# Final Year Project Interim Report

# Predicting of Web 2.0 Thread Updates

U096883L Shawn Tan

## 1   Introduction

With the advent of Web 2.0, sites with forums, or similar thread-based discussion features are increasingly common.

Table 1 shows us that many of the popular Web 2.0 sites have comment features. This suggests that content on the web is increasingly being created by users alongside content providers. While mining structured, curated content from sites like Amazon, for data like prices is easy and effective, data that can be obtained from user-generated content are of a different nature. One may be able to infer public sentiment about a given product that would not be readily available from an e-commerce site.

In some cases, news may travel more quickly through such online community discussion than through traditional media. Users also typicaly discuss purchased products bought online via these forums, and companies that want to get timely feedback about their product should turn to data mined from such sites.

A naive way of getting timely updates is to aggressively hit the pages repeatedly downloading the pages at a very frequent rate. However, the number of pages in a forum site are far too large to perform this efficiently on every forum.

| | T | FB L | FB S | G +1 | L | DL | C | PV | Follows |
|---|---|---|---|---|---|---|---|---|---|
| http://www.lifehacker.com | 1 | 1 | 1 | | | | 1 | 1 | |
| http://digg.com/ | 1 | 1 | | | 1 | 1 | 1 | 1 | |
| http://9gag.com/ | 1 | 1 | 1 | 1 | 1 | | 1 | 1 | |
| http://www.flickr.com/ | | | | | 1 | | 1 | 1 | |
| http://news.ycombinator.com/ | | | | | 1 | | 1 | | |
| http://stackoverflow.com/ | | | | | 1 | | 1 | 1 | |
| http://www.youtube.com/ | | | | | 1 | 1 | 1 | 1 | |
| http://www.reddit.com/ | | | | | 1 | 1 | 1 | | |
| http://www.stumbleupon.com/ | | | | | 1 | | 1 | 1 | |
| http://delicious.com/ | 1 | 1 | | | | | 1 | 1 | 1 |

Table 1: Features of popular Web 2.0 sites

T        = Twitter mentions
FB L     = Facebook Likes
FB S     = Facebook Shares
G +1     = Google +1
L        = Likes (Local)
DL       = Dislikes (Local)
C        = Comments
PV       = Page Views
Follows  = Site-local feature for keeping track of user's activities

In this report, we will discuss the *timeliness* of our crawler to maintain the freshness of the local database, which refers to how new the extracted information is. Web crawlers can be used to crawl sites for user comments and threads for postprocessing later. Web crawlers which maintain the freshness of a database of crawled content are known as incremental crawlers. Two tradeoffs these crawlers face cited by Yang et. al. 2009 [10] are *completeness* and *timeliness*. *Completeness* refers to the extent which the crawler fetches all the pages, without missing any pages. *Timeliness* refers to the efficiency with which the crawler discovers and downloads newly created content. We focus mainly on timeliness in this project, as we believe that timely updates of active threads are more important than complete archival of all threads in the forum site.

Let us define all such thread-based discussion styled sites as forums. Ideally, an incremental crawler of such user-generated content should be able to maintain a fresh and complete database of content of the forum that it is monitoring. However, doing so with the previously mentioned naive method would (1) incur excessive costs when

downloading un-updated pages, and (2) raise the possibility of the web master blocking the requester's IP address.

Thus, we need a strategy of revisiting pages that will reduce the cost of downloading unchanged pages, while at the same time downloading them as soon as possible after it's update. This year-long project proposes to use content-based features of a given thread to predict its next update time. We argue, that the content within the posts of the thread should be important in predicting the thread updates, and propose a Hidden Markov Model (HMM)

# 2   Related work

## 2.1   Refresh policies for incremental crawlers

In order to devise such a strategy, we need to predict how often any user may update a page. Some work has been done to try to predict how often page content is updated, with the aim of scheduling download times in order to keep a local database fresh.

Many such works have used the Poisson distribution to model page updates. Coffman et. al. [5] analysed the theoretical aspects of doing this, showing that if the page change process is governed by a Poisson process $\lambda e^{-\lambda \mu}$, then accessing the page at intervals proportional to $\mu$ is optimal.

Cho and Garcia-Molina trace the change history of 720,000 web pages collected over 4 months, and showed empirically that the Poisson process model closely matches the update processes found in web pages[2]. They then proposed different revisiting or refresh policies [3, 4] that attempt to maintain the freshness of the database.

The Poisson distribution were also used in Tan et. al. [7] and Wolf et. al. [9]. However, the Poisson distribution is memoryless, and in experimental results due to Brewington and Cybenko [1], the behaviour of site updates are not. Moreover, these

studies were not performed exclusively on online threads, where the behaviour of page updates may be very different from that of static pages.

Yang et. al. [10], attempted to resolve this by using the list structure of forum sites to infer a sitemap. With this, they reconstruct the full thread, and then use a linear-regression model to predict when the next update to the thread will arrive.

Online forums and bulletins have a logical, hierarchical structure in their layout, which typically alerts the user to thread updates by putting threads with new replies at the very top of the thread index. Yang's work exploits this as well as their linear model to achieve a predicton of when to retrieve the pages. However, this is not so for comments found on blog sites or discussion threads in an e-commerce site about a certain product and the lack of these pieces of information may result in a poorer estimate, or no estimate at all.

For example, a thread in a technical forum about a Linux distribution may start out as a question. Subsequent questions that attempt to either clarify or expand on the original question may then be posted, resulting in a quick flurry of messages. Eventually, a more technically savvy user of the forum may come up with a solution, and the thread may eventually slow down after a series of messages thanking the problem solver. Suppose 10 days later, someone with a slight variation of the same problem posts on the thread again. A crawler solely on the age of the thread to determine its download rate of the thread may not update itself with the thread.

Our perspective is that the available content on the thread at the time of the retrieval should also be factored into the model used to predict the page updates. Next, we look at some of the related work pertaining to thread content.

## 2.2  Thread content analysis

While there is little existing work using content to predict page updates, we will review some existing work related to analysing thread-based pages which we think will aid us in our efforts to do content-based prediction.

Wang et. al did work in finding out linkages between forum posts using lexical chaining [8]. They proposed a method to link posts using the tokens in the posts called $Chainer_{SV}$. While they do analyse the content of the individual posts, the paper does not make any prediction with regards to newer posts. The methods used to produce a numeric similarities between posts may be used as a feature to describe a thread in its current state, but incorporating this into our model is non-trivial.

With these related work in mind, we next propose our modelling of a thread as a Markov chain, and our approach to solving the problem.

# 3  Approach

Our objective is to minimise the cost of fetching pages, while still being able to keep our local database of pages updated. To this end, we model a thread as being governed by a Markov chain. In each state, a certain set of words and their frequencies will be emitted with certain probability. Each state has a certain probability of transitioning back to itself or to the other states after the next observation or refreshing of the thread.

We could modify Kleinberg's use of HMMs to detect bursts in his email stream [6]. Instead of using time differences as the observations we would use the content of the pages.

As an example, if we assume that a thread is governed by a set of discrete states, say 'Active'($q_0$) and 'Inactive'($q_1$), and each of these states stochastically transit from one

to the other or back to itself. The states also produces certain observable characteristics $\mathbf{z}$, with probability $P(\mathbf{z}|q)$ where $q$ is a state. In the case of our thread content, possible observations include:

**Average length of a post** The average length of a post may be indicative of the activity on the thread. An active discussion may result in long posts to explain steps, or to elaborate arguments.

**Word frequencies** Certain words may appear often in an active or inactive thread. A thread that may be transitioning toward an 'Inactive' state may be more likely to have higher occurences of 'Thank you', for example.

**Time between posts** We could include the average time differences, since they are indicative of a thread that is still relatively active.

Since new posts are more likely to be influenced by more recent posts, a form of weighted importance should be introduced, giving the later posts greater influence on the prediction.

With these observations, the states that produce them, and the transitions between the states, given a thread, we may be able to produce a sequence of states that are highly likely to generate the observations. If we assume a certain rate of updates in each state, we would be able to calculate the expected interval between the last post and a future post.

# 4    Work in progress

## 4.1    Extracting data

In order to determine if there is a relationship between the lexical chains in a thread and the arrival of the next post, we need to collect data from a popular forum with

fairly long threads. We chose `http://forum.hardwarezone.com.sg` as our data set.

We have been scraping the Hardwarezone forums daily in order to build a database of threads and their User, Timestamp, Content tuples. The scraped data from the forums are still currently in their raw HTML format. We need to extract the feature vectors in order to train the HMM.

## 4.2   Implementation of baseline

We intend to implement the algorithm used in Yang et. al. [10], as a baseline to evaluate our system. However, since the algorithm uses the overall sitemap as additional information, only the linear regression model used to predict future incoming posts will be used. We will also use an SVM, trained using discretised time intervals and the extracted feature vectors.

## 4.3   Modeling of threads

Modelling the problem as an HMM problem requires us to define a set of possible states that emmit the observable features on the thread. This requires more in-depth understanding of how users behave on a forum. Once this is done, a sets of training data would have to be annotated as produced by the different states. We also need to train the transitions between the states.

The feature vector to be extracted from the current thread would also need to be defined. More study must be done to see the most effective way to weigh the features extracted per post to make the more recent posts more 'important' in the extracted feature vector.
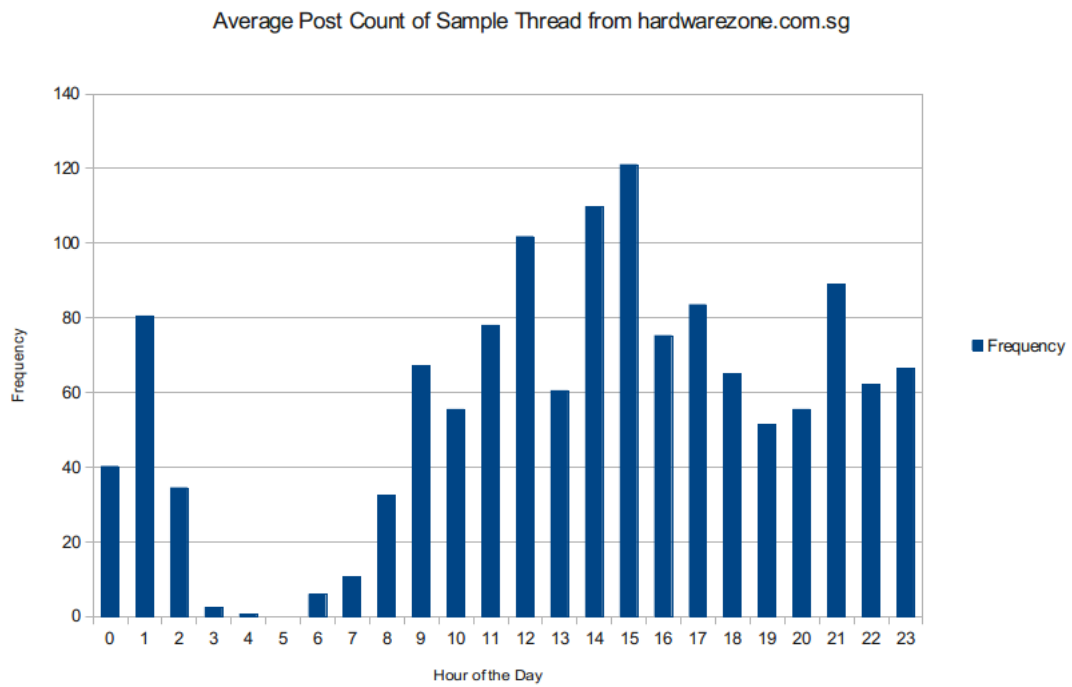
Figure 1: The post frequency of a thread from forums.hardwarezone.com.sg shows a pattern cycle of the forum's users.

### 4.3.1  Proposed model

From our initial analysis of the extracted data from the Hardwarezone forums, we have made the following observations about non-sticky threads in the forum:

1. There are threads that are very short-lived, and some that are long lived. This is largely dependent on the thread content. For example, a user looking to buy/sell a product that no one has an interest in.

2. The long-lived threads typically have a pattern to the frequency of postings. A high frequency of posts are seen around lunch time (12-3 PM) and another spike is seen during the night at 9 PM. (Figure 4.3.1). It is yet to be determined if this type of cycle is site-wide or thread-specific.

Our initial model aims to incorporate these observations. A new thread (represented by $q_0$ which only has one post, may transition to either an active or active but 'sleeping' thread, or an inactive thread.

We define active here as threads with a high post frequency, and inactive threads and as posts with low post frequency. Since our observations show that threads have different levels of post frequencies due to usage patterns consistent with user's sleep cycles and work commitment, we use 3 states in total to represent the state of a thread with more than 2 posts. A 'day' state (represented by $q_{AD}$) and a 'night' state (represented by $q_{ND}$). The inactive threads are represented by the state $q_I$

The intuition behind this is the fact that active threads can transition to an inactive state, but the presence of a new post is able to spark off a new discussion within the thread, causing it to be an active thread again.

The graphical representation of this automata is seen in Figure 2.

Our initial collection of data to identify changes in state for active threads to inactive threads do not show any observable patterns (See Figure 3). More data has to be

collected and analysed before conclusions can be made. Statistics of textual content, along with data for a more complete set of threads may provide some insight to their behaviour.

# 5 Conclusion

A lot remains to be done over the next half of the academic year. More data for more sites with thread-based comment systems have to be collected in order to come up with a more general model. The baseline algorithms have to be implemented in order to perform evaluation. Most importantly, our algorithm has to be implemented and then evaluated against the baseline.

# References

[1] BREWINGTON, B. E., AND CYBENKO, G. Keeping up with the changing web. *Computer* (2000), 52–58.

[2] CHO, J. The evolution of the web and implications for an incremental crawler. *Science* (1999), 1–18.

[3] CHO, J., AND GARCIA-MOLINA, H. Effective page refresh policies for Web crawlers. *ACM Transactions on Database Systems 28*, 4 (Dec. 2003), 390–426.

[4] CHO, J., AND GARCIA-MOLINA, H. Estimating Frequency of Change.

[5] COFFMAN, E., AND LIU, Z. Optimal robot scheduling for web search engines. *Sophia* (1997).

[6] KLEINBERG, J. Bursty and hierarchical structure in streams. *Data Mining and Knowledge Discovery* (2003), 1–25.

[7] TAN, Q., ZHUANG, Z., AND MITRA, P. Designing efficient sampling techniques to detect webpage updates. *Proceedings of the 16th 1*, 3 (2007).

[8] WANG, L., AND McCARTHY, D. Predicting Thread Linking Structure by Lexical Chaining. *Proceedings of the* (2011), 76–85.

[9] WOLF, J., SQUILLANTE, M., AND YU, P. Optimal crawling strategies for web search engines. *on World Wide Web* (2002).

[10] YANG, J., CAI, R., WANG, C., AND HUANG, H. Incorporating site-level knowledge for incremental crawling of web forums: A list-wise strategy. *on Knowledge* (2009), 1375–1383.
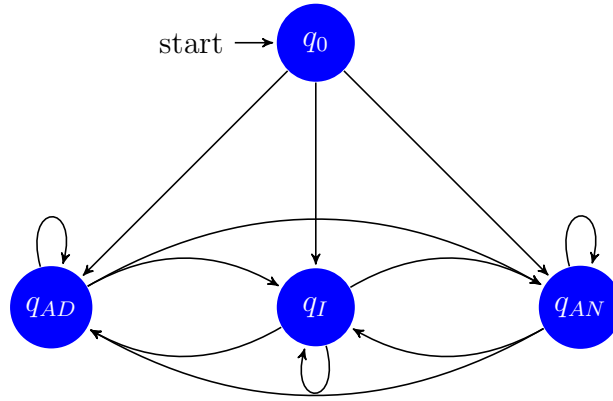
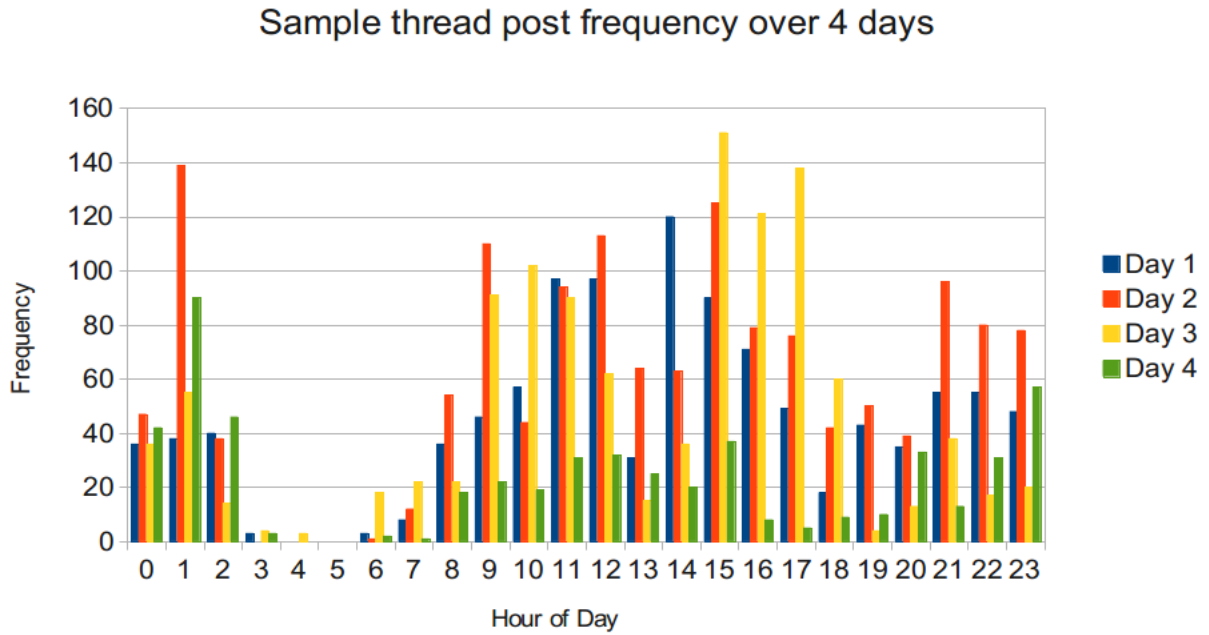Figure 2: Preliminary modelling of a thread as a probabilistic automaton



Figure 3: Not enough information for conclusion to be drawn for post frequencies over four days.