Quality Control for Crowdsourcing with Spatial and Temporal Distribution

Gang Zhang and Haopeng Chen

REINS Group, School of Software
Shanghai Jiao Tong University
Shanghai, P.R. China
{infear,chen-hp}@ sjtu.edu.cn

Abstract. In the past decade, crowdsourcing has become a prospective paradigm for commercial purposes, for it brings a lot of benefits such as low cost and high immediacy, particularly in location-based services (LBS). On the other side, there also exist many problems need to be solved in crowdsourcing. For example, the quality control for crowdsourcing systems has been identified as a significant challenge, which includes how to handle massive data more efficiently, how to discriminate poor quality content in workers' submissions and so on. In this paper, we put forward an approach to control the crowdsourcing quality from spatial and temporal distribution. Our experiments have demonstrated the effectiveness and efficiency of the approach.

Keywords: crowdsourcing, location-based service (LBS), quality control, spatial and temporal distribution.

1 Introduction

The proposal of "Crowdsourcing" paradigm extends back to 2006. It was defined as "a company or organization outsources their tasks to those who are not specific in the form of free voluntary (and usually large public networks)"[1]. As we can see, "crowdsourcing" actually originates from the transformation of innovation mode of enterprises. Nowadays, with the popularity of Internet, It is becoming a trend that consumers generate contents by their enthusiasm on innovation. For instance, BMW [2] found its innovation laboratory in Germany to provide users with online tools which can help them participate in the production design, which would not only promote innovation, but also increase the popularity of enterprise.

In the age of the Internet, the concept of crowdsourcing, that virtually anyone has the potential to plug in valuable information is extended to wiki and other collaboration tools [3]. For example, Amazon's Mechanical Turk [4], one of the most successful commercial crowdsourcing platforms, offers businesses and developers access to an on-demand, scalable work force. Essentially, potential employers post tasks and workers select jobs they would like to perform.

Recently this paradigm has also flourished in location based services (LBS), in which the smart-device users contribute information about their surroundings, thereby

M. Pathan, G. Wei, and G. Fortino (Eds.): IDCS 2013, LNCS 8223, pp. 169–182, 2013.

providing a collective knowledge about the physical world [5]. LBS are typical application scenario of crowdsourcing, since it always need large amounts of resource to collect the information about locations. This would cost a lot without using crowdsourcing. It also requires high immediacy, since the physical world is always changing. So it still faces the challenge of quality control. In some cases, the services rely on mapping software such as Google Maps. For example, CROWDSAFE [6], a novel convergence of Internet crowdsourcing and portable smart devices to enable real time, location based crime incident searching and reporting. In addition, there are also many other indoor LBS based on the Access Points (AP). But no matter which kind of LBS, the quality of service completely depends on the crowdsourcing quality. In this paper, we regard indoor LBS as our research background and discuss about the quality control in crowdsourcing.

In our system, we hope to locate all APs accurately in an area in order to provide people with value-added services later. As long as one's smart-device gets connected to an AP, he could submit the information about it, such as its identification and its possible position to the server. Then, these submitted contents would be aggregated to accurately locate AP on the server side. But since the worker's smart-device may not be precise or there exist workers' cheating behaviors, the aggregation quality would be poor. So here we try to solve such three problems: 1.How to eliminate useless contents that workers submit in order to handle massive data more efficiently. 2. How to aggregate these contents and generate results more accurately. 3. How to evaluate workers' single contribution in each task and overall performance periodically.

The paper is structured as follows. Section 2 gives a quick overview of the concept of crowdsourcing and the research already done in the area. In Section 3, we present our main method on how to solve the questions mentioned above. Detailed implementation would be involved in Section 4 and experience results would be analyzed in Section 5. In Section 6, we would do conclusion and some discussion about our future work.

2 Related Work

In crowdsourcing paradigm, there are two roles, employer and worker as shown in Fig.3. People called employer submit tasks, evaluate worker's submitted results and pay workers, while workers pull and complete tasks, get payment from employers.[7]

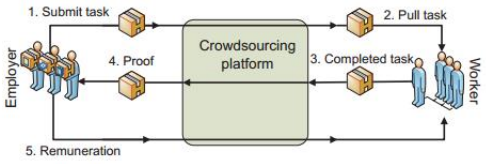

Fig. 1. Crowdsourcing scheme

Generally, crowdsourcing task is simple but needs large amounts of resource to be accomplished. Thanks to global growth in Internet connectivity and bandwidth, we can now harness human resource in near-real time from a vast and ever-growing, distributed population of online Internet users [8]. In this way, crowdsourcing brings low cost and high efficiency. But if exists cheating in workers' accomplishments, the quality of tasks would be influenced. Meanwhile, it costs a lot that validating whether a worker is cheating in the task. For example, on EBay website, everyone could maliciously add good or bad comments on products which would affect their reputation later. So it is essential to have a mechanism to judge workers' overall performance during a period and single contribution in each crowdsourcing task.

Some researches focus on judging whether the contents which workers submit should be accepted or rejected. They consider that inaccurate acceptance or rejection would affect not only current task, but also possibly drive a new wave of fraudsters, because those who have cheated do not receive any punishment. For example, Matthias et al. raise "Majority Decision Approach" [7] to judge whether worker's submission is correct in simple tasks, and using "Control Group Approach" method in complicated cases. Besides, Petros and Hector propose "Gold Standard Performance" to detect one worker's performance before the crowdsourcing task starts [9]. Many other researchers think that workers' characteristics such as demographics or personality traits are related to the quality of their work under specific task conditions [10]. In [11], Winter Mason et al. put forward a data-driven model for quality control in the context of crowdsourcing systems, which aims to assess the quality of individual contribution for parallel distributed tasks. There are also many other discussion on quality control for crowdsourcing in certain fields such as geographic [12] and real-time applications [13] . In addition, it is also a hot topic that to balance the task quality and reward cost [14]. For instance, in 2009, Yahoo's research institute made a quantitative analysis on the relationship between "Financial Incentives" and "Performance of Crowds" [15], and found that higher reward can accelerate the accomplishment of the task, but cannot improve its quality.

In this paper, we discuss about the task quality from a new aspect which is called analysis with spatial and temporal distribution. It has to be noted that the value of contents submitted by workers would attenuate from two prospective: time and space. From the point of space, for example, if one has submitted a piece of record *r*, his subsequent submissions during a short period, which contain the same contents about the position as the previous one, are less meaningful. It is because that AP's position is fixed during a short period in reality. On the other hand, the submission of one worker's history tasks is not important as that of up-to-date ones, which is a kind of attenuation on value with time. It is assumed one worker's late submission is more persuasive than before, for AP's position may be changed during a long period.

3 Method

As shown in Fig.2, there are three phases in our system design: *Filtering*, *Aggregating*, and *Feedback*. Since the scale of contents submitted may be too large like several TBs every day, we would use some parallel-computing model such as Mapreduce [16] here to improve efficiency of data processing.

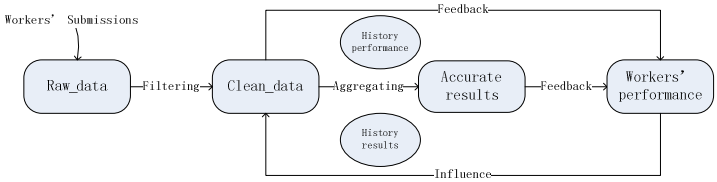

Fig. 2. System design

3.1 Background

In our design, each record submitted by workers contains a key-value pair which is made up of an AP's id (*apid*) and its position (*pos*). The record would also include other information about the worker such as his id (*workerid*) and the time he submits the record (*timestamp*). We use a 16-bit unsigned integer to denote these elements respectively except *timestamp*. In particular, the position is made up as follows:

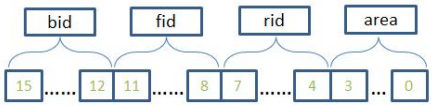

Fig. 3. 16-bit integer denote each AP's position

The position is composed of a building (*bid*), a floor (*fid*), a room (*rid*) and the location in the room (*area*). As shown in Fig.3, there are max 2^16 different positions and each room is partitioned to 2^4 grids. For the sake of simplification, we define the distance *d* between two positions pos_1 and pos_2 as follows:

$$d_{pos_1,pos_2} = \left|pos_1 - pos_2\right| \tag{1}$$

It is assumed that two positions are close as their values are numerically close. For example, the room of rid 1 and that of rid 2 are supposed to be close geographically. So these two positions may be covered by the same signal from one AP. Meanwhile, if two positions are located in different buildings, the distance between them would be very large (at least 2^12).It is almost impossible that they could access to the same AP.

The records set submitted by workers are called "raw data", since it is highly possible that some records among the set are incorrect and meaningless. So it is necessary to filter out these records to obtain the clean data. After that, we try to obtain the accurate results of AP's position from the clean data. It is realistic that the position of AP is comparatively fixed during a short period. Therefore, we regard one day as the minimum unit to do aggregation. However, each worker's performance or credibility is different, so the records in their submission should also have different quality. Here, we define *ar* to describe each worker's overall performance which is summarized from his history task. Last but not least, it is necessary to update each

worker's performance according to the latest aggregated result (still regard one day as the minimum unit). It is a kind of feedback on *ar*, which would come into effect in one's next task.

3.2 Spatial Distribution

Since our research is based on LBS, it is natural to utilize spatial distribution to evaluate workers' submission. It was involved in the *Filtering* and *Feedback* phase.

In Filtering phase, we want to find out those meaningless records. But which records are meaningless? It is supposed that AP's position is fix during a short period. So the frequent submissions which contain the same contents are meaningless. It is highly possible since workers would like to get more rewards by more submissions. This is thought of as a kind of cheating like that in "Page Rank" [17]. Therefore, only the records earliest submitted are considered meaningful and the subsequent ones are less valuable. We introduce r_v *to* describe such value of each record(r) and p_c to denote the short period. For example, if one worker first submits a record which contains the pair $\{apid_1, pos_1\}$, the record would own a value of 1(r_v =1). But the records of the same pair in his subsequent submission during p_c would have value smaller than 1 and get lower and lower. A shrink factor on value f is defined to demonstrate the speed of such attenuation on value. The detail is shown in Fig.4. It is supposed that p_c lasts 1 minute here and the shrink factor f_1 is set 0.5:

Workerid	Records=(apid, position)	Timestamp	Value(r_v)
10	(1, 32)	12:00:00	1
20	(1, 32)	12:00:00	1
20	(1, 32)	12:00:10	0.5
20	(1, 32)	12:00:40	0.25
20	(1, 32)	12:00:55	0.125

Fig. 4. Records value table with f_1=0.5 and p_c=1 min

On the other hand, we also use spatial distribution to evaluate the accuracy of workers' submissions. It is natural to compare one's submissions and the accurate results to get how many records match. We introduce *hit* to denote the worker's contribution in each task, which can be calculated as follows:

$$hit=mrw/trw \tag{2}$$

Where *mrw* denotes the count of one's matched records, *trw* is the count of all one's records in his submissions. It has to be noted that even if one worker doesn't have any cheating deliberately in the task, it is also highly possible that his submission is not accurate due to device problems or wireless interference. Therefore, if the record in one's submission is different from the aggregated result, which means they contain different positions of the same AP, it would still be retained instead of discarded while the worker's *hit* is calculated. But the weight of this record, which was denoted by w in our definition, is related to the distance between these two positions. As pos_1 and pos_2 denote the positions, the weight w of the record would be:

$$w = \begin{cases} e^{-d_{pos_1,pos_2}}, & d_{pos_1,pos_2} \leq \mathrm{R} \\ 0, & d_{pos_1,pos_2} > R \end{cases} \tag{3}$$

If pos_1=pos_2, it is considered to be an accurate match. So the weight of the matched record is 1.Otherwise, the weight would get smaller as the position in the record gets farther from the accurate one. When the distance is large than the threshold R in our definition, the record is supposed to have no weight. Suppose that r_i denotes the i_{th} record, the total weights of one's records denotes his *mrw* each day, and *trw* is just the number n of all his records.

$$mrw = \sum_{i=1}^{n} w_{r_i} \tag{4}$$

$$trw = n \tag{5}$$

3.3 Temporal Distribution

It is obvious that worker's history performance could determine his later performance to some extent. Here, we define *ar* to describe each worker's overall performance which initial value is 1. It is also used to judge the credibility (r_c) of the records in one's later submissions. Two arguments s_1 and s_2 are introduced to denote the *ar* thresholds. For example, if one's *ar* is more than s_1, all records he submits in next task would be regarded as credibility of 1(r_c=1) and accepted completely. Because he is supposed to be an honest worker without any cheating before. On the other side, if one's *ar* is lower than s_2, his submissions may also have poor credibility and thus, discarded. Otherwise, one's submissions would be accepted partly and his records would have credibility of his updated *ar* (r_c=*ar*). Meanwhile, we generate other four records whose contents are close semantically to the origin one, for these records are also trustable to some extent. But their credibility is lower than the original one. The details are shown in Fig.5.

	Workerid	Ar	Records	Credibility(r_c)
Completely accepted	10	1	(1, 32)	1
Partly accepted	20	0.6	(1, 32)	0.6
Generated records	20	0.6	(1, 33)	0.1
	20	0.6	(1, 34)	0.1
	20	0.6	(1, 31)	0.1
	20	0.6	(1, 30)	0.1
Discarded	30	0.1	(1, 100)	0

Fig. 5. Worker's *ar* and his records credibility with s_1=0.8 and s_2 =0.2

Then, we use one's history *ar* and his recent contributions *c* to obtain his up-to-date performance *ar'*. It is calculated as follows:

$$ar' = (1 - m_1) * ar + m_1 * c \quad 0 < m_1 < 1 \tag{6}$$

An impact factor m_1 is still defined here, which denotes the weight of the worker's latest contribution while update his *ar*. It has to be noted that the weight of one's initial performance would decrease exponentially and close to zero at last. The later one's contribution is the more weight it would have.

Similarly, although the position of AP is fixed during a short period, it is still of high possibility that AP would be located in any other places during a long period. It is natural to use both the history result and the latest result to obtain the up-to-date location of AP. The details would be involved in the Section 4.

4 Implementation

In this Section, the details, particularly the algorithms, about our system implementation are involved. According to our design, the section would be divided into three parts.

4.1 Filtering with Quality Table

In crowdsourcing, it is essential to discriminate the quality of workers' submissions. Here, we define the *quality* (r_q) of each record (r) in our system to describe how the record is trustable, which is affected by the *credibility* (r_c) and the *value* (r_v) of the record mentioned in Section 3. For the sake of simplification, the product of *credibility* and *value* are used to denote the *quality*:

$$r_q = r_c * r_v \tag{7}$$

For example, if one's *ar* is 0.6, which means that each records except our generated ones in his submissions would have a credibility of 0.6(r_c=0.6). Meanwhile, the value of his one record r is 1, then this record r would have a quality of 0.6*1=0.6(r_q=0.6). So we have the Quality table [18] which is extended from Fig.4 and Fig.5 as follows:

Workerid	Ar	Records	Timestamp	Credibility(r_c)	Value(r_v)	Quality(r_q)
10	1	(1, 32)	12:00:00	1	1	1
10	1	(1, 32)	12:00:10	1	0.5	0.5
10	1	(1, 32)	12:00:20	1	0.25	0.25
20	0.6	(1, 32)	12:00:00	0.6	1	0.6
20	0.6	(1, 33)	12:00:00	0.1	1	0.1
20	0.6	(1, 34)	12:00:00	0.1	1	0.1
20	0.6	(1, 31)	12:00:00	0.1	1	0.1
20	0.6	(1, 30)	12:00:00	0.1	1	0.1
30	0.1	(1, 100)	12:00:00	0	1	0

Fig. 6. Records Quality Table with s_1=0.8, s_2 =0.2, f_1=0.5 and p_c=1 min.

As shown in Fig.6, the records submitted by the worker 10 whose *ar* is 1 are always of high credibility (r_c = 1), but of low value if he try to cheat by repeated submission (r_v = 0.5 or 0.25).So his third record of (1, 32) only has a quality of 0.25. On the other hand, since the *ar* of worker 20 is between s_1 and s_2, we generated other four records of low credibility (r_c =0.1). Otherwise, the worker's submission would be discarded (r_c=0 and r_q=0) as his low *ar.*

4.2 Aggregating with Mapreduce Model

Majority Decision Approach (MDA)

Majority Decision Approach could be explained through a simple case. For example, for AP of id 1, there exist 100 records (including the generated ones) that show the position of it is pos_1 and another 60 records that tell pos_2.Then, the first result is

obviously more trustable than the other, since it has more supporters. So pos_1 is considered the position of this AP.

Here, we use the sum of quality instead of the count of each record. For instance, in Fig.6, the total quality of the record (1, 32) would be 1+0.5+0.25+0.6=2.35, while that of the record (1, 34) is only 0.1. Therefore, the AP of id 1 is more likely located at position 32.

As mentioned in the design part, one day is regarded as the minimum unit to do aggregation. It is assumed that there are n workers who involved in the task during the period. Besides, we have the following arguments as input:

- S_r: clean data set derived from workers' submitted records. The format of each record is (*apid, pos, workerid, timestamp).*
- A= {$ar_1 \ldots ar_n$}: the latest *ar* of each worker.
- S_1, S_2: two *ar* thresholds used to judge how worker's submission would be accepted.

Since the records set may be too large to handle, we use HADOOP [19] framework here to improve performance of data processing. So our algorithm can be divided into mapper part and reducer part. The Mapper procedure is as follows:

```
Algorithm1: MDA Mapper Procedure
Input: Records Set S_r, Accept Rate (ar) Set A = {ar_1…ar_n},
S_1 and S_2 act as ar thresholds
Output: Record Set S'. Each record owns an extra element
quality describes the accuracy of this record.
For each (apid, pos, workerid, timestamp) in S:
       ar = A [workerid]
       If ar > S_1
// completely accepted, quality=1.
// output format:key= apid:pos, value=quality.
              Output (apid:pos, 1)
       Else if ar > S_2:
//partly accepted, quality=ar.
              Output (apid:pos, ar)
               _quality = (1 - ar) /4
//generate records contain positions around.
              p^1, p^2, p^3, p^4=gen (pos)
              Output (apid: p^1, _quality)
              Output (apid: p^2, _quality)
              Output (apid: p^3, _quality)
              Output (apid: p^4, _quality)
       Else: // omitted
              Continue next loop.
```

Algorithm1: MDA Mapper Procedure

If one's submission is accepted partly, we would generate other four records whose contents are close to the origin one. The implementation of the method *gen ()* is omitted here, since it is not our focus and may involve some knowledge of geography. It has also to be noted that our approach is not limited to the LBS scenario, as long as you generate the data which is semantically close in your scenario context.

Now we do reduce job (Fig.8). Since each record in *S'* contains an *apid*, position information *pos* and a quality r_q, it is natural to count the quality each key (*apid:pos*) appears first and group them by *apid* later. Then, we sort them by the quality each one appears in each group and the max one is our wanted result, for most workers think this position is the accurate one where the AP is located. For example, the total quality of record (1, 100) is 80.0, while the record (1, 200) has a quality of 100.0. Then we consider the second one more trustable and output the result like (1:200,100.0), which means position 200 is a more accurate position.

```
Algorithm2: MDA Reducer Procedure
Input: Set S' consists of the records whose format is
(apid: pos, quality).
Output: Each AP's id, its accurate position and the total
quality.
input = {} //key= (apid:pos), value=total quality.
result_table = {}   // key= apid, value=pos.
quality_table={}   // key= apid, value=quality
For each (apid:pos, quality) in S':
       k, v = (apid:pos),  quality
       If k in the keyset of input:
              Update its value by adding v
       Else:
              Insert (k, v) into input
//find the max quality of each in each AP group.
For each (apid:pos, quality) in input:
       If apid in the keyset of result_table:
              k',v'= apid,quality_table[k']
              If quality>v':
//find more accurate position.
                     Update by set:
                            result_table[k] =pos,
                            quality_table[k]=quality
       Else:
              Insert (apid, pos) into result_table
              Insert (apid, quality) into quality_table
//output results.
For each apid in the keyset of result_table:
       Output apid,
result_table[apid],quality_table[apid]
```

Algorithm2: MDA Reducer Procedure

Aggregating with Temporal Distribution

The position got above is only temporary result, since we need to combine it with the history result as mentioned in Section 3. While both of them are useful in determining the up-to-date position of AP, it is natural that the former would have more weights.

The weight of the history result could be divided into two parts. The first part is its total quality. For instance, if the history position of one AP has a quality of 100.0, which means there exists at least 100 supporter before (because each record has at most 1.0 quality), then it is obviously more trustable than the temporary result of quality 1.0.So,the quality of result plays a role in determining the latest position of AP.

On the other hand, it had s to be noted that the quality would attenuate over time. Suppose that we got the history result of one AP with a timestamp one month ago, the quality of it has decreased dramatically. So it is not accurate as before and may be replaced by the temporary result.

We define f_2 to denote the speed of the attenuation in the quality of history results one day. Given that pos_{apid_h} is the history result of the AP of *apid* and its quality is q_{apid_h}. Meanwhile, we got the temporary result pos_{apid_t} and its quality q_{apid_t} from the last step. The up-to-date result pos_{apid} is:

$$pos_{apid} = \begin{cases} pos_{apid\_h},\ q_{apid_h} * f_2 > q_{apid\_t} \\ pos_{apid\_t},\ q_{apid_h} * f_2 \leq q_{apid\_t} \end{cases} \tag{8}$$

It has to be noted that the quality of up-to-date result is always higher than or equal the quality of history result which has shrunken. Therefore, our aggregated position of each AP would be more and more accurate.

4.3 Feedback: Sliding Window Analysis

Sliding Window Analysis is a flexible and accurate approach which is used to evaluate each worker's performance periodically. Here, we still regard one day as the minimum unit to calculate each one's hit and also update his *ar* every day. In addition, we introduce the following definitions:

- $SW_i,\ i \in N$: A sliding window denotes the i_{th} dynamic period $[t_{i1},\ t_{i2}]$. Here the length of ***SW*** is p days for example and we mark each day as an integer from 1. So first SW is *[1, p]* and it would slide to right by increasing both t_{i1} and t_{i2} by 1. For instance, the third SW would be *[3, p+2]* and the n_{th} SW is *[n, n+p-1]*.
- $H_i,\ i \in N$:The set contains one's contribution every day during the current SW_i period. Each contribution is a tuple *(mrw, trw, hit)*
- $ar_i,\ i \in N$: One worker's *ar* on the i_{th} day.
- m_1: an impact factor, which denotes the weight of the worker's latest contribution while update his *ar*

For instance, if given period one week and we want to calculate one's ar_{10} on the 10_{th} day, the 4_{th} sliding window, which denotes the period [4, 10], and H_4 would be used. That is to say, when we want to calculate ar_i, the SW_{i-p+1} would be *[i-p+1, i]* and all the tuples in H_{i-p+1} are used. The details could be seen in Fig.7.

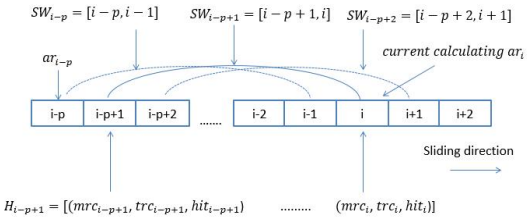

Fig. 7. Sliding Window Analysis

So in the formula (6), worker's recent contribution *c* would be calculated as:

$$c = \frac{\sum_{i-p+1}^{i} mrw_j}{\sum_{i-p+1}^{i} trw_j} \tag{9}$$

And *ar* would be:

$$ar_i = (1 - m_1) * ar_{i-p} + m_1 * \frac{\sum_{i-p+1}^{i} mrw_j}{\sum_{i-p+1}^{i} trw_j} \tag{10}$$

It has to be noted that we take the worker's history performance into account by giving ar_{i-p} a certain weight, regarding that it would influence one's later performance (ar_i) to some extent. Second, instead of the average of one's hits every day, we make use of the weighted average. It is because that the former may not reflect one's contribution accurately. For example, if one submits 10 records and 9 of them match aggregated results, the hit would be 90%. Meanwhile, he submits only 1 record which is also correct in another day, the hit is 100%. But obviously the first hit is more persuasive than the latter, since it contains more records. Last but not least, we also do periodically analysis which would not suffer from the case in which one's contributions variy dramatically from day to day.

With Sliding Window Analysis, one's history performance and his contributions in recent tasks are combined together, which is more persuasive and thoughtful. Meanwhile, it is also not sensitive to one's single contribution each day. But the cost on time is also higher, since each worker's *ar* need to be updated every day.

5 Experiments and Evaluation

To evaluate the effectiveness of our approach, we generate massive data set to simulate the submitted records under crowdsourcing model. The data set has the characteristics of that in crowdsourcing in reality. For example, while most workers are honest and always submit accurate contents, there exist a group of cheaters who may submit wrong contents. In addition, the data set has a kind of self-learning ability. For instance, if a cheater doesn't get his *ar* reduced, he would do more and more cheating in subsequent tasks and even drive other honesty workers to cheat, since he doesn't receive any punishment. We would see the improvements as using our approaches. Time cost and the accuracy of aggregated results are our focus.

Given 1000 workers and 1500 Access Points (AP), we generate data of 1GB size to simulate the records that submitted by workers in crowdsourcing every day during a period of 30 days. The other arguments are set as follows:

- Thresholds: S_1=0.8, S_2=0.3, R=2^8=256.
- Impact factor: m_1=0.3.
- Shrink factors: f_1=f_2=0.5.
- Sliding Window of 5 days length.
- Each worker's initial ar is 1 or 100%.

We pick up a piece of period (5 days) of the results in our experiments to be shown. There are four cases. (1) C_1: without filtering and quality table. (2) C_2: only using filtering procedure. (3) C_3: only aggregate with quality table. (4) C_4: using both filtering procedure and quality table to aggregate.

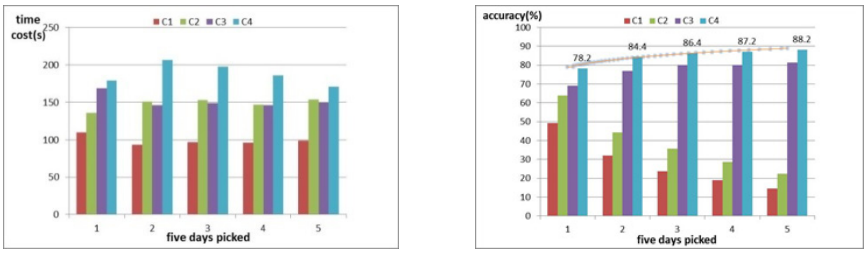

Fig. 8. Time cost and accuracy in different cases

As shown in the Fig.8, without our approach, the accuracy of the aggregated result is lower than 30%, which is an unacceptable value in practical applications. In addition, as time goes on, the accuracy is going down continuously (include the case of C_2), since more and more workers submit records of low quality.

On the other side, although the time cost of aggregating that utilize filtering and quality table is higher (2~3 times more than C_1), its accuracy is very satisfied (more than 80% at last). Since each worker's performance tends to be stable, it would be more and more accuracy to validate one's submission quality. Besides, aggregating with quality table would bring more continuous improvements than filtering.

In conclusion, although time cost is a little higher, our approach, particularly the aggregating procedure with quality table, performs well on quality control of the workers' submission.

6 Conclusion and Future Work

In crowdsourcing, the contents submitted by workers are always large and inaccurate. So as an employer, it is essential to control the quality of the contents. Here, we regard our indoor AP located system as research background and discuss how to qualify these data efficiently and accurately.

First, the meaningless contents in the raw data are eliminated with spatial distribution analysis, since they would only cause interference in our subsequent procedure. After that, we obtain the clean data. Then, we define *ar* to describe one's overall performance and it also determines the probability that one worker's submission would be accepted. There exist three cases, completely or partly accepted and discarded. Last but not least, we utilize HADOOP framework to aggregate the records to get the accurate position of each AP with Majority Decision Approach.

Moreover, the approach Sliding Window Analysis is introduced to update workers' *ar*. It utilizes time distribution analysis to describe one's performance accurately and dynamically. But it would cost high on time, since it need update one's *ar* every day.

It has to be noted that our approach can be applied to many other applications. Our core idea is to utilize spatial and temporal distribution to analyze the submission quality, which is irrelevant to the semantics of the contents in the submission. The detection of malicious grading on E-commerce sites is another suitable scenario. For example, it is treated as malicious behavior that one's frequent commits of the same grading during a short period. Meanwhile, we can still bind one's credibility with the quality of his grading together. If necessary, we could also generate the possible grades which are close to the origin one.

Our next work will focus on further experiments on our idea. For example, the length of the Sliding Window is the point. Although short sliding window could evaluate workers' performance accurate, it also brings high time cost. So it is essential to find a tradeoff between accuracy and cost. On the other hand, data refreshing in crowdsourcing is also a problem need to be solved. As time goes on, the scale of the data would be too massive to retain. So it is necessary to have a mechanism to eliminate those data with low value or freshness.

References

1. Howe, J.: The Rise of Crowdsourcing, Wired (June 2006), http://www.wired.com/wired/archive/14.06/crowds.html
2. BMW innovation lab, https://www.bmwgroup-cocreationlab.com/cocreation/project/customer-innovation-lab
3. Greengard, S.: Following the crowd. Communications of the ACM 54(2), 20–22 (2011)
4. Amazon Mechanical Turk, http://www.mturk.com
5. Alt, F., Sahami, A., Schmidt, S.A., Kramer, U., Nawaz, Z.: Location-based crowdsourcing: extending crowdsourcing to the real world. In: 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries, pp. 13–22
6. Shah, S., Bao, F., Lu, C.-T., Chen, I.-R.: CROWDSAFE: crowdsourcing of crime incidents and safe routing on mobile devices. In: 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 521–524
7. Hirth, M., Hoβfeld, T., Tran-Gia, P.: Cost-Optimal Validation Mechanisms and Cheat-Detection for Crowdsourcing Platforms. In: 5th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, pp. 316–321
8. Lease, M., Yilmaz, E.: Crowdsourcing for information retrieval. Newsletter ACM SIGIR Forum Archive 45(2), 66–75 (2011)
9. Venetic, P., Garcia-Molina, H.: Quality control for comparison microtasks. In: The 1st International Workshop on Crowdsourcing and Data Mining, pp. 15–21

10. Kazai, G., Kamps, J., Milic-Frayling, N.: The face of quality in crowdsourcing relevance labels: demographics, personality and labeling accuracy. In: 21st ACM International Conference on Information and Knowledge Management, pp. 2583–2586
11. Zhu, S., Kane, S., Feng, J., Sears, A.: A Crowdsourcing Quality Control Model for Tasks Distributed in Parallel. In: CHI 2012 Extended Abstracts on Human Factors in Computing Systems, pp. 2501–2506 (2012)
12. Andrew, J., Flanagin, M.J.: Metzger. The credibility of volunteered geographic information. Geo Journal, An International Journal on Geography, Published Online (July 24, 2008)
13. Mashhadi, A.J., Capra, L.: Quality control for real-time ubiquitous crowdsourcing. In: 2nd International Workshop on Ubiquitous Crowd Souring, pp. 5–8
14. Kamar, E., Horvitz, E.: Incentives for truthful reporting in crowdsourcing. In: 11th International Conference on Autonomous Agents and Multiagent Systems, vol. 3, pp. 1329–1330
15. Mason, W., Watts, D.J.: Financial incentives and the "performance of crowds". ACM SIGKDD Explorations Newsletter 11(2), 100–108 (2009)
16. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Communications of the ACM - 50th Anniversary Issue: 1958 - 2008 51(1), 107–113 (2008)
17. Chen, Z., Ma, J., Cui, C., Rui, H., Huang, S.: Web page publication time detection and its application for page rank. In: 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 859–860
18. Cheng, R., Chen, J., Xie, X.: Cleaning uncertain data with quality guarantees. Journal VLDB Endowment 1(1), 722–735 (2008)
19. Hadoop, `http://hadoop.apache.org/`