 83–94, 2005.

[8] Y. An, A. Borgida, R. J. Miller, and J. Mylopoulos, "A semantic approach to discovering schema mapping expressions," in *ICDE*, 2007.

[9] D. Aumueller, H. H. Do, S. Massmann, and E. Rahm, "Schema and ontology matching with COMA++," in *SIGMOD*, 2005.

[10] P. A. Bernstein, S. Melnik, and J. E. Churchill, "Incremental schema matching," in *VLDB*, 2006.

[11] L. Chiticariu and W. C. Tan, "Debugging schema mappings with routes," in *VLDB*, 2006.

[12] X. Chai, M. Sayyadian, and A. Doan, "Analyzing and revising database schemas to improve their matchability," University of Wisconsin, Madison, Tech. Rep., 2007.

[13] "http://web.media.mit.edu/ push/omcs-research.html."

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

[15] R. Ramakrishnan, "Mass collaboration and data mining," 2001, keynote address, KDD-01, www.cs.wisc.edu/ raghu/Kddrev.ppt.

[16] T. Joachims, "Optimizing search engines using clickthrough data," in *KDD*, 2002.

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

[18] S. Amer-Yahia, "A database solution to search 2.0," in *WebDB*, 2007.

[19] F. Wang, C. Rabsch, P. Kling, P. Liu, and P. John, "Web-based collaborative information integration for scientific research," in *ICDE*, 2007.

[20] R. Ramakrishnan, "Community systems: The world online," in *CIDR*, 2007.

[21] A. Doan, P. Bohannon, R. Ramakrishnan, X. Chai, P. DeRose, B. Gao, and W. Shen, "User-centric research challenges in community information management systems," *IEEE Data Eng. Bull.*, vol. 30, no. 2, pp. 32–40, 2007.

[22] "Sixth international workshop on information integration on the Web," 2007.

[23] S. Sarawagi and A. Bhamidipaty, "Interactive deduplication using active learning," in *KDD*, 2002.

[24] K.-U. Sattler, S. Conrad, and G. Saake, "Interactive example-driven integration and reconciliation for accessing database federations," *Inf. Syst.*, vol. 28, no. 5, pp. 393–414, 2003.

[25] L. Yan, R. Miller, L. Haas, and R. Fagin, "Data Driven Understanding and Refinement of Schema Mappings," in *SIGMOD*, 2001.

[26] R. McCann, A. Doan, A. Kramnik, and V. Varadarajan, "Building data integration systems via mass collaboration," in *WebDB*, 2003.

[27] R. McCann, A. Kramnik, W. Shen, V. Varadarajan, O. Sobulo, and A. Doan, "Integrating data from disparate sources: A mass collaboration approach," in *ICDE*, 2005.

[28] R. McCann, "Efficient data integration: Automation, collaboration, and relaxation," Ph.D. dissertation, University of Illinois, Urbana-Champaign, 2007. [Online]. Available: http://dblife.cs.wisc.edu/people/mccann/papers/phd_thesis.pdf

[29] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos, "iMAP: Discovering complex mappings between database schemas," in *SIGMOD*, 2004.

[30] H. Do and E. Rahm, "COMA: A system for flexible combination of schema matching approaches," in *VLDB*, 2002.

[31] A. Doan, P. Domingos, and A. Halevy, "Reconciling Schemas of Disparate Data Sources: A Machine Learning Approach," in *SIGMOD*, 2001.

[32] A. Doan, J. Madhavan, P. Domingos, and A. Halevy, "Learning to map ontologies on the Semantic Web," in *WWW*, 2002.

[33] P. Bernstein, Personal Communication, 2004.

[34] S. Melnik, H. Molina-Garcia, and E. Rahm, "Similarity Flooding: A Versatile Graph Matching Algorithm," in *ICDE*, 2002.

[35] J. Madhavan, S. Cohen, X. L. Dong, A. Y. Halevy, S. R. Jeffery, D. Ko, and C. Yu, "Web-scale data integration: You can afford to pay as you go," in *CIDR*, 2007.

[36] A. Y. Halevy, M. J. Franklin, and D. Maier, "Principles of dataspace systems," in *PODS*, 2006.