Quality Control of Massive Data for Crowdsourcing in Location-Based Services

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. Crowdsourcing has become a prospective paradigm for commercial purposes in the past decade, since it is based on a simple but powerful concept that virtually anyone has the potential to plug in valuable information, which brings a lot of benefits such as low cost and high immediacy, particularly in some 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' submission and so on. In this paper, we put forward an approach to control the crowdsourcing quality by evaluating workers' performance according to their submitted contents. Our experiments have demonstrated the effectiveness and efficiency of the approach.

Keywords: crowdsourcing, massive data, quality control, location-based services (LBS).

1 Introduction

The proposal of "Crowdsourcing" paradigm was first 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]. In today's Web 2.0 world, the concept of crowdsourcing, that virtually anyone has the potential to plug in valuable information is extended to wiki and other collaboration tools [2].Recently this paradigm has also flourished in location-based service (LBS) [3], in which the smart-device users contribute information about their surroundings, thereby providing a collective knowledge about the physical world. In some cases, these services rely on mapping software such as Google Maps. For example, CROWDSAFE [4], 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), which is called "wifi-type" service. But no matter which kind of LBS， the quality of service completely depends on the crowdsourcing quality. In this paper, indoor LBS is our research background.

J. Kołodziej et al. (Eds.): ICA3PP 2013, Part II, LNCS 8286, pp. 112–121, 2013.

The paper is structured as follows.Sec.2 gives a quick overview of the concept of crowdsourcing and the research already done in the area. In Sec.3, we present our main design about how to solve the questions mentioned above. Detailed implementation would be involved in Sec.4 and experience results would be analyzed in Sec.5. In Sec.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 is shown in Fig.1. People called employer submit tasks, evaluate worker's submitted results and pay workers, while workers pull and complete tasks, get pays from employers.[5]

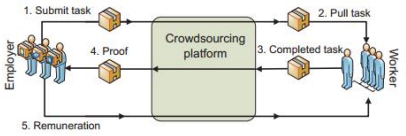

Fig. 1. Crowdsourcing scheme

Generally, crowdsourcing task is simple but needs large amount of resource. But now we can now harness human resource in near-real time from a vast and ever-growing, distributed population of online Internet users [6]. In this way, crowdsourcing brings low cost and high efficiency. But if exists cheating, the quality of tasks would be influenced. Meanwhile, it cost a lot that validating whether a worker is cheating in the task.

Some researches 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" [5] 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 [7]. In addition, it is also a hot topic that to balance the task quality and rewards cost. For instance, in 2009, Yahoo's research institute made a quantitative analysis on the relationship between "Financial Incentives" and "Performance of Crowds" [8], and found that higher rewards can accelerate the accomplishment of the task, but cannot improve its quality. In this paper, we would discuss about the task quality from a new aspect.

3 System Design

The goal of our system is to locate all Access Points (AP) accurately in an area, thus providing value-added services later. While one's smart-device is access to one AP, he could submit a record contains the information about it to server as a worker's task in crowdsourcing. On the server side, the system would filter and aggregate these submitted records as an employer's job. As is shown in Fig.2, there are three phases in our system design: *Filtering*, *Aggregating*, and *Feedback*.

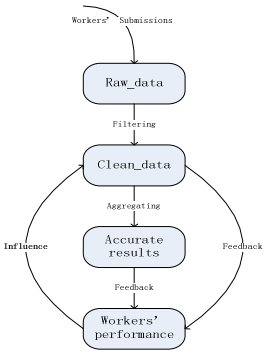

Fig. 2. System design

3.1 Filtering: Refine Submitted Contents

We call the records set submitted by workers **raw data**, since it is necessary to preprocess these records to reduce the scale of data and make our later work more accurate.

First, the records with wrong formats or values are rejected. In our design, each record is composed of a tuple like this: (*wid, apid, bid, fid, rid, lx, ly*). We use a 32-bits positive integer to denote each worker's id(*wid*), AP's id (*apid*) and AP's position which consists of a building (*bid*), a floor (*fid*), a room (*rid*) and a pair of coordinates (*lx*, *ly*) in the room respectively. So it is easy to eliminate those records with wrong formats or values. Second, we detect cheating workers. For instance, one worker may submit same records frequently during a period (i.e.10 seconds) in a crowdsourcing task. This is supposed to be a kind of cheating like that in "Page Rank" [9]. So we retain the records earliest submitted and discard the remainder.

After filtering, the scale of records set is trimmed first, which can accelerate our later work. Meanwhile, we make the records more meaningful, which would improve our aggregation. The records set after filtering is called **clean data**.

3.2 Aggregating with Quality Table

It is realistic that the position of AP is comparatively fixed during a short period such as one day. Then, we periodically aggregate the clean data to locate the position AP in this phase, which can be divided into two parts:

First, we define *ar* ("Accept Rate") to describe each worker's overall performance which is summarized from his history task. It is also used to judge the probability that the worker's submitted records would be accepted in his next task. Here, Quality Table [10] is introduced to denote such probability. For example, in Fig. 3, if one's *ar* is close to 1 or 100%, all records he submits in next task would be regarded as quality of his *ar* and accepted completely. On the other side, if one's *ar* is close to 0, the quality of his submission may be also poor and thus, discarded. In addition, one's submission would be accepted partly. This case would be introduced in detail in the Sec.4.

Second, we do aggregation on the result of the previous step with "Majority Decision Approach". For example, for AP whose *apid* is 10, there exist 100 records(including the generated ones) that show the position of it is (bid_1, fid_1, rid_1, lx_1, ly_1) and another 60 records that tell (bid_2, fid_2, rid_2, lx_2, ly_2).Then, the first result is more trustable.

Fig. 3. Quality Table

3.3 Feedback on Workers' Performance

In this phase, we update one's *ar* according to his contribution in the current task and his history performance, which is kind of feedback on *ar*.

First, for each worker, we count the number of the records in his submission which match our aggregated results, which is named, *hit*. It demonstrates one's contributions in the tasks during a short period. For example, one worker submits 100 records one day and 80 of them match the aggregated results, his *hit* is 80/100=80%.For the sake of accuracy, these three values (*match records number*, *submitted records number*, *hit*) would be all retained for future use. It has to be noted that the clean data is used as our input but not the results after the first step of the aggregate phase, since it is more reasonable.

Then, we update each worker's *ar* according to his contribution in the current task and previous ones. Here, two approaches are introduced; ***Static Period Analysis*** and ***Sliding Window Analysis***. The details would be involved in Sec.4.

4 Implementation

In this Sec, we discuss about our system implementation in detail, particularly in algorithms. Due to limited space, we focus on the algorithms part of aggregating and feedback phases.

4.1 Aggregating with Mapreduce Model

As mentioned in the design part, the clean data is aggregated periodically. We donate: [*t1, t2*] as such period, and assume there are n workers who involved in the task during the period. Besides, we have follow arguments as input:

- ***S***: clean data set derived from workers' submitted records during this period $[t_1, t_2]$. The format of each record is *(apid, bid, fid, rid, lx, ly, wid).*
- ***A***= $\{ar_1 \ldots ar_n\}$: the latest *ar* of each worker. It is supposed to be fixed during the period $[t_1, t_2]$.
- ***S1***, ***S2***: two thresholds used to judge how worker's submission would be accepted. There are three cases according to the value of *ar*:
  — *ar* is greater than ***S1***:accepted completely with quality of one's *ar*.

- *ar* is between ***S1*** and ***S2***: partly accepted with quality of one's *ar*. Meanwhile, we generate another four records near the position of the origin one and give each one quality *_q*= (1-*ar*)/4, which means that these records are also the possible positions of the AP. For the sake of simplification, it is assumed that the possible positions includes the room upstairs (*fid*+1), downstairs (*fid*-1), left (*rid*-1) and right (*rid*+1).And the *lx* and *ly* are the same as that of the origin one.
- *ar* is smaller than ***S2***:the records would be discarded.

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

```
Algorithm1: Aggregating Mapper
Input: Records Set S consists of records. Accept Rate Set A= {ar1…arn}
describe each one's honesty.S1 and S2 act as ar thresholds.
Output: Record Set S'. Each record owns an extra element quality de-
scribes the accuracy of this record.

For each record = (apid, bid, fid, rid, lx, ly, wid) in S:
      ar = A[wid]
      If ar > S1
            // completely accepted
            Output (apid, bid, fid, rid, lx, ly, ar)
      Else if ar > S2:
            //partly accepted, quality=ar.
            Output (apid, bid, fid, rid, lx, ly, ar)
            _q = (1 - ar) / 4 //generate records of positions around.
            Output (apid, bid, fid+1, rid, lx, ly, _q)
            Output (apid, bid, fid-1, rid, lx, ly, _q)
            Output (apid, bid, fid, rid+1, lx, ly, _q)
            Output (apid, bid, fid, rid-1, lx, ly, _q)
      Else:
            Continue next loop. // discarded
```

Now we do reduce job. Since each record in S' contains an *apid*, position information (*bid, fid, rid, lx, ly*) and a quality, we could first count the total quality of each tuple (*apid, bid, fid, rid, lx, ly*) and group them by *apid* later. Finally, we sort them by the quality each one has in each group and the max one is our aggregation result, for most workers think that this position is the most accurate one where the AP is located.

```
Algorithm2: Aggregating Reducer
Input: Records Set S' consists of the records whose format is (apid,
bid, fid, rid, lx, ly, quality).
Output: Each AP's id and its accurate position.

quality_table = {}
```

```
result_table = {}
For each r in S':
      k = (r.apid, r.bid, r.fid, r.rid, r.lx, r.ly)
      v = r.quality
      If k in the keyset of quality_table:
              Update its value by adding v
      Else:
              Insert (k, v) into quality_table
For each (k, v) in quality_table:
      If k.apid in the keyset of result_table:
              k' = (k.id, result_table[k.apid])
              v' = quality_table[k']
              If v'>v:
                      Update by set result_table[k]=v'
      Else:
              k' = k.apid
              v' = (k.bid, k.fid, k.rid, k.lx, k.ly)
              Insert (k', v') into result_table
For each (k, v) in result_table:
       output (k, v)
```

4.2 Feedback: Static Period Analysis and Sliding Window Analysis

Here, we present two approaches: ***Static Period Analysis*** and ***Sliding Window Analysis***, which are used to evaluate each worker's performance (*ar*). We regard one day as the minimum unit to do aggregation and calculate each one's hit.

Static Period Analysis

The approach is called "*Static Period Analysis*", for we consider each one's performance keeps stable within a comparatively short period such as one week. During the period, one's last updated *ar* is always used to judge how his submission would be accepted (completely or partly accepted or rejected). As usual, we calculate hit every day for future use. At the end of this period, his up-to-date *ar* would be summarized according to these hits. It would come into effect in next period.

Given the period *p* days, we have:

- (arc_i, rc_i, hit_i), i=1…p: tuples denote one's contribution in the task every day. It contains:
  - arc_i: count of accurate records in one's submission.
  - rc_i: counts of all records in one's submission.
  - hit_i: the proportion of *arc* in *r*.

Then, the up-to-date *ar* would be calculated as follows:

$$ar = \frac{\sum_1^p arc_i}{\sum_1^p rc_i} \tag{1}$$

Since one's performance is supposed to be stable during a short period, it is considered that each worker's *ar* keeps constant during p days. This would make one's performance not sensitive to his contribution every day. It has to be noted that we make use of the weighted average here, since it is more accurate.

Sliding Window Analysis

Static Period Analysis could reflect one worker's performance periodically and also time efficient. But it does not take one's history performance into account. Besides, one's *ar* would not ascend no matter how great his contribution is during the period. Relatively speaking, "*Sliding Window Analysis*" is more flexible and accurate.

Here, we introduce the following definitions:

- $\boldsymbol{SW_i}$, i∈N: A sliding window denotes the i_{th} dynamic period [t_{1i}, t_{2i}]. 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_{1i} and t_{2i} by 1. For instance, the third SW would be [3, p+2] and the n_{th} SW is [n, n+p-1].
- $\boldsymbol{H_i}$, i∈N: The set contains one's contribution every day during the current SW_i period. Each contribution is a tuple (*arc, rc, hit*).
- $\boldsymbol{ar_i}$, i∈N: worker's *ar* on the i_{th} day.
- ***w***: impact factor which denotes one's history performance influence.

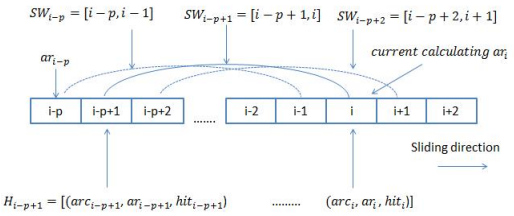

Fig. 4. Sliding Window

Then, the ar_i is calculated by:

$$ar_i = w * ar_{i-p} + (1-w) * \frac{\sum_{i-p+1}^{i} arc_j}{\sum_{i-p+1}^{i} ar_j} \tag{2}$$

First, 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, *arc* and *rc* are still used instead of hit here for accuracy. Last but not least, we also do periodically analysis like that in Static Period Analysis which would not suffer from the case in which one's contribution varies dramatically from day to day.

With *Sliding Window Analysis*, we combine one's history performance and contributions in recent tasks together, which is more persuasive and thoughtful. Meanwhile, it is also not sensitive to one's contribution each day. But the cost on time is also higher.

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. We would see the improvements as using our approaches. Time consuming and the aggregated results' accuracy are the points.

5.1 Filtering and Quality Table

In this part, the effectiveness of aggregating with filtering and quality table is evaluated. 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. We pick up a piece of period (5 days) of the results. *Static Period Analysis* is used here with 5 days (p = 5). In addition, we also have:

- Two thresholds: $S1$=0.8, $S2$=0.3.
- Each worker's initial ar is 1 or 100%.

And four cases：

- None: without filtering and quality table.
- Filtering only: only using filtering.
- Aggregating only: only aggregate with quality table.
- Both: using both filtering and quality table to aggregate.

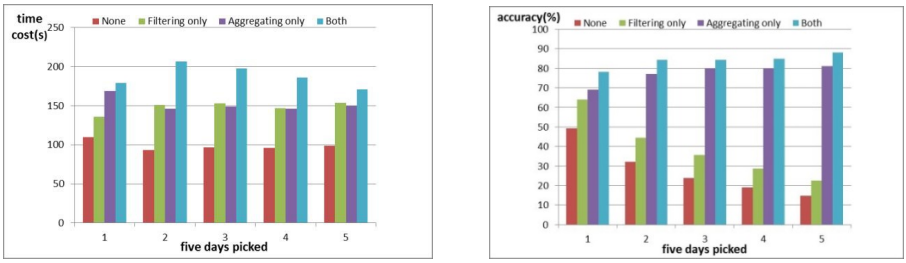

Fig. 5. Time cost and Accuracy in different cases

As is shown in the Fig.5without filtering and quality table, 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 Filtering only), since 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 case of None), 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, though time cost is a bit higher, our approach, particularly the aggregating with quality table, performs well on quality control of workers' submission.

5.2 Static Period Analysis and Sliding Window Analysis

We would still compare the difference of these two approaches from two aspects, performance and accuracy. The arguments *S1*, *S2* and the period length are set the same as those in the previous experiment.

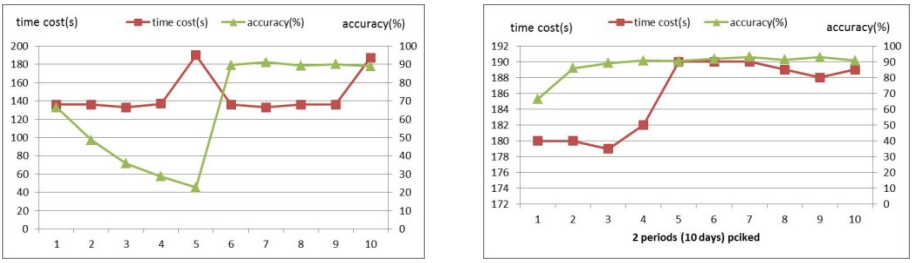

Fig. 6. Static Period Analysis and Sliding Window Analysis (w=0.25)

In Fig.6, the performance of approach Static Period Analysis is evaluated. As the period is 5 days long, the feedback phase is always called at the end of each period, such as the 5_{th} and 10_{th} day. Therefore, the time cost keeps stable during each period and ascend dramatically on these days. On the other hand, the accuracy is continuously going down during the first period, since each worker's *ar* is not updated until the end of the period. Thereby, more and more records of poor quality are accepted and the task quality is affected. After that, the accuracy begins to pick up and keeps high, for all the workers' performance has been evaluated properly.

On the other side, the evaluation of approach Sliding Window Analysis is demonstrated in Fig.6. It has to be noted that the accuracy is going up continuously and tends to be stable gradually (above about 90%), even if the worker's performance is very changeable in the period. *Sliding Window Analysis* could always obtain each worker's performance accurately, thereby improve the task quality. But the time cost is 30% higher than that of Static Period Analysis in average.

On the whole, if workers' performance keeps stable, it is efficient and acceptable to take the approach Static Period Analysis. Otherwise, Sliding Window Analysis is more preferred for its flexibility and accuracy.

6 Conclusion and Future Work

In crowdsourcing, the contents submitted by workers are always large and inaccuracy. So it is essential to control the quality of the contents. Here, we regard indoor LBS as

research background and discuss how to qualify these data efficiently and accurately. It has to be noted that our approach can be applied to many other applications such as the detection of malicious comments on E-commerce sites, since it is irrelevant to the semantic of the contents. Our focus is each worker's contribution within all workers.

Our next work will focus on data refreshing in crowdsourcing. As times 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 [11].

Acknowledgement. This paper is supported by Shanghai Municipal Science and Technology Commission under Grant No. 11dz1502500.

References

1. Howe, J.: The Rise of Crowdsourcing, Wired (June 2006), http://www.wired.com/wired/archive/14.06/crowds.html
2. Greengard, S.: Following the crowd. Communications of the ACM 54(2), 20–22 (2011)
3. 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
4. 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
5. 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
6. Lease, M., Yilmaz, E.: Crowdsourcing for information retrieval. Newsletter ACM SIGIR Forum Archive 45(2), 66–75 (2011)
7. Venetic, P., Garcia-Molina, H.: Quality control for comparison microtasks. In: The 1st International Workshop on Crowdsourcing and Data Mining, pp. 15–21
8. Mason, W., Watts, D.J.: Financial incentives and the "performance of crowds". ACM SIGKDD Explorations Newsletter 11(2), 100–108 (2009)
9. 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
10. Cheng, R., Chen, J., Xie, X.: Cleaning uncertain data with quality guarantees. Journal VLDB Endowment 1(1), 722–735 (2008)
11. Bouzeghoub, M.: A framework for analysis of data freshness. In: 2004 International Workshop on Information Quality in Information Systems, pp. 59–67 (2004)