

# Interpersonal Relationship Mining on Bulletin Board Systems

李奎翰、呂敏中、林芳而、黃宥、黃彥傑<sup>†</sup>

Department of Computer Science and Information Engineering  
National Taiwan University  
Taipei, Taiwan

## ABSTRACT

Bulletin board systems (BBS) are currently still prevalent in Taiwan among the young multitude. In Taiwan, one of the most popular BBS sites is PTT2, which is multi-utilized much like blogs and social network websites, with an estimated user base of about 1 million accounts. In PTT2, anyone may own a board and post articles onto boards. People's activity on a board (such as posting) is to some degree recorded on that board and usually publicly viewable. In this work, we use those recorded and publicly available data to mine people's interpersonal relationship on PTT2. We propose a mathematical model to calculate a person's relationship with others from their activity on PTT2, and present means to fetch and analyze data from PTT2, to process them with Hadoop, the publicly-available distributed computing framework, and calculate people's relationship using the proposed mathematical model. Finally, we briefly introduce our implemented user interface for users to query people's relationship with others. We pick a sample of 40 boards for mining and release the querying interface to those related to the sample boards in expectation for their feedback on the quality of the mined relationship results. In general, our mining results are met with overall user satisfaction.

## 1 INTRODUCTION

Bulletin board systems (BBS), despite their age, still prevail in the cyberspace in Taiwan. One of the most popular site is PTT2 [1], whose main functionality is much like a mixture of community forums, personal blogs and social networks, with about a million user accounts and a hundred thousand sign-ins per day connecting through the terminal-operating telnet protocol. In PTT2, people mainly socialize through personal boards and user-published articles along with specialized commentary system. As with any social networks, it would be interesting if we are able to see how and with whom people socialize. Yet, as naïve and archetypical, PTT2 does not have such public social network-related functionality built in mind. An observation is that people interact with others on their boards and such interactivity would usually leave traces like articles or comments. As such, in this work, we crawl and analyze information, mostly articles, on PTT2 and extract interpersonal relationship from them. We process the data with Apache Hadoop [2], a free distributed framework that supports intensive data processing inspired by Google's MapReduce framework [3], and store the data in HBase [4],

a distributed database sub-project of Hadoop inspired by Google's Bigtable [5]. People's activity on boards is treated as the source that may be used to calculate interpersonal relationship. We define three categories of the activity and calculate relationship with the mathematical model presented in Section 2 using the number of occurrences of different kinds of activity. Finally, our data is queryable through user's input on PTT2 account name (an *ID*). We support data query through a user interface consists of PHP [6] and Thrift [7] on the server side and Adobe Flash [8] as the graphics-rich client side. In the following sub-sections, we introduce the PTT2 system and give brief introduction on our idea to mine relationship on the system.

### 1.1 The PTT2 Bulletin Board System

The PTT2 BBS is based on the pttbbs system [9]. Users connect to PTT2 using the telnet protocol and operate like terminals. PTT2 consists of boards *owned* by people identified by their IDs. The ID-owns-board relation is not one-on-one; an ID may own multiple boards while a board may have more than one owners.

A board has its own unlimited number of text articles *posted* by people (not limited to the owner of the board, by default). An article consists of four sequential sections. The first is the header, of three lines, where the title of the article, the *poster* of the article and the time when the article is posted are recorded. The second section is the article body, and may be arbitrarily long in terms of the number of lines. The third section is of two lines long, containing the site signature and the IP address of the poster as recorded by the server when the article is posted. Lastly, the remaining section contains the *pushed* comments or records of *forwarding*, as introduced in the next paragraph. This section may be arbitrarily long in terms of number of lines.

People can comment an article (again, not limited to articles they have posted, and not limited to boards they own, as is with forwarding) by *pushing* the article, one comment per line, in the fourth area of the article. Also, people can *forward* articles from one board to another. Forwarding an article would leave a record on its origin article, one record per line. The IDs that push or forward articles are recorded along with the push comment or forward record. Also, the time when the push or forward takes place is recorded too. All the ID, the push comment or the forward record and the timestamp are recorded in the same line. These lines are added to the fourth article

<sup>†</sup> Author names are ordered alphabetically in Chinese. Please direct any correspondence to 呂敏中 (Min-Zhong Lu) through this e-mail address: b94075@csie.ntu.edu.tw .

section in an appending fashion, with later events being placed on the more bottom space. In this paper, we collectively call *post*, *push* and *forward* “actions”.

After signing-in to PTT2, users may enter the “board list” view to view what boards they may enter. On entering a board, users are presented with the “article list” view where the titles of the articles that have been posted on that board are listed. Then, users may select any article to view. The bulletin system is purely terminal-based and may be operated only with keyboard strokes. People usually go to boards of their interest, i.e. boards owned by those known to them.

Additionally, a board owner may mark his board private (called *invisible*) anytime, and specify a list of IDs whom he grants access to the board. This list is only viewable to the board owner and, although of our interest, may not be mined and used for relationship calculation.

The number of public boards in PTT2 is about 25000 as of writing of this paper, and we estimate the total number of articles to be 30 to 40 million. On average, an article is about tens of KB in size, which indicates that the total size of all PTT2 articles should sum to several hundred GB, which matches our belief that PTT2 is hosted with commodity hardware. In turn, it is possible for us to download all articles from PTT2 for off-line analysis. We assume a total number of just over 1 million active accounts in PTT2 according to its past facts on total registrations and sign-in counts in one day, accessible through its Xyz - History function.

## 1.2 Relationship Mining on PTT2

Our idea on relationship mining mainly relies on the fact that when people post, push or forward, they leave some traces, be they the posted article, the pushed comments or the forward records. We assume that people mainly leave such records on boards owned by people they know, i.e. they post articles on boards owned by their acquaintances, they push articles on boards owned by their acquaintances or articles posted by their acquaintances, and they forward articles on boards owned by their acquaintances or posted by their acquaintances and they forward articles to boards owned by their acquaintances. As such, we may reconstruct people’s relationship by learning where they leave their activity traces. If person A leaves many action traces in the board owned by person B, then we assume the two people are related. Also, we may emphasize certain actions’ relative importance in a relationship, such as stressing posting over pushing; this is reflected as adjustable weights in our relationship calculation function. Also, as articles record their post time and comments and forward records are recorded with their push or forward time, we also calculate relationship as a function of time. This is helpful if we want to know the variation of a person’s relationship across time, which may also be interesting.

We model the relationship calculation derived from our idea in section 2, where we further subcategorize the three actions into six actions in order to include all the people related in one action, e.g. for a forward action, we need to capture the relationship of the forwarder, the poster of the forwarded article, the owner of the board where the

forwarded article is originally on, and the owner of the board where the article is forwarded to.

The rest of this paper is organized as follows. In the next section, we model the interpersonal relationship as a function of the number of occurrences of aforementioned actions (*post*, *push* and *forward*) given a time period. Section 3 discusses the methodology on data retrieval from PTT2 and relationship analysis, and the relationship query interface. Section 4 gives brief results with our experiments. We conclude our work and give ideas for possible future work in Section 5.

## 2 RELATIONSHIP MODEL

We model the relationship of two IDs as a numeric function of the *post*, *push* and *forward* actions performed by the two IDs given a time period. The relationship  $\mathcal{R}$ , called the relationship “strength”, of  $id_1$  and  $id_2$  in the time period  $t$  is modeled as:

$$\mathcal{R}(id_1, id_2, t) = \sum \alpha_i (ACTION_i(id_1, id_2, t) + ACTION_i(id_2, id_1, t)) \quad (1)$$

In Equation 1, the relationship strength is the summation of occurrences of different actions (the *ACTION* function) multiplied by action-dependent weights  $\alpha$  ranging from 0 to 1. In this work, six actions are used for relationship calculation:

- (1) *Post on Board*. The *POSTONBOARD* function of  $(id_1, id_2, t)$  specifies the number of occurrences that  $id_1$  posts an article onto the board owned by  $id_2$ , in the given time period  $t$ .
- (2) *Push on Board*. The *PUSHONBOARD* function of  $(id_1, id_2, t)$  specifies the number of occurrences that  $id_1$  pushes articles on the board owned by  $id_2$ , in the given time period  $t$ .
- (3) *Push to Poster*. The *PUSHTOPOSTER* function of  $(id_1, id_2, t)$  specifies the number of occurrences that  $id_1$  pushes an article posted by  $id_2$  in all the boards, in the given time period  $t$ .
- (4) *Forward from Board*. The *FORWARDFROMBOARD* function of  $(id_1, id_2, t)$  specifies the number of occurrences that  $id_1$  forwards an article on the boards owned by  $id_2$ , in the given time period  $t$ .
- (5) *Forward from Poster*. The *FORWARDFROMPOSTER* function of  $(id_1, id_2, t)$  specifies the number of occurrences that  $id_1$  forwards an article posted by  $id_2$ , in the given time period  $t$ .
- (6) *Forward to Board*. The *FORWARDTOBOARD* function of  $(id_1, id_2, t)$  specifies the number of occurrences that  $id_1$  forwards an article to the board owned by  $id_2$ , in the given time period  $t$ .

To capture the actions performed from and to an ID in the calculation of the interpersonal relationship of it, we include the number of occurrences of the six actions in a bidirectional fashion. Thus, given a time period  $t$ ,  $\mathcal{R}(id_1, id_2, t)$  should always equal to  $\mathcal{R}(id_2, id_1, t)$ .

### 3 METHODOLOGY

As stated above, our analysis on interpersonal relationship sources from articles on boards in PTT2. We use self-programmed crawlers to retrieve articles, parsers to extract relationship information from the articles, Hadoop Mappers and HBase for relationship entry storage. Finally, the HBase-stored relationship information, which is essentially the materialized occurrences of the actions defined in Section 2, may be retrieved with Thrift and PHP for relationship calculation on the user interface side. The calculation is done in real-time because users are allowed to adjust the action-dependent weights (the  $\alpha$  in Equation 1) when they query for relationship results. The flow for one board is depicted in Figure 1.

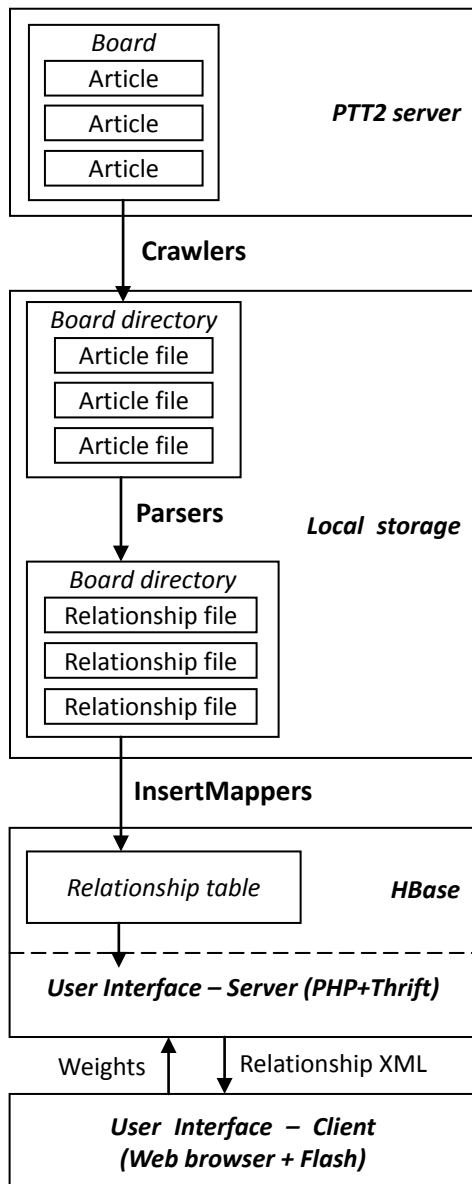


Figure 1. The flow of our methodology

#### 3.1 Data Acquisition with Crawlers

We use home-brewed crawlers for data acquisition. Crawlers are programmed in C, and use sockets to connect to PTT2 through port 23. Crawlers are implemented like state

machines and act like humans in terms of determining what keys or control sequences to send to the server according to the response text of the server. They are automated to sign in and to fulfill their tasks: retrieving the list of public boards, retrieving the list of board owners of a given board, retrieving the list of articles and subsequently the articles themselves of a given board. As crawlers use *guest*-like accounts, *invisible* boards are not crawled.

Throughout the implementation of the crawlers, the source codes of pttbbs [9] are consulted for the details of encountered problems with the bulletin system. For example, pttbbs does not retransmit a character if it detects that the position of the terminal window is currently holding the same character – instead it sends a CSI HVP sequence to skip that position [10]. Also, articles may be completely empty (which results in the system showing a special error message), or be a *pmore*-powered animation, or a BBS-Lua scripted program [9], where simply “sending the page-down key and save the result” would not work. Often, specific solutions have to be applied to bypass such problems to allow crawlers finish their jobs successfully.

Crawlers save retrieved board owners in one text file. For crawled articles, each article is saved as a separate text file and each board has its own directory containing the articles on that board. Board directories are named after their board names, and articles are named after their article numbers and the poster’s ID.

After board owner and article crawling is done, article files are parsed for relationship information (as results of the *post*, *push* and *forward* action traces).

#### 3.2 Article Parsing

Article files are parsed for relationship information. The C-programmed home-brewed parser reads the board *owner* list, and is aware of the board it is parsing. As it reads each article text file under the board directory, it parses the content of the file and outputs all relationship information (as a result of *post*, *push* and *forward* action traces) found in that file as *relationship entries*.

A *relationship entry* is essentially one defined action expressed in text form split into several columns. The first column stores the action, which may be one of the six defined in Section 2. The second and third columns store the two respective IDs of the action. The last column stores the timestamp of the action, converted into seconds elapsed since Unix epoch. This timestamp is based on the article time and calculated from article time and the time recorded in the line of the action.

In summary, each input article file has its corresponding output relationship file, which contains several relationship entries. Relationship files are placed under the same directory as article files (i.e. the board directory) and are named after the corresponding article’s article number.

Readers might ask why we do not use MapReduce for relationship parsing. The reason is that not only do parsers need to have knowledge of the input file’s article time when parsing an arbitrary line of the article, they also should only parse (push and forward) actions appearing after the last site signature and poster IP address, to avoid unwanted reading of actions having taken place on other boards and been recorded and forwarded alongside to the current parsing

board. As it is hard to achieve within Mappers to have knowledge of the content of an arbitrary line of the parsing file and the current parsing line number, we opt not to use MapReduce for less programming complexity.

After parsers transform article files into relationship files, those relationship files may be further processed by Hadoop Mappers to store relationship entries into the HBase storage.

### 3.3 Information Storage with Hadoop and HBase

We use Hadoop MapReduce framework to insert relationship entries from relationship files output by parsers into HBase database. Reducers are not used; Mappers, specifically called InsertMappers, insert data directly into HBase. Each InsertMapper processes one relationship entry line by splitting it into strings, each as an entry column, and storing it into HBase. InsertMappers do not produce any MapReduce output for collection.

Our HBase consists of only one table responsible for storing all the relationship entries. The row keys of the table are IDs. We define 12 column families, two for each of the six actions defined in Section 2, to achieve the bidirectional relationship model as mentioned in the section. The column qualifier is the ID with which the row-key ID is associated for the action occurrence stored in the cell specified by the row key, the column key and the timestamp. Every cell is set to be able to contain a very large number of versions available. We refer readers to [5] for the details of the HBase database structure. The actual data string stored in the cell is the board and article number where the action occurs. We store this information for possible future reverse look-ups. Thus, consider this example relationship entry in relationship file `/SampleBoard/3.rel1`:

```
postOnBoard id1 id2 548247900
```

Note this would mean `id2` is an owner of `SampleBoard`. This relationship entry would be inserted into the HBase table as two records:

```
{"id1", "postOnBoard:id2", 548247900}
    → "SampleBoard/3"
{"id2", "boardWasPostedBy:id1", 548247900}
    → "SampleBoard/3"
```

The `boardWasPostedBy` column family is the mnemonic for the “reverse” relationship of the `Post on Board` action.

It is worth noting that an ID does not have to be an owner of a board to be stored in the database, since it may be the case that the ID has activity on boards none of which it has ownership of.

InsertMappers insert the relationship entries from all the parser-output relationship files into the database. Having harvested all the action occurrences of all articles on boards, the system is ready for interpersonal relationship querying.

### 3.4 Relationship Querying with User Interface

The user querying interface provides the interface where the relationship of the user-specified ID may be queried. It consists of the server side, where web server-hosted PHP

scripts communicate with HBase through Thrift to retrieve all the relationship entries (i.e. the actions) related to the specified ID, do relationship strength calculation as outlined in Section 2 using the user-provided weights, and output the relationship information for the client side’s use. The client side is a web browser loading a Flash file, responsible for taking user input on the ID to query, the adjust of the weights, and most importantly, displaying the relationship in a visualized way.

The server side, upon receiving request from the client side that consists of the ID to query and the six weights (as in Section 2), connects to HBase through Thrift and asks for data stored in the row whose key is the ID. It then categorizes the returned data (i.e. the action occurrences) according to the column families (i.e. the type of the actions) and column qualifiers (i.e. the ID that the querying ID is related to through the actions). The categorized data is further subcategorized according to timestamp, so that we get one relationship strength value for each “time period” (defined as a season in this implementation). A special time period is the “all-time” period, where all the occurrences regardless of the timestamp for the related-ID are calculated. After this subcategorization is done, the occurrences of the actions are weighted and summed using the weights for action, for each related-ID and for each time period. The list of related-IDs and their relationship strength value for each time period is output as XML to the client side. On a side note, the number of related-ID for each time stamp is restricted to 100 so as to avoid overflowing the user’s screen.

The client side’s Flash view is designed to fit a 1024x768 screen. It consists of a textbox, where users may specify the ID whose relationship to query, and an option pane, where users can adjust the six weights. The client side uses HTTP to communicate the ID and the weights to the server side. On receiving the relationship XML, the client side parses it and displays the relationship as depicted below. Each ID is shown as a tag. The queried ID is in the center of the view and is white. The related-ID tags are placed around the center ID and are colored. The distance of a related-ID tag to the center tag, and the size of the related-ID tag depend on that ID’s relationship with the queried ID. The stronger the relationship is, the closer and larger the tag is. Also, the color saturation of a related-ID tag is also reciprocal to that ID’s relationship strength with the queried ID. Lines connect the center queried-ID tag to other tags to give visual cues. By default, only 50 most strong related-IDs are shown, yet users may adjust to show fewer or more, up to the 100 limit. Also, the default time period to show is “all time”, and users may alternatively select other time periods (seasons) to show.

## 4 EXPERIMENTS AND RESULTS

We installed the Hadoop environment, including the HDFS and the HBase database, on a 3-node distributed cluster, with each node powered by two 8-way capable AMD K8 Opteron Dual-Core CPUs running at 2.0GHz and equipped with 1GB main memory and 20GB secondary storage. A sample of about 40 selected public boards owned by those mostly known to us with a total number of

about 60000 articles was crawled and subsequently analyzed for relationship information. Our InsertMapper jobs continued to run for several hours to finish analyzing articles and to store relationship results into the database.

The querying user interface was released to the public related to the sample boards, after InsertMapper jobs finished, in expectation for user feedback on the quality and accuracy on the results. In general, most users gave positive feedback and review on the relationship query results. Weight values as in Equation 1 were defaulted as follows: we used near-1 values for  $\alpha_{POSTONBOARD}$ , near-0.5 values for  $\alpha_{PUSHONBOARD}$  and  $\alpha_{PUSHTOPOSTER}$ , and depreciated forward-related  $\alpha$  values with near-0 values.

## 5 CONCLUSION AND FUTURE WORK

In this work, we seek to mine interpersonal relationship on PTT2, a popular bulletin board system in Taiwan where people communicate and interact with others with their personal boards. We propose a mathematical relationship model to calculate people's relationship based on their activity on PTT2. We use our home-brewed crawlers and parsers to extract such activity information from articles on PTT2 and transform them into relationship entries. We use Hadoop and HBase for fast and distributed store and retrieval of the large amount of relationship data, and a user interface is designed to access the data and calculate the relationship of user-querying IDs. Preliminary testing on a sample of 40 boards is met with overall user satisfaction. In the future, we expect to extend the data magnitude first to 569 boards with roughly 540000 articles, which are owned by those we expect to grant us permission for relationship mining. We expect to further broaden the data set size to the whole PTT2 site should we be able to get the permission of the all users on PTT2.

## 6 REFERENCES

- [1] telnet://ptt2.cc
- [2] <http://hadoop.apache.org/>
- [3] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the 6<sup>th</sup> Symposium on Operating System Design and Implementation*, Dec. 2004.
- [4] <http://hadoop.apache.org/hbase/>
- [5] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A Distributed Storage System for Structured Data. In *Proceedings of the 7<sup>th</sup> Symposium on Operating System Design and Implementation*, Nov. 2006.
- [6] <http://www.php.net/>
- [7] <http://incubator.apache.org/thrift/>
- [8] <http://www.adobe.com/products/flash/>
- [9] <http://www.ptt.cc/index.source.html>
- [10] Ecma international. Standard ECMA-48: Control Functions for Coded Character Sets. 5<sup>th</sup> edition (June 1991)