

國立交通大學

資訊科學與工程研究所

碩士論文

社群網路興趣探勘

Mining Interest Topics from Plurk

研究生：李宜謙

指導教授：蔡錫鈞 教授

中華民國一零一年七月

社群網路興趣探勘

Mining Interest Topics from Plurk

研究生：李宜謙

Student: Yi-Chien Lee

指導教授：蔡錫鈞

Advisor: Shi-Chun Tsai

國立交通大學

資訊科學與工程研究所

碩士論文



Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

July 2012

Hsinchu, Taiwan, Republic of China

中華民國一零一年七月

摘要

近年來隨著社群網路服務的蓬勃發展，越來越多使用者透過微網誌服務認識新朋友。然而，利用微網誌服務來認識新朋友會遇到一個問題：例如當你看到一位微網誌使用者的大頭照覺得她是你喜歡的異性類型，進而想要認識這位網友，那麼你可能得先大致看完她的留言內容，先大致了解這位網友對哪些話題有興趣後開始嘗試談話。因為大部分的微網誌服務沒有提供類似 Facebook 的個人資訊頁面，陌生人無法透過閱讀使用者主動提供的資訊去投其所好，加上微網誌的文章發表量很大，想要看完一個人的留言去推測他的興趣是很困難的。此外，如果想要認識的異性網友沒有公開她的留言，那麼想要一親芳澤的難度就更高了，因為你無從得知她對那些話題有興趣。

為了解決上述的問題，我們針對噗浪 (Plurk) 微網誌服務設計了一套興趣探勘系統。這套探勘系統能夠快速整理受測者發表過的關鍵字並視覺化該使用者的交友網路。若受測者將他的時間軸設定為私密狀態，意即留言內容不公開，我們透過整合該受測者朋友的留言資訊去推測他會感到興趣的話題與關鍵字。我們也可將受測者感興趣的關鍵字使用於個人化、廣告業務以及朋友推薦等應用。

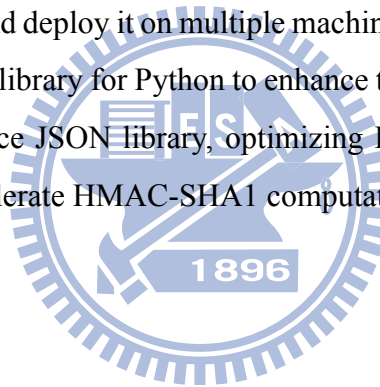
為了快速蒐集噗浪上的資訊，我們開發了一套基於 ZeroMQ 的分散式資料蒐集框架並佈署到多台機器上增加資料蒐集的速度。此外，由於噗浪的 Python API 函式庫效能不甚理想，所以我們透過更換 JSON 函式庫、強化 HTTP 連線管理以及撰寫 OpenSSL 擴充套件加速 HMAC-SHA1 運算速度等手段改善效能瓶頸並大幅增加蒐集的效率。

ABSTRACT

People started to make friends with micro-blogging service in recent years; however, it is difficult to read all messages posted by those whom you are interested in but not familiar with to find out what he/she is interested in to start a conversation. Furthermore, unlike blog or Facebook, most of micro-blogging services do not provide profile functionality (self-description page) for users to describe him/her-self for people to know what he/she is interested in.

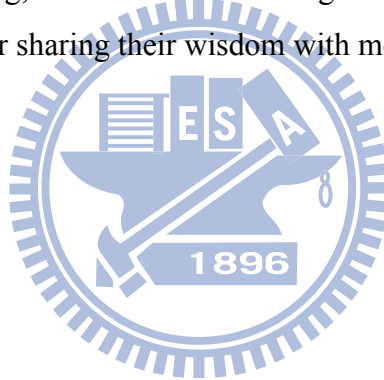
To address this demand, we build an online Social Networking Service Discovery (SNSD) system for Plurk users (plurkers) to find out a plurker's interest topics/keywords and relationships/connections. The results are presented in graphics on a web browser. With the derived interests and relationships/connections, applications of the system include friend recommendations and personalized advertisements.

To enhance crawling performance, we develop a distributed crawling system based on ZeroMQ messaging protocol and deploy it on multiple machines to crawl data from Plurk. In addition, we patch the Plurk API library for Python to enhance throughput by replacing the standard library with high-performance JSON library, optimizing HTTP connections and customizing Python C-extensions to accelerate HMAC-SHA1 computation.



Acknowledgments

I would like to thank my parents for offering me an opportunity to accomplish this thesis. I am greatly indebted to Dr. Shi-Chun Tsai, my advisor, for his patience, guidance and encouragement. I also wish to thank Dr. Wen-Guey Tzeng, Dr. Shih-Kun Huang, Dr. Tyng-Ruey Chuang, Dr. Ying-ping Chen, Dr. Tzong-Han Tsai, Dr. Cheng-Zen Yang, Dr. Yi-Yu Liu, Mr. Jim Huang, Dr. Min-Zheng Shieh, Mr. Min-Chuan Yang, Mr. Chuan-Yu Tsai, Mr. Min-Cheng Chan, Mr. Huai-Sheng Huang, Mr. Chun-Yuan Cheng and all the other members in the CCIS research group and CSCC for sharing their wisdom with me.



誌謝

首先誠摯的感謝指導教授蔡錫鈞博士，老師悉心的教導使我得以一窺圖論與演算法領域的深奧殿堂並指點我正確的方向，使我獲益匪淺。老師對學問的嚴謹更是我輩學習的典範。

本論文的完成另外亦得感謝撲浪的創辦人雲惟彬先生與 Oxlab 的 Jserv 大大提供意見，以及元智大學蔡宗翰教授和鄭鈞元同學的支持。因為有你們的幫忙，使得本論文能夠更完整而嚴謹。

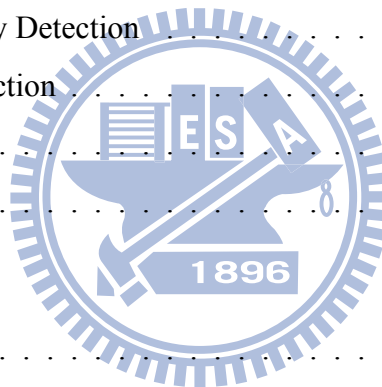
感謝楊名全、謝旻錚、詹珉誠以及黃懷陞學長協助我進行研究，且總能在我迷惘時為我解惑，也感謝蔡權昱同學的幫忙，恭喜我們順利走過這兩年。實驗室的方智誼、邱韜璋以及系計中的許伯羽、游傑、林宏昱學弟們當然也不能忘記，你們的協助我銘記在心。

感謝我的摯友馬安妤、李金樺、黃暄仁以及許健聖在我陷入低潮時能鼓勵我並使我重拾研究的熱情。

因為需要感謝的人太多了，就感謝少女時代吧！最後，謹以此文獻給我摯愛的雙親。

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Related Work	8
1.3	Challenges	11
1.4	Approach	11
1.4.1	Community Detection	11
1.4.2	Data Collection	12
1.5	Results	13
1.6	Thesis Structure	15
2	System Architecture	16
2.1	Overview	16
2.1.1	Social Networking Service Discovery (SNSD) System	16
2.1.2	Distributed Crawling System	16
2.2	SNSD System Design and Architecture	18
2.3	Crawling System Design Considerations	19
2.3.1	Concurrent Programming	19
2.3.2	Messaging Protocol	22
2.3.3	Data Serialization Format	24
2.3.4	Datastore	27
2.3.5	Task Queuing for Crawling	28
2.3.6	Security	29
2.4	Distributed Crawling System Architecture	30
2.4.1	System Architecture	30
2.4.2	Work Flow	32



2.4.3	Task Queuing	32
3	Implementation Details	35
3.1	Data Collection	35
3.1.1	Overview	35
3.1.2	Plurk API and Library	36
3.1.3	Library Optimization	37
3.2	Preprocessing	39
3.2.1	A Plurk and Its Data	39
3.2.2	Elements of a Plurk	40
3.2.3	URL Filtering Mechanism	41
3.2.4	Tokenization	44
3.2.5	Plurks Preprocessing	48
3.3	Community Detection	51
3.3.1	Snowball Sampling	51
3.3.2	Modularity and Louvain Algorithm	51
3.3.3	Filtering	54
3.4	Interest Hierarchy Model	54
3.5	Datastore Architecture	57
3.6	Celery Task Queue	59
3.7	Celery Cluster Layout and Worker Configurations	62
3.8	Delta Cluster Deployment	65
4	Experiments	69
4.1	Environment	69
4.2	Performance Benchmarks	70
4.2.1	Python JSON Libraries	70
4.2.2	Python Serialization	72
4.2.3	HMAC-SHA1	72
4.2.4	Python Plurk API Library	75
4.2.5	Redis Connection	76
4.3	Interest Derivation	78
4.4	Website Implementation	79

5	Conclusions and Future Works	84
	Bibliography	86
	Appendix A Diskless Linux Cluster Installation	94
A.1	Base System	94
A.2	Network Block Device (NBD) Server	97
A.3	DHCP and PXE Server	99
	Appendix B MongoDB Cluster Installation	100
B.1	MongoDB Installation	100
B.2	Replica Sets	104
B.3	Sharding	106



List of Tables

3.1	Comparison of hierarchical data model design from Ref. [45]	55
4.1	Machine specifications and roles	69

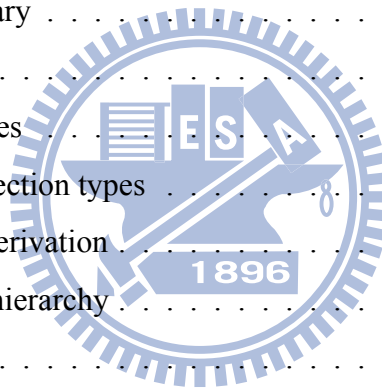


List of Figures

1.1	Plurk, Weibo, and Twitter daily visitors count graph (Taiwan only)	3
1.2	Alexa traffic rank for Plurk	3
1.3	Visitors statistics by Alexa	3
1.4	Plurk timeline	4
1.5	A sample plurk	4
1.6	Plurk profile: extra information	5
1.7	A Plurk user with his plurks public	6
1.8	A Plurk user with his plurks private	6
1.9	Interests derivation from public or private plurkers	7
1.10	Google Analytics for Go!Plurk	9
1.11	Go!Plurk system flow	9
1.12	Sample news articles for training	10
1.13	Sample plurks for training	10
1.14	Interest pie chart generated by Go!Plurk	11
1.15	Architecture for two-tier parallel crawler	13
1.16	SNSD website overview	14
2.1	Components in the SNSD system and crawling system	17
2.2	Work flow for generation of interest keywords hierarchy	17
2.3	GIL Behavior	19
2.4	Computation bound	20
2.5	Thread	20
2.6	Event Loop	21
2.7	Coroutine	22
2.8	Coroutine with I/O	23
2.9	Coroutine with Event Loop	23

2.10	Transports of ZeroMQ	24
2.11	RPC over AMQP	25
2.12	RPC over ZeroMQ	25
2.13	Various message formats within the crawling system	28
2.14	RPC over AMQP on Crawling	29
2.15	RPC over ZeroMQ on Crawling	29
2.16	Queuing work flow	30
2.17	Distributed crawling architecture	31
2.18	Messaging patterns between components	32
2.19	Work flow of crawling system	33
2.20	States and Redis data type of the task queue	34
3.1	Plurk mobile view	36
3.2	HTTP persistent connection	38
3.3	Elements of a plurk	42
3.4	URL filtering for a file	44
3.5	URL filtering for an image	45
3.6	URL filtering for a Youtube link	45
3.7	URL filtering for a web page	46
3.8	Demonstration of the normalize function	47
3.9	Demonstration of the tokenize function	49
3.10	Demonstration of the preprocessing	50
3.11	Visualization of the steps of Louvain algorithm	54
3.12	Plurk profile: general information	55
3.13	A sample Closure Table	57
3.14	Table Relationships	58
3.15	MongoDB cluster architecture	60
3.16	MongoDB cluster configuration	60
3.17	Celery cluster architecture	64
3.18	OpenStack security group configurations	64
3.19	CS workstation cluster architecture	65
3.20	Servers and racks donated by Delta, Inc.	66
3.21	A single Delta server	67

3.22	Delta server with VGA card	67
3.23	Servers installed in rack	68
3.24	Delta cluster architecture	68
4.1	Encoding performance	71
4.2	Decoding performance	71
4.3	Big data performance	71
4.4	Memory usage of JSON libraries	72
4.5	Serialization performance	73
4.6	Memory usage of serialization libraries	73
4.7	Encoded data size	74
4.8	HMAC-SHA1 performance	75
4.9	Original API library	76
4.10	Enhanced API library	76
4.11	Improvements	77
4.12	Redis binding modes	77
4.13	Redis remote connection types	78
4.14	Result of interest derivation	79
4.15	Interest keywords hierarchy	80
4.16	Interest tag cloud	81
4.17	View communities in pack layout	81
4.18	View communities in treemap layout	82
4.19	Focus community on pack layout	82
4.20	Focus community on treemap layout	83
4.21	Parameter filter	83



List of Algorithms

1	URL filtering mechanism	43
2	Tokenization process	48
3	Louvian algorithm	53



Chapter 1

Introduction

1.1 Motivation

Social networking service on the Internet can be traced back to mid 1990s when providers such as Geocities launched the service in the form of generalized online communities, which offered two ways of inter-personal interaction: chat rooms and personal web pages.

Rapid development of the Internet led to the next generation of social networking services in late 1990s through early 2000s. This new generation of services include the following two features, among others: user profiles and blog.

User Profiles allow users to define lists of “friends” and search for other users with similar attributes [71]. Active providers in this period include SixDegrees.com (1997-2001), Friendster (2002), MySpace (2003), LinkedIn (2003) and Facebook (2004).

Blogs emerged in the late 1990s. A blog (a portmanteau of the term web log) [12] is a discussion or information site published on the Internet. Most blogs operate in an interactive manner, meaning that visitors are allowed to leave comments or even messages to each other. Bloggers not only produce content to post on their blogs but also build social relations with their readers and other bloggers [28]. In that sense, blogging can be regarded as a form of social networking.

Microblogging, “the SMS of the Internet,” [24] is a broadcast medium in the form of blogging. A microblog differs from a traditional blog in that its content is much smaller in size. For example, Twitter and Plurk enable their users to write and read text-based messages up to 140 characters in length. Most social networking websites offer their own microblogging feature via “status updates.” [92] Leading micro-blogging providers include Twitter (2006), Facebook,

Sina Weibo (2009) and Plurk (2008).

Use of social networks for making friends and keeping in touch with friends has become popular recently.[16, 38, 27] Micro-blogging services such as Plurk, Twitter, and Weibo are popular in Taiwan. According to Alexa [4], Plurk ranked 45th in Taiwan and 2,014th worldwide on June 1, 2012, as shown in Figures 1.2 and 1.3. This statistics indicates that Plurk is an active social network service especially in Taiwan.

Furthermore, according to Google Trends on June 1, 2012 (Figure 1.1), Plurk is among the most popular micro-blogging services in Taiwan, probably due to its user-friendly interface named “Timeline” as shown in Figure 1.4. As such, this study is based on the Plurk community.

Freshman students exchange their Plurk accounts via bulletin board system (BBS) to get familiar with each others. Open source developers share their Twitter and Plurk accounts in their presentation slides for audiences to contact them if they have any comments or are interested in the project.

Most micro-blogging services allow users to make two types of relationships: friend and follower. The friend relationship requires that both individuals confirm they are friends while the follower relationship can be established without confirmation. As such, followers may not be connected to the target individual in real life.

In Plurk, individual’s profile information (Figure 1.6) is not publicly available. Users can set their conversations (plurks) as shown in Figure 1.5, public (Figure 1.7) or private (Figure 1.8), i.e. only specific users or friends of those who post the contents can view the contents while anonymous users and followers are not allowed to.

Given the constraints, it’s difficult to know someone via plurks even though all his/her plurks are public, as shown in Figure 1.9(a) and 1.9(b). However, affiliation and interests information can be derived from his/her friends in order to conjecture who he/she is or what he/she is interested in even though we know nothing about him/her, as shown in Figure 1.9(c).

According to the hypothesis, we build an online analysis system for users to find out what he/she might be interested in by providing his/her Plurk account name.

After generating interest topics/keywords information about someone whom you are interested in, by our system, you can use the information to refer him/her to your friends who share the same interests. Search engine service provider and commercial company can use our system to build user profiles for customized service and advertising.

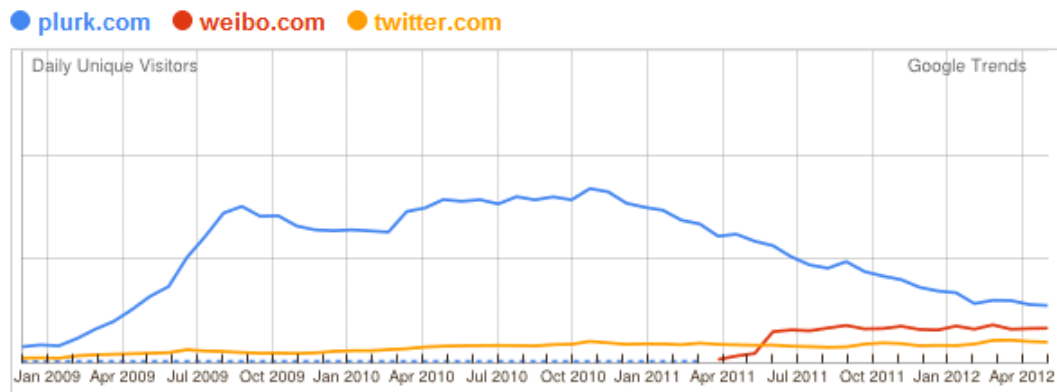


Figure 1.1: Plurk, Weibo, and Twitter daily visitors count graph (Taiwan only)

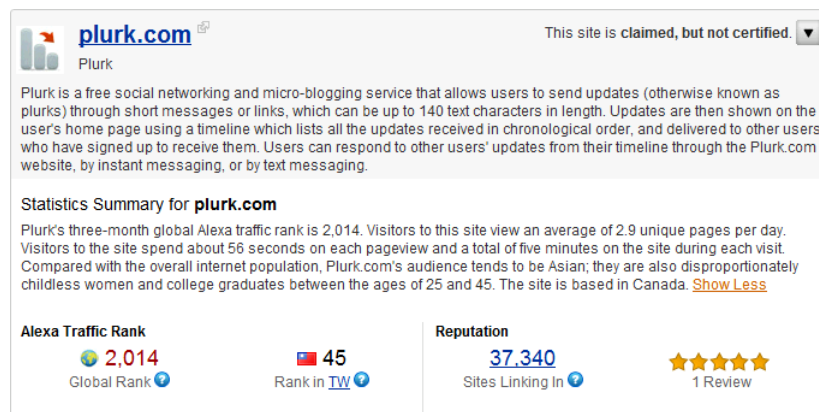


Figure 1.2: Alexa traffic rank for Plurk

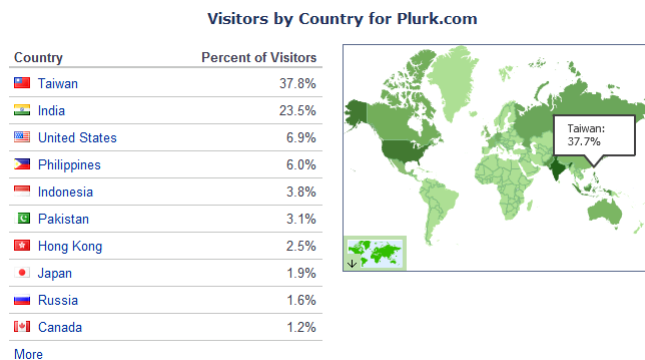


Figure 1.3: Visitors statistics by Alexa

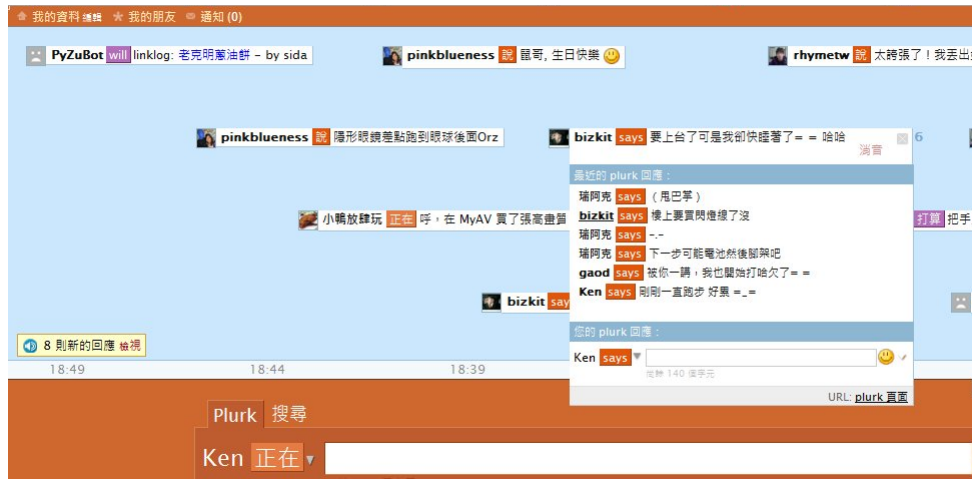


Figure 1.4: Plurk timeline

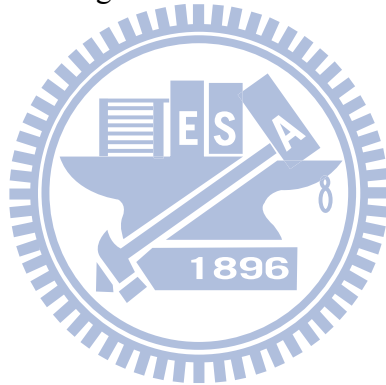


Figure 1.5: A sample plurk

(a) Additional information

(b) Interests

(c) Schools & Work

Figure 1.6: Plurk profile: extra information



Figure 1.7: A Plurk user with his plurks public



Figure 1.8: A Plurk user with his plurks private

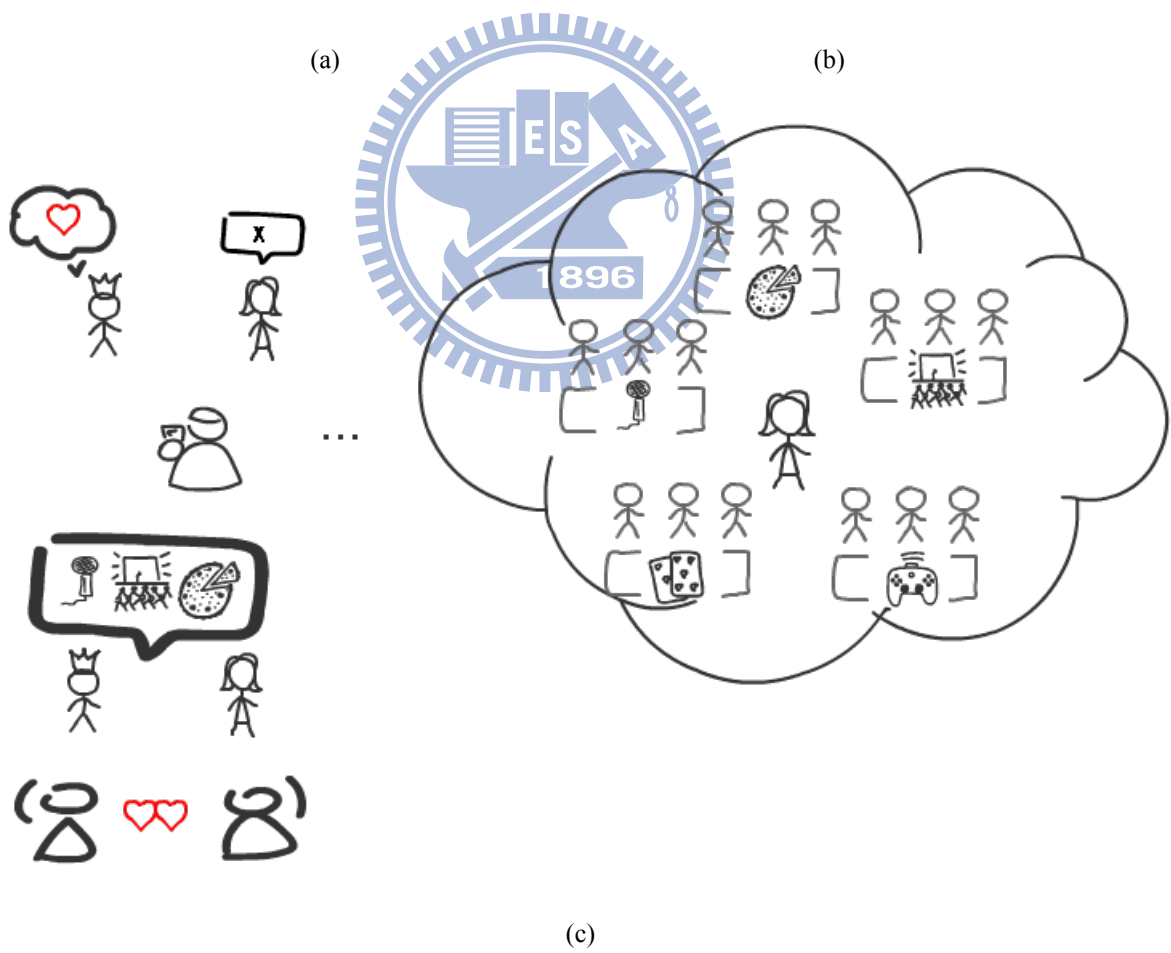
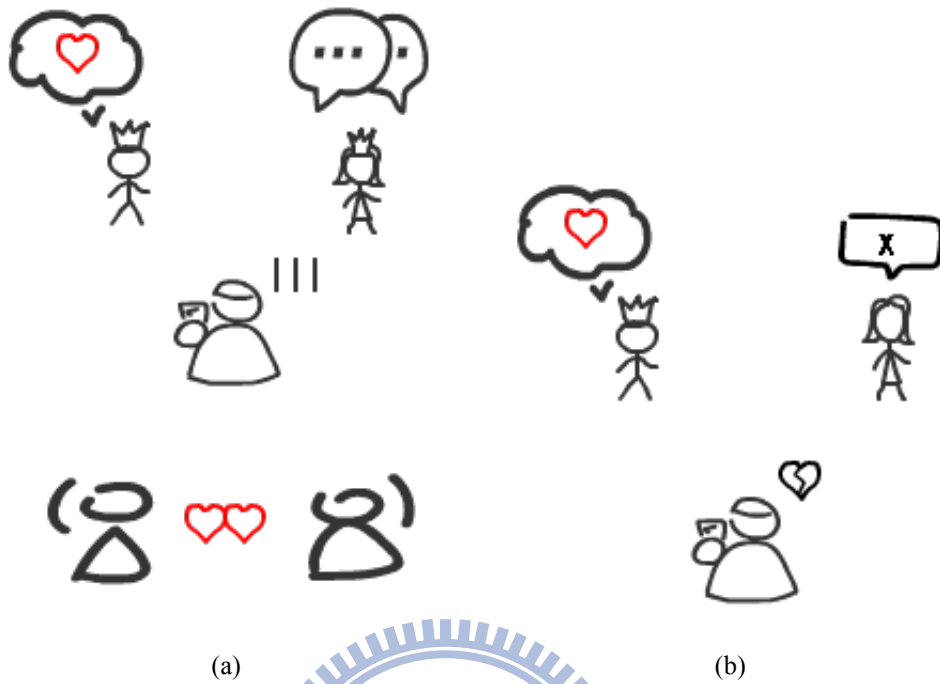


Figure 1.9: Interests derivation from public or private plurkers

1.2 Related Work

The Go!Plurk project [47], developed by Ken Lee, Bryan Cheng, and Sean Lee, is the first service to find users' interest topics based on the content they posted on Plurk. In this thesis, we extend and enhance the preliminary work of the Go!Plurk project.

Go!Plurk was announced via Plurk on June 15, 2009. There were at least 13,527 unduplicated users visited our website and we analyzed more than 30,000 Plurk accounts in the following week (Figure 1.10).

This project was reported by United Daily News (UDN) [95] and a famous blogger Briian [94] in June 2009, and PChome magazine also introduced the project in August of the same year.

We used 300 news articles from Yahoo! Taiwan and plurks from top-100 active plurkers as training sources, which were classified into ten pre-defined categories: chitchat, delicacies, education, lifestyle, movies, music, drama, sports, technology, and travel. Besides, we use CKIP [58] from Academia Sinica as Chinese tokenization engine, and defined a reserved lexical category list for filtering returned tokens from CKIP as shown in Figure 1.12 and Figure 1.13. Figure 1.11 depicts an overview of the Go!Plurk work flow.

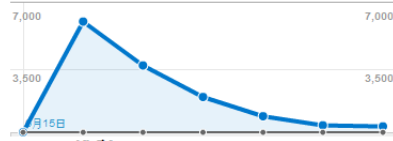
In a simple Go!Plurk test, we sampled 20 latest plurks from tester, use CKIP to extract Chinese tokens, apply filtered tokens into Naïve Bayes classifier to calculate scores for each category, and finally render a pie chart to visualize the interest distribution, as shown in Figure 1.14.

Although this service is popular with Plurk communities, there are several known issues and limitations which need to be improved. First, if the tester set his/her plurks private, we cannot get what he/she said in order to analyze his/her interest. Second, the quality and quantity of training articles are poor due to short training period and limited labor hours. Third, the ten pre-defined categories are not general enough to represent interests and users cannot get details due to the flat structure. Lastly, we only sample 20 latest plurks from tester via RSS feed provided by Plurk in order to simplify implementation; mechanism for handling plurk content with URL link was not implemented. As such, we cannot get enough tokens to represent the tester.

Given these issues and limitations, in this thesis we try to increase the accuracy of prediction results even if tester set his/her plurks private by collecting as much public plurks as we can to expand training scale, applying automatic training process, and deriving interests information from one's friends.

基準化

比較：類似規模的全部網站 [開啟類別清單](#)

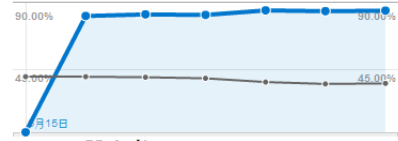


13,527 造訪

基準：3 (+450,800.00%)

同時請參閱 [所有流量來源](#) 報表，以取得此指標的其他分析。

2009/6/15 - 2009/6/21



84.47% 跳出率

基準：38.38% (+120.07%)

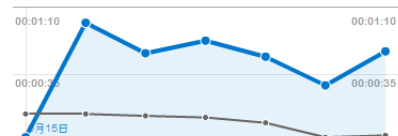
同時請參閱 [主要到達網頁](#) 報表，以取得此指標的其他分析。



16,994 瀏覽量

基準：5 (+339,780.00%)

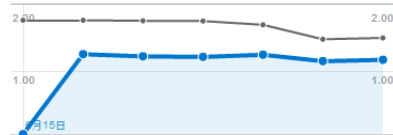
同時請參閱 [標題內容](#) 報表，以取得此指標的其他分析。



00:00:56 平均網站停留時間

基準：00:00:10 (+455.10%)

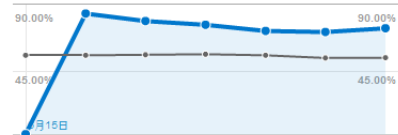
同時請參閱 [造訪時間長度](#) 報表，以取得此指標的其他分析。



1.26 單次造訪頁數

基準：1.76 (-28.56%)

同時請參閱 [造訪深度](#) 報表，以取得此指標的其他分析。



82.49% 新造訪

基準：56.57% (+45.82%)

同時請參閱 [訪客忠誠度](#) 報表，以取得此指標的其他分析。

Figure 1.10: Google Analytics for Go!Plurk

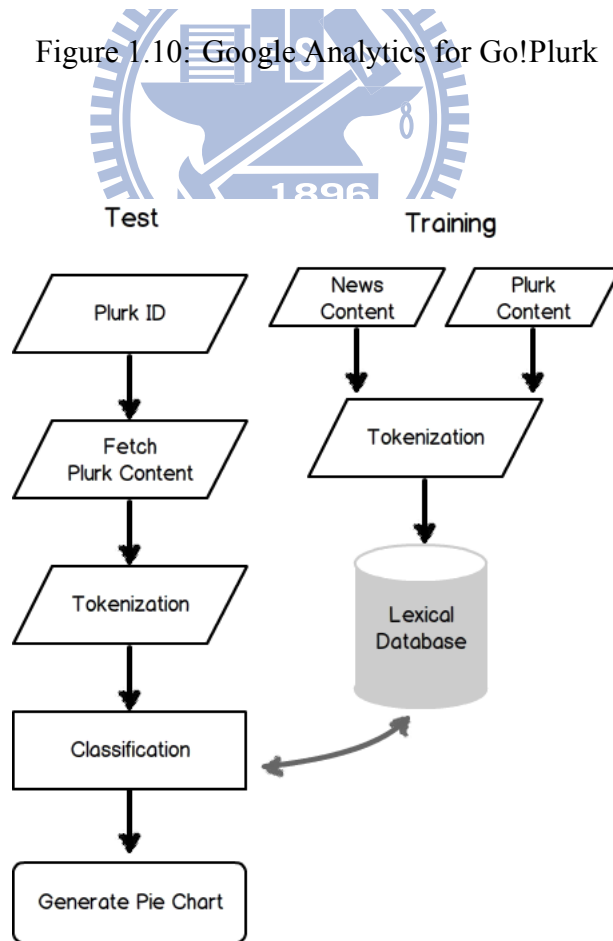


Figure 1.11: Go!Plurk system flow

Done back to index

工商時報【袁穎庭／台北報導】亞馬遜推出的電子書閱讀器掀起電子書熱潮，吸引相關硬體、內容、電信業者搶進。資策會MIC預料，電子書閱讀器開始進入成長期，更多競爭者加入供應鏈行列，電子書閱讀器市場的價格壓力也增加，預估每年價格將以5%的速度下滑。MIC指出，在亞馬遜、新力等大廠積極推動下，再加上全球環保節能趨勢抬頭，高喊無紙化的電子閱讀器市場受到重視。同時又有愈來愈多的數位閱讀內容出現，從一般的書籍、小說、進一步延伸到報紙、期刊、教科書，電子書閱讀器的基礎設施的建構和商業模式也日益成熟。在電子書營運模式方面，MIC認為未來會朝向以集團、或是結盟的方向發展，除了中華電信自有服務平台和內容之外，遠傳則是和誠品、三立合作，博客來則是與聯合線上攜手。產品尺寸方面，2006年全球電子書閱讀器都集中在6吋，不過電子書也有往大尺寸發展的趨勢，包括8吋、9吋級的產品都陸續問世。MIC預估，從2010年到2013年，全球6吋級電子書閱讀器產品的市佔率，將從70%逐步下降到42%。至於接近雜誌及教科書大小的9吋級電子書產品，市佔率將從18%，攀升到30%。至於更小、更便宜的5吋電子書閱讀器，以及10吋以上更大尺寸機種也都各有一定市場。根據MIC統計，去年全球電子書閱讀器市場總出貨量為110萬台，年增率120%，今年隨著Kindle熱賣、再加上多家品牌廠陸續推出電子書產品，預估今年全球電子書閱讀器市場出貨量將成長至305萬台，年成長率177%，近三年都有100%以上的成長率。2010年電子書閱讀器出貨量估計769萬台，2011年上看1,429萬台、年增85%，到了2013年全球規模達1,976萬台、年增12%。隨著愈來愈多電子紙、電子書供應商加入競爭行列，以及相關技術持續提升，電子紙及電子書閱讀器產品成本將逐步下滑，產品價格壓力將越來越大，預估每年價格將以5%的速度下滑。從產值來看，MIC統計去年全球電子書閱讀器產品的產值約2.77億美元，今年可望增至7.49億美元、年增170%，2010年全球電子書閱讀器產值將達18.17億美元，年增143%。隨著市場趨於成熟，預估到2012年電子書閱讀器產值將達39.99億美元，年增率只剩16.8%，2013年成長率只剩個位數。

斷詞之後的結果

工商時報 袁穎庭 台北報導 亞馬遜推出 電子書 閱讀器 掀起 電子書 熱潮 吸引 相關 硬體 內容 電信 業者 搶進 資策會 MIC 預料 電子書 閱讀器 進入 成長期 競爭者 加入 供應 鏈行 列 電子書 閱讀器 市場 價格 壓力 增加 預估 每 價格 速度 下滑 MIC 指出 亞馬遜 新力 等 大廠 積極 推動 加上 全球 環保 節能 趨勢 抬頭 高喊 無紙化 電子 閱讀器 市場 受到 重視 數位 閱讀 內容 出現 一般 書籍 小說 延伸 到 報紙 期刊 教科書 電子書 閱讀器 基礎 設施 建構 商業 模式 成熟 電子書 營運 模式 方面 MIC 認為 朝向 集團 結盟 方向 發展 中華 電信 自有 服務 平台 內容 遠傳 誠品 三立 合作 博客 聯合 線上 產品 尺寸 方面 全球 電子書 閱讀器 集中 電子書 往 大 尺寸 發展 趨勢 包括 6 吋 產品 問世 MIC 預估 全球 級 電子書 閱讀器 產品 市 佔 率 下 降 接近 雜誌 教科書 大小 級 電子書 產品 市 佔 率 攀 升 小 便宜 電子書 閱讀器 大 尺寸 機種 一定 市場 MIC 統計 全球 電子書 閱讀器 市場 總 出貨 量 為 年 增 率 120% Kindle 熱賣 加上 家 品牌 廠 推 出 電子書 產品 預估 全球 電子書 閱讀器 市場 出貨 量 成 長 率 近 成 長 率 177% 2010 年 電子書 閱讀器 出貨 量 估計 769 萬 台 2011 年 上看 1,429 萬 台、年 增 85%，到 了 2013 年 全球 規模 達 1,976 萬 台、年 增 12%。隨 著 愈 來 愈 多 電 子 紙、電 子 書 供 應 商 加 入 競 爭 行 列 相 關 技 術 提 升 電 子 紙 電 子 書 閱 讀 器 產 品 成 本 下 滑 產 品 價 格 壓 力 大 預 估 每 價 格 速 度 下 滑 產 值 看 MIC 統計 全 球 電 子 書 閱 讀 器 產 品 產 值 可 望 增 至 年 增 全 球 電 子 書 閱 讀 器 產 值 達 年 增 市 場 成 熟 預 估 電 子 書 閱 讀 器 產 值 達 年 增 率 剩 成 長 率 剩 個 位 數

刪除的字節

【】的、|、|、|、|，開始，更多，的也，|年將以5%的。|，|在、|下|，|再能，|的。|同時又有愈來愈的、|從的、|、|進一步、|、|的的的也日益。|在、|未來會以、|或是的，除了和之外，|則是和|，|來則是與攜手。|，|2006年都在|吋，|不過也有|的，|吋、|吋的都陸續。|，|從2010年到2013年，|吋的，|將從70%逐步到42%。|至於及的吋，|將從18%|，|到30%。|至於更、|更的吋，|以及10吋以上更也都各有。|根據，|去年110萬台，|120%|，|今年隨著、|再多陸續，|今年將至305萬台，|177%|，|三年都有100%以上的。|2010年769萬台，|2011年1,429萬台、|85%|，|了2013年1,976萬台、|12%。隨着愈來愈烈、|，|以及持續，|及|將逐步，|將以5%的。|從來，|去年的約、|17.7億美元、|170%|，|2010年將18.17億美元，|143%。隨着趨於，|到2012年將39.99億美元，|16.8%|，|2013年只。|

Figure 1.12: Sample news articles for training

Done back to index

使用者暱稱: bryanyuan2
對象網址: http://www.plurk.com/user/bryanyuan2.xml
噏文數量: 20

bryanyuan 覺得 禮拜六下午做自己的事+聽陳綺真的歌 = 是一種享受

bryanyuan 覺得 今天翁茲蔓報新聞好正!!!

- # 電影噏
- # 運動噏
- # 美食噏
- # 科技噏
- # 生活噏 bryanyuan 說 今天第一次嘗試用手動的刮鬍刀.感覺還是好恐怖的說~~
- # 亂聊噏
- # 學習噏
- # 音樂噏 bryanyuan 說 聽微軟的演講 (原來買office 會有 manual
- # 旅遊噏
- # 戲劇噏

bryanyuan 說 小睡一下.現在精神很好!!!

bryanyuan 說 晚上在院內運動.想說健身中心的教練怎麼這眼熟.原來是體能極限王的選手 (大驚!!!

bryanyuan 說 littlebtc: "你已經大四了" (阿阿阿~~

Figure 1.13: Sample plurks for training

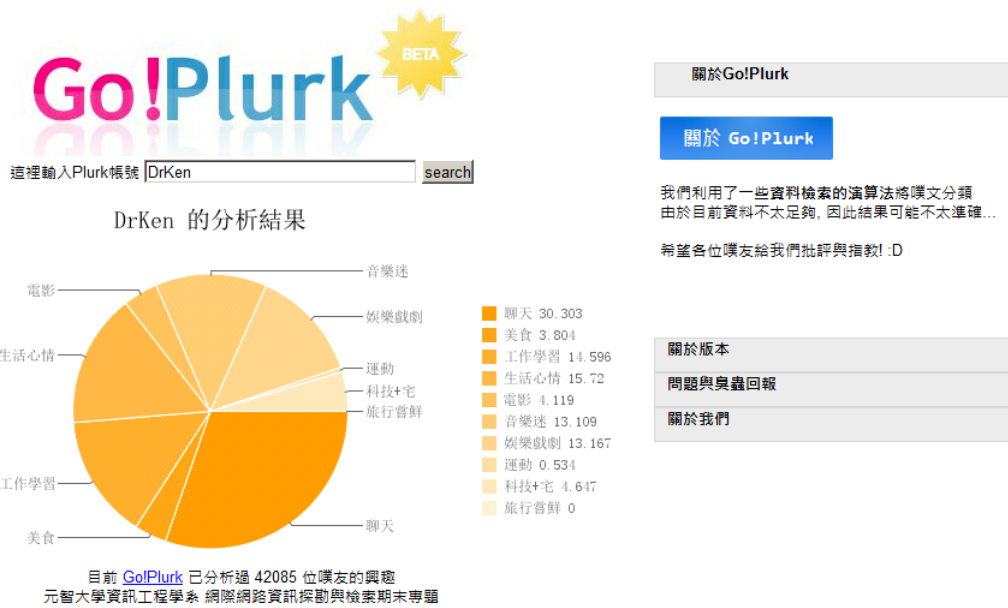


Figure 1.14: Interest pie chart generated by Go!Plurk

1.3 Challenges

In order to enhance and improve the Go!Plurk system, we have to collect plurk data back to local datastore efficiently, and the datastore must provide good durability and excellent reading performance for online retrieval. Traditional database management system (DBMS) is not suitable for managing big data; there are more than one billion of plurks to crawl. As such, we need to find out a database solution for big data, which is crucial to efficient web crawling.

Moreover, in order to derive a plurker's interest topics with his/her conversation private, we have to compute community partition information for the plurker and extract public plurks posted by the partition members to derive associated interest topics and keywords. Since the community detection problem is known for high computational complexity, we have to employ proper algorithm and optimize performance for online service.

1.4 Approach

1.4.1 Community Detection

Girvan and Newman [30] presented the Girvan-Newman algorithm for community detection by measuring the graph-theoretic measurement of betweenness. This algorithm returns reasonable quality of result but runs slowly in worst-case time $O(m^2n)$ on a network of n vertices and

m edges or $O(n^3)$ on a sparse network. The poor computational complexity makes it impractical for detecting communities in large networks.

Newman [60] proposed an enhanced community detection algorithm by employing modularity [59, 61] as objective function to maximize it. Modularity is a metric to measure the quality of a particular division of a network into communities. For a weighted network G , the modularity is defined as

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(C(i), C(j)),$$

where A_{ij} is the weight of edges between vertex i and j , m is the number of edges of G , $C(i)$ is the community of vertex i , and the δ -function $\delta(C(i), C(j))$ is equal to 1 if $C(i) = C(j)$, i.e. i and j are in the same community, and 0 otherwise.

The modularity maximization method employs exhaustive search for all possible divisions of a network for the highest modularity value to detect community and this method is considered intractable [14]. Newman [62] then proposed an approximate optimization algorithm which is similar to his previous research and the worst-case running time is $O((m+n)n)$ or $O(n^2)$ on a sparse network.

According to Fortunato [26], the computational complexity of Louvain algorithm [85] is $O(m)$. This algorithm is extremely fast and graphs with up to 10^9 edges can be analyzed in a reasonable time on current computational resources. Therefore, we use Louvain algorithm to detect community partitions in this thesis and the details about the Louvain algorithm is listed in the Section 3.3.2.

1.4.2 Data Collection

Chau [15] presented a framework which guarantees that no redundant crawling would occur while executing parallel crawlers for online social networks. He also demonstrates how to employ parallel crawlers and improve crawling performance for online social networks including LinkedIn and Friendster via centralized queue by using MySQL database as shown in Figure 1.15. The crawler architecture is based on two-tier parallelism, i.e. the coordinator or scheduler schedule tasks for multiple agents in parallel. Besides, each agent itself employs multiple threads for crawling. This architecture allows simultaneous failures of member crawlers. However, details of the protocol between crawler agent and scheduler, implementation of the crawler and datastore design for storing large number of records are not revealed.

Kwak [49] and Russell [74, 75] employ Twitter API to crawl Twitter data and demonstrated

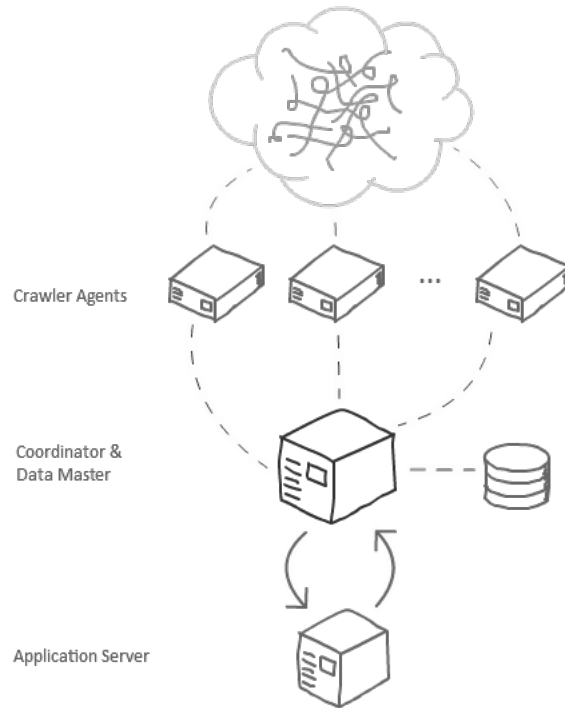



Figure 1.15: Architecture for two-tier parallel crawler

the basic usage with single threading. As the Twitter API has query rate limitation, Kwak employ 20 machines with different IPs and self-regulating collection rate at 10000 requests per hour. However, there are billions of tweets, millions of user profile and tens of billions of user relationship connections on the Twitter social network. We have to employ more efficient ways to crawl data from social network service provider.

1.5 Results

We build an online social networking service discovery (SNSD) system for Plurk users (plurkers) to find out interest topics/keywords and relationship . The results can be viewed on a website as shown in Figure 1.16. Besides, we develop a new distributed crawling system framework based on ZeroMQ messaging protocol and deploy it on several machines to crawl data from Plurk. Finally, we patch the Plurk API library for Python to enhance throughput by replacing the standard library with high-performance JSON library, optimize HTTP connections and customize Python C-extensions for accelerating HMAC-SHA1 [48] computation.

Social Search Engine Home TODO Contact About



Yi-Chien Lee (Ken)
24 歲, 男性

生日	1988-07-01
現居	Taipei, Taiwan
朋友 / 粉絲數	117 / 98
Karma	127.3

快速連結

- 🏠 前往他的個人資料頁面
- 👤 Profile
- 🔧 參數設定
- 👉 分享给网友

- 📖 Help
- 🗨️ 提供意見

興趣取向
交友小團體
Tag Cloud

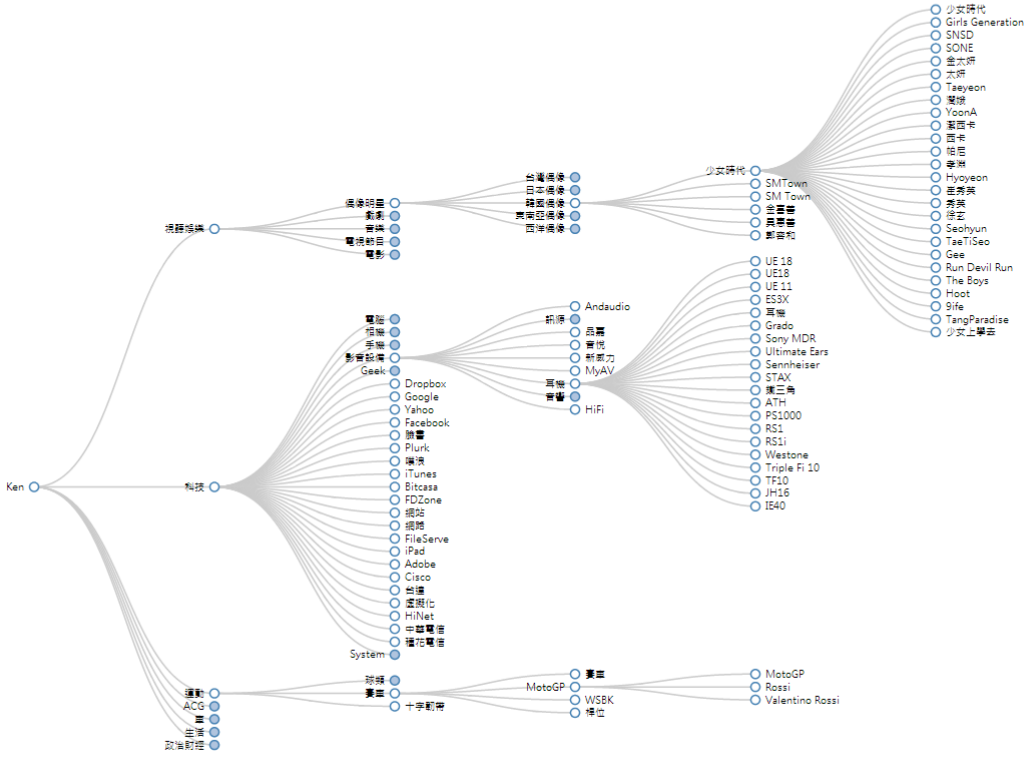


Figure 1.16: SNSD website overview

1.6 Thesis Structure

The remainder of the thesis is organized as follows. Chapter 2 introduces social networking services discovery (SNSD) system and the distributed crawling system for this thesis along with its high-level design. Chapter 3 describes system implementation details. Chapter 4 summarizes results of the implemented enhancements and illustrates the website built for visualizing SNSD system. Chapter 5 discusses future work and concludes the thesis.



Chapter 2

System Architecture

2.1 Overview

In this chapter, we will introduce two systems: (1) a social networking services discovery (SNSD) system for discovering user's relationship and interest from Plurk and (2) a distributed crawling system for crawling data from the Internet efficiently. The architecture diagram for these two systems is depicted in Figure 2.1.

2.1.1 Social Networking Service Discovery (SNSD) System

Recent studies [89, 49] indicate that micro-blogging services such as Twitter and Plurk are used as news aggregation services and ties in Facebook are driven by personal contacts. That is, networks may be clustered by communities of interests and geography is less significant for micro-blogging services. Offline relationships drive friendship in Facebook. As such, we can discover interest topics of users via community detection because micro-blogging services users connecting to each other are probably driven by interests instead of having offline relationships.

Given the hypothesis, we propose a framework to discover interest topics for a micro-blogging service user based on his/her conversations, even if conversations are private, by aggregating interest information from communities of the user. Figure 2.2 depicts an overall work flow for generation of interest keywords hierarchy.

2.1.2 Distributed Crawling System

Distributed crawling is a distributed computing technique employing many computers to fetch data from the Internet. For example, Internet search engines such as Google and Yahoo!

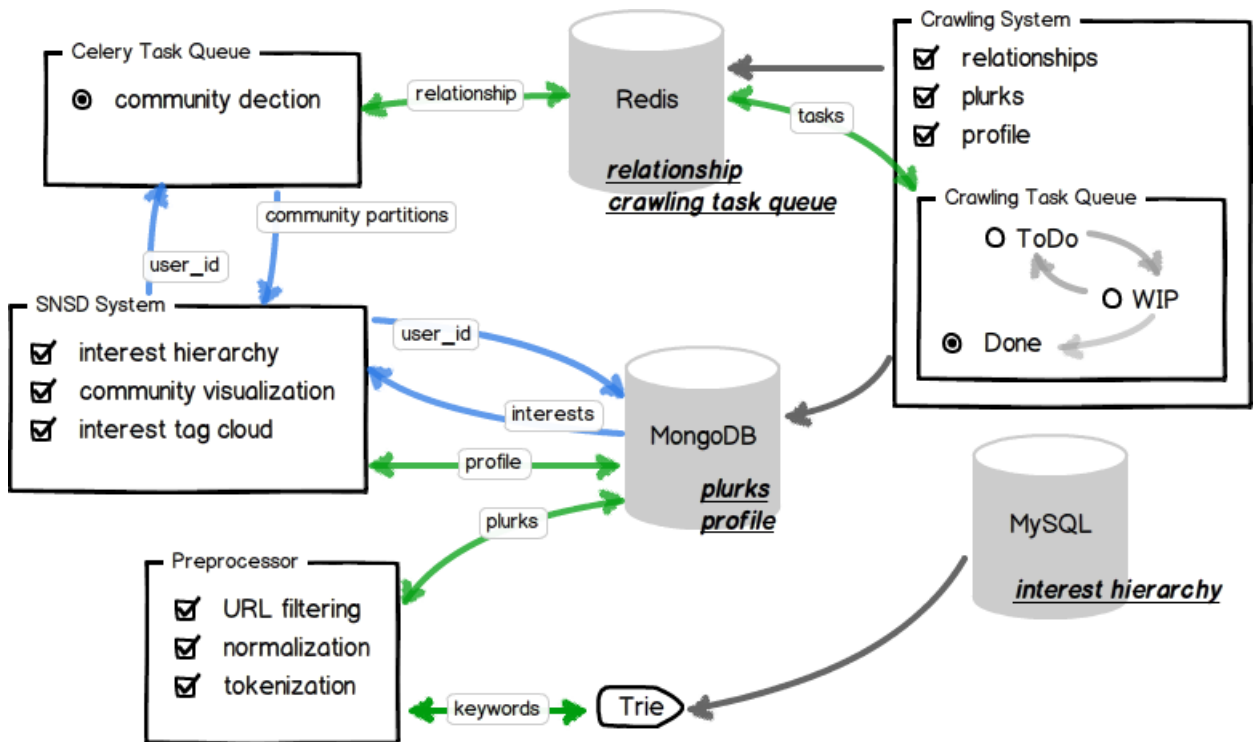


Figure 2.1: Components in the SNSD system and crawling system

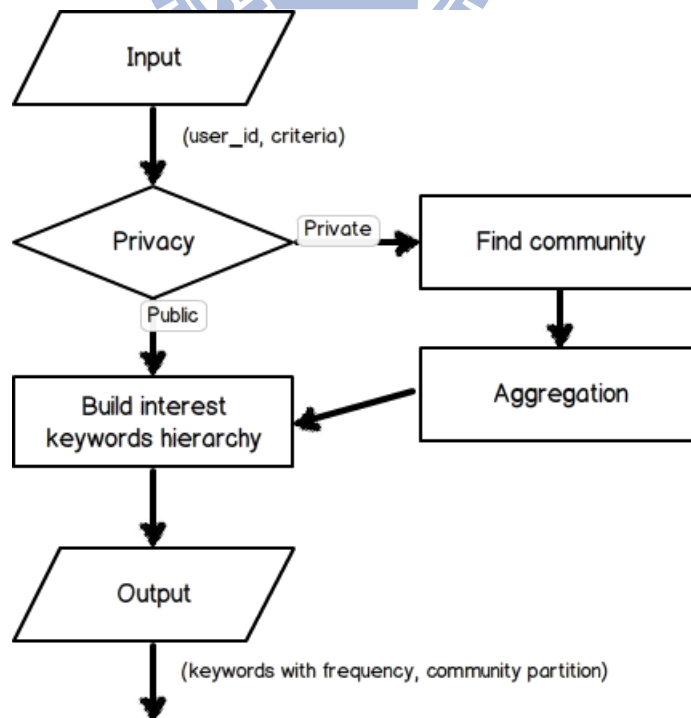


Figure 2.2: Work flow for generation of interest keywords hierarchy

built server farms distributed geographically to fetch web pages and build indices for indexing the Internet.

In this thesis, we deploy several computers as crawlers which call Plurk API to request Plurk users' profile, relationship and plurks. Besides, we utilize these crawlers as load balancers to fetch Uniform Resource Locator (URL) from plurks to extend content while avoiding blocking by the service provider.

Even though previous researches [15, 49] had proposed architectures for parallel crawlers, they did not provide implementation details such as protocol and datastore. Our design and implementation will be depicted later in this thesis.

2.2 SNSD System Design and Architecture

In this section, we will describe the framework which analyzes interest information for micro-blogging users even when their conversations are private.

Firstly, we try to collect users' conversations and relationship network as much as possible. In general, micro-blogging service providers (MBSPs) provide application programming interfaces (API) for users to access data; however, most of MBSPs limit the request rate by an API key or IP addresses. We will cover mechanisms for distributed crawlers to access data from MBSPs beyond the rate limitation in Chapter 3.

Secondly, we apply community detection algorithm when the requested user's conversations are private. According to previous hypothesis, micro-blogging networks are clustered by communities of interests. We use this idea to derive interest information from communities when conversation data for interest analysis is not available.

Thirdly, tokenize the incoming conversation and response data then apply syntactic filter for removing stop words and uncommon tokens. In western languages, words are separated by spaces in a sentence, so we only need to split the data by spaces and punctuation marks such as periods, commas, etc. for tokenization. But in Chinese, there is no simple ways to tokenize because Chinese text does not have word boundaries and each character is a fundamental linguistic unit. Therefore, we have to apply Chinese tokenization algorithms to tokenize data.

Lastly, merge interest tags and return them in a hierarchical structure by the pre-defined interest hierarchy generated from user's conversation or derived from communities. We get various interest tags from the previous step, but it is not suitable for visualization directly because they are still meaningless. Thus, we need to summarize distinct interest tags into formatted

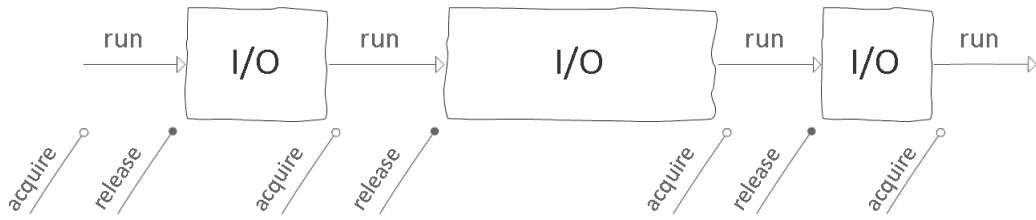


Figure 2.3: GIL Behavior

hierarchical structure so that users can view the results easily. Besides, if interest tags are derived from communities, it should render an additional community graph to indicate where these tags are derived from.

2.3 Crawling System Design Considerations

2.3.1 Concurrent Programming

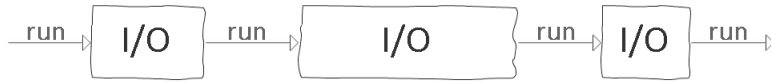
CPython, written in C, is the default Python bytecode interpreter. However, this interpreter is not fully thread-safe. In order to support concurrency, global interpreter lock (GIL), a mutex lock, was introduced. That is, only one thread is allowed to execute at a given moment, as shown in Figure 2.3. This restricts multi-threaded CPython programs from fully utilizing all processors in a multi-processor system. It becomes a computational bottleneck while processors are not fully utilized, as shown in Figure 2.4(a).

Therefore, for multiprocessing module, a process-based threading interface is available since CPython version 2.6 [40], and it side-steps the GIL effectively by using subprocesses instead of threads. Instead of threads, processes use interprocess communication (IPC) to communicate with each other, which is a much heavier solution.

The GIL is released on blocking I/O, when the thread is forced to wait, other threads in “ready” state will be chosen to execute and get into “running” state, as shown in Figures 2.4. Therefore, I/O bound Python programs are recommended to use threading module, and CPU bound programs fit better the multiprocessing module in general. Nevertheless, threading solution is not good enough in the C10K problem [46]. The C10K problem refers to handling of concurrent ten thousand connections. Several I/O models are introduced to achieve the goal as described below; We choose Gevent for this thesis.



(a) CPU Bound Tasks



(b) I/O Bound Tasks

Figure 2.4: Computation bound

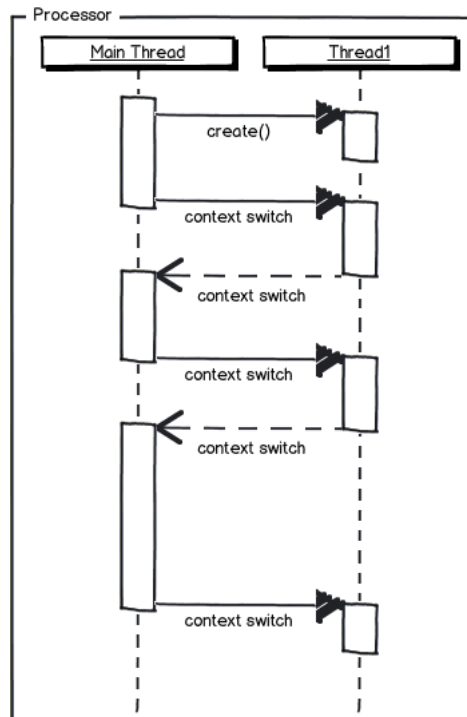


Figure 2.5: Thread

Blocking sockets with single thread

This model is the simplest implementation with one loop in one process, but it can only accept one connection at a time.

Blocking sockets with multi-thread

In order to accept multiple connections at the same time, this model will create a new thread to accept each connection request. Although it can deal with multiple connections, it is an inefficient approach because it will spend most of CPU time on context-switching when handling massive concurrent connections.

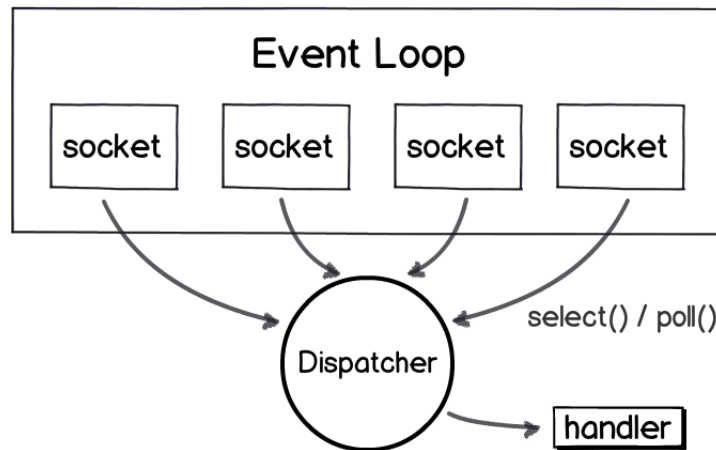


Figure 2.6: Event Loop

Non-blocking sockets by event-driven

In order to reduce the context-switching overhead, this approach creates a loop to wait for occurrence of I/O events and executes the registered handler associated with the event, as shown in Figure 2.6. This approach is also called event-driven programming. For example, Twisted [102] is a Python networking framework by using this approach to accomplish non-blocking asynchronous I/O. The main benefit of this approach is less context-switching, but it makes program complicated because multiple events might be raised simultaneously.

Non-blocking sockets by coroutine

Coroutine [11, 10, 50] is an alternative concurrency approach using Python generator function available since CPython version 2.5[32]. Unlike normal function, generator function produces sequence of results instead of a return value, and it yield a value then “throw” it back when called. In contrast to thread, coroutine does not use context-switching because all coroutines run in a single process, as shown in Figure 2.7. Besides, as coroutines are not run in multiple processes, they will not be restricted by GIL and we can fully control the scheduling of coroutines. Furthermore, it’s much cheaper to create a coroutine than a thread, we can spawn massive coroutines without significant overheads.

To improve the crawling performance, we finally choose Gevent [23], a coroutine-based networking library, as our crawling backend. Gevent uses a Greenlet [70], a micro-thread or lightweight coroutine library as the synchronous API on top of the libevent [63] event loop.

In Gevent programming model, every coroutine has a parent, i.e., the caller, and the top coroutine is the main thread or the current thread. Sub-coroutines yield execution to their parents

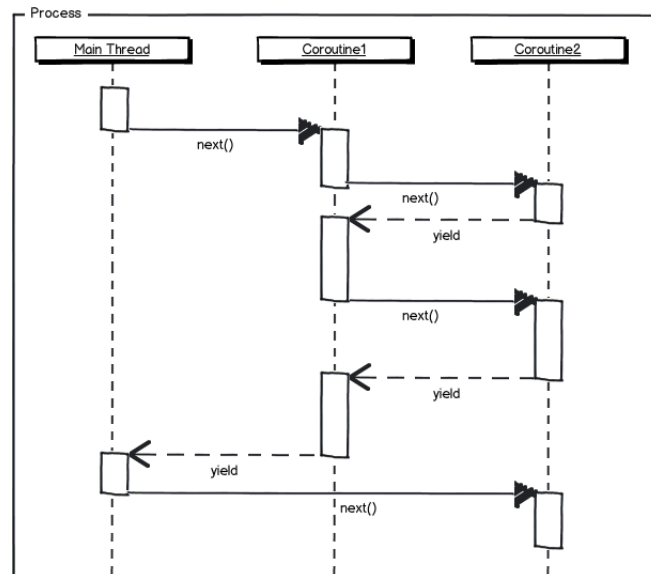


Figure 2.7: Coroutine

when starting to wait for completion of I/O operations, as shown in Figure 2.8. The parent coroutine will monitor which I/O is done from the event loop, and yield the execution back to the calling sub-coroutine to achieve the asynchronous non-blocking I/O operation, as shown in Figure 2.9.

Furthermore, Gevent provides a cooperative socket module which ensures coroutines by Greenlet can access sockets simultaneously. This feature, along with `urllib3` [5], is exploited to speed up the connection performance.

2.3.2 Messaging Protocol

Advanced Message Queuing Protocol (AMQP) [65] is an application layer protocol for message-oriented middleware (MOM) and is an evolution of semantics taken from the Java Messaging Service (JMS). AMQP covers two main enterprise messaging patterns: (1) topic-based publish-subscribe distribution and (2) reliable request-reply with persistent queues by pre-defined resources: exchange, queue, and binding.

ZeroMQ (ZMQ) [37] is an intelligent transport layer library of messaging functionalities inspired by the Internet Protocol (IP) [55]. It's a redesign of messaging to pursue the objective of uniformity and scalability, i.e. it aims to solve the problem of how to connect thousands of clients and do millions of messages in a second in a large messaging system. Furthermore, ZeroMQ covers four main patterns: transient pub-sub, unreliable request-reply, pipeline, and peer-to-peer. In addition, it provides broker devices and message routing when necessary.

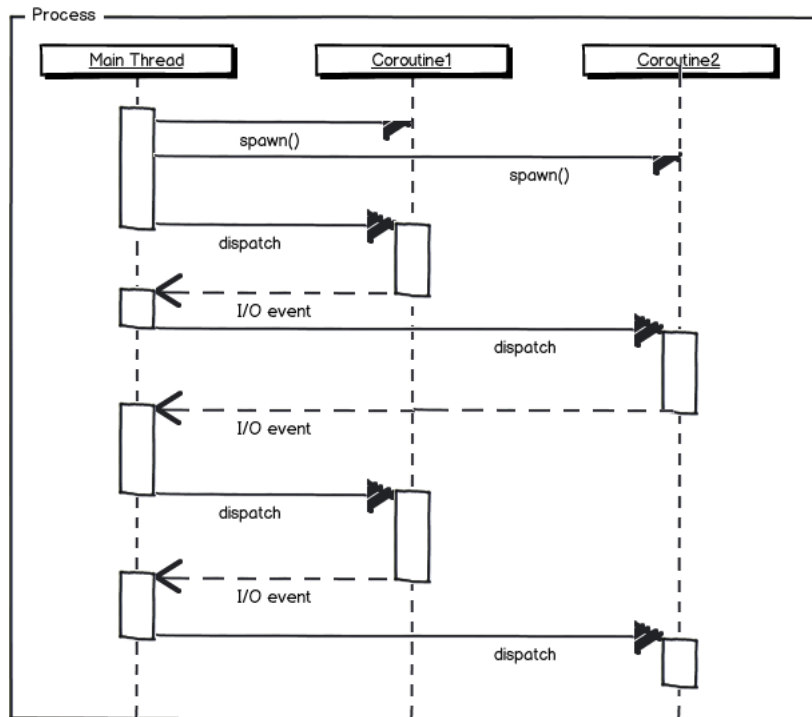


Figure 2.8: Coroutine with I/O

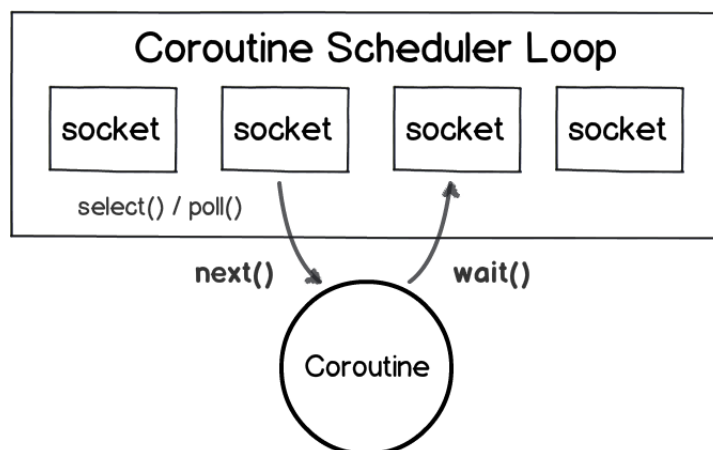


Figure 2.9: Coroutine with Event Loop

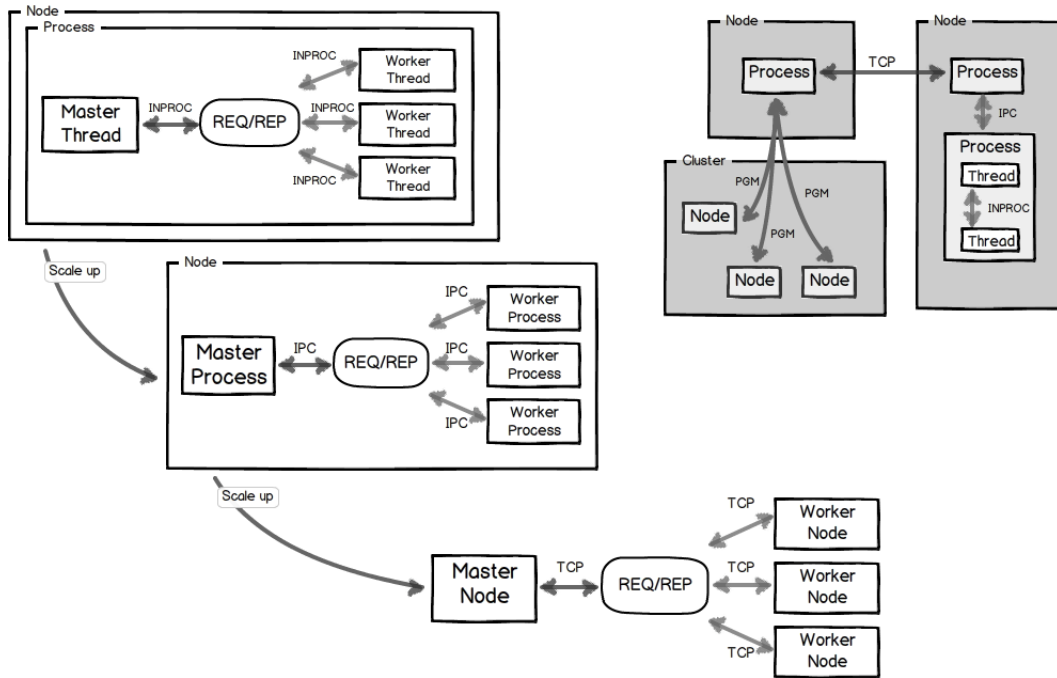


Figure 2.10: Transports of ZeroMQ

In general, AMQP is essentially centralized with a broker and provides reliable persistent queuing. ZeroMQ is essentially distributed with no pre-defined broker and aims at dealing with massive messages currently. We choose ZeroMQ as the crawling messaging framework and AMQP as the backend for Celery [8], the task queuing system for the web worker mentioned in the previous section.

Remote Procedure Call (RPC) over AMQP requires two queues for storing tasks data and result, as shown in Figure 2.11. Even though this scenario guarantees reliability and security, it has a high overhead in the queuing when employing it into a crawling system, as shown in Figure 2.12.

In order to reduce queue usage, we employ ZeroMQ as messaging protocol in the crawling system.

2.3.3 Data Serialization Format

In this section, we will introduce several data serialization formats employed in our systems: Pickle [33], JavaScript Object Notation (JSON) [43], Binary JSON (BSON) [96], and MessagePack (MsgPack) [76].

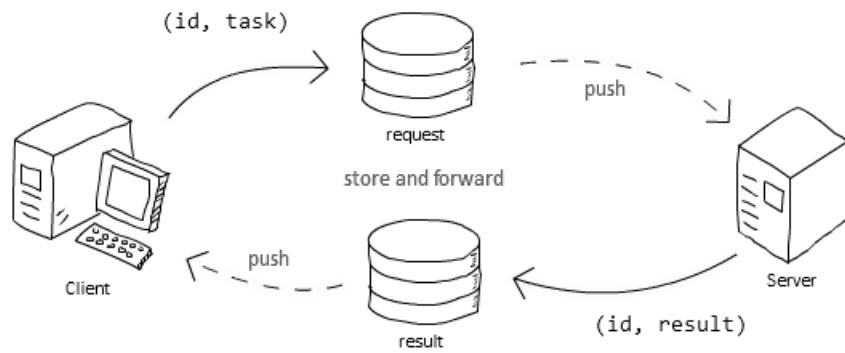


Figure 2.11: RPC over AMQP

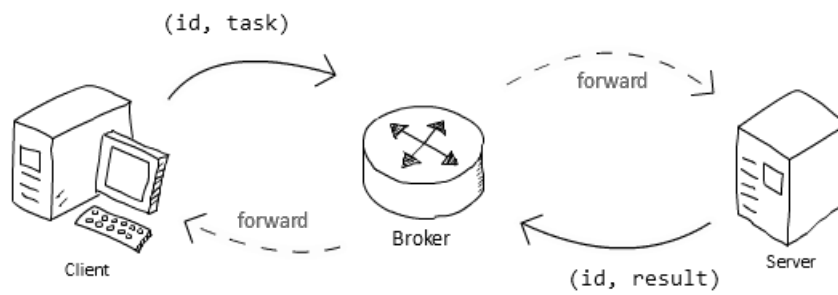
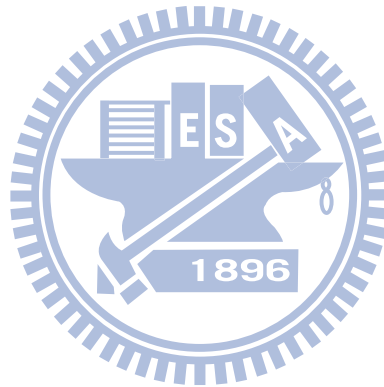


Figure 2.12: RPC over ZeroMQ

Pickle

Pickle is a standard Python module for serializing Python object structures. It converts a Python object into a byte stream when serializing; a byte stream is converted back into a Python object on de-serializing. However, the Pickle module is not intended to be a secure format against erroneous or maliciously constructed data. We need to authenticate the pickled object before de-serializing it.

JavaScript Object Notation (JSON)

JSON is a lightweight human-readable open standard [21] for data serialization. It is derived from JavaScript for representing data types and data structures. JSON is widely deployed by Web APIs such as Twitter, Plurk, and Facebook Graph API, etc.

Binary JSON (BSON)

BSON is based on JSON and is adopted by MongoDB for data storage. It is designed to be efficient both in storage space and scan-performance. Unlike JSON, BSON uses binary form for representing data types and data structures. In addition, it extends JSON with the date, byte array, and regular expression types.

MessagePack (MsgPack)

MsgPack is based on JSON and aims to be as compact and simple as possible. It is very similar to BSON except it does not support the date and regular expression data type but more space-efficient. The Protocol Buffers (PB) [31] format by Google Inc. also aims to be compact and is compared with MsgPack. However, it is necessary to define a schema which describes the structure for PB before serializing or de-serializing an object can be performed. But MsgPack and JSON are compatible to serialize arbitrary data structures.

Listing 2.1 demonstrates how to encode a dictionary object by the above four serialization formats in Python. The encoded data size for Pickle, JSON, BSON, and MsgPack are: 218, 164, 116, and 151 respectively.

Listing 2.1: Serialization

```
>>> import pickle, marshal, json, bson, msgpack
>>> data = {
...     "fans_count": 98,
...     "friends_count": 120,
```



```

...     "privacy": "only_friends",
...     "user_info": {
...         "display_name": "Ken",
...         "karma": 131.32,
...         "gender": 1,
...         "id": 3461880,
...         "avatar": 10
...     }
... }
>>> pickle.dumps(data)
"(dp0\nS'fans_count'\np1\nI98\nsS'user_info'\np2\n(dp3\nS'gender'\np4\nI1\nsS'display_name'\np5\nS'
Ken'\np6\nsS'karma'\np7\nF131.32\nsS'avatar'\np8\nI10\nsS'id'\np9\nI3461880\nssS'friends_count
'\np10\nI120\nsS'privacy'\np11\nS'only_friends'\np12\ns."
>>> json.dumps(data)
'{"fans_count": 98, "user_info": {"gender": 1, "display_name": "Ken", "karma": 131.32, "avatar":
10, "id": 3461880}, "friends_count": 120, "privacy": "only_friends"}'
>>> bson.BSON.encode(data)
'\x97\x00\x00\x00\x10fans_count\x00b\x00\x00\x00\x03user_info\x00J\x00\x00\x00\x10gender\x00\x01\
x00\x00\x00\x02display_name\x00\x04\x00\x00\x00Ken\x00\x01karma\x00\n\xd7\xa3p=j`@\x10avatar\
x00\n\x00\x00\x00\x10id\x00\xf8\xd24\x00\x00\x10friends_count\x00x\x00\x00\x00\x02privacy\x00\r
\x00\x00\x00only_friends\x00\x00'
>>> msgpack.dumps(data)
'\x84\xaafans_countb\xa9user_info\x85\xa6gender\x01\xacdisplay_name\xa3Ken\xa5karma\xcb@`j=p\xa3\
xd7\n\xa6avatar\n\xa2id\xce\x004\xd2\xf8\xadfriends_countx\xa7privacy\xaconly_friends'

```

In the crawling system, we decode JSON data from Plurk API then store the results into MongoDB in BSON format by MongoDB driver (PyMongo) [2]. Besides, scheduler transmits control signal and crawlers return crawled data to handler in MsgPack format via ZMQ, as shown in Figure 2.13. Furthermore, we return user profile and relationship data in JSON for AJAX HTTP requests and use Pickle as format for the Celery web task queue via AMQP in the SNSD system.

2.3.4 Datastore

In order to store as many conversations from Plurk as we can for the SNSD system, we have come up with the criteria for choosing proper data store: scalability, high availability (HA), performance and index support.

Scalability means we can easily scale out the data store by adding resources to a single node (scale vertically) or adding more nodes to the system (scale horizontally). High availability (HA) ensures the data store works properly even if a node in the system is down or out of service. Index support is required for improving performance and guarantee data uniqueness.

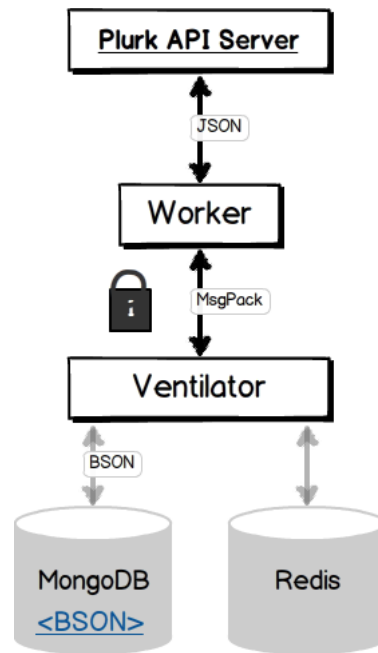


Figure 2.13: Various message formats within the crawling system

According to these criteria, we choose MongoDB, a document-oriented database system, as data store for storing conversations from MBSPs; MySQL, a relational database management system, for interest hierarchy; and Redis, a in-memory, key-value data store with optional durability, for storing user relationships.

2.3.5 Task Queuing for Crawling

Traditional task queuing systems based on Remote Procedure Call (RPC) require two extra queues to store task requests, as shown in Figure 2.14, and the result produced by workers for each request. However, it is not suitable for handling large number of requests by storing extra data for the traditional RPC.

In order to improve performance and storage efficiency, we replace AMQP with ZeroMQ library as messaging protocol, as shown in Figure 2.15, and introduce a new mechanism: let the worker pull tasks from ventilator (dispatcher) instead of having ventilator push tasks to available worker. Besides, ventilator maintains a priority queue to store states and creation timestamp for to-do tasks.

When a worker connects to ventilator and asks for a new task, the ventilator pop the oldest to-do task and check if the task is in done state and has exceeded the time to live (TTL) or not. If it is not done and exceeded TTL, return this task to worker and set it to the work-in-progress (WIP) state; otherwise, return the task in to-do state which is generated by ventilator. Figure

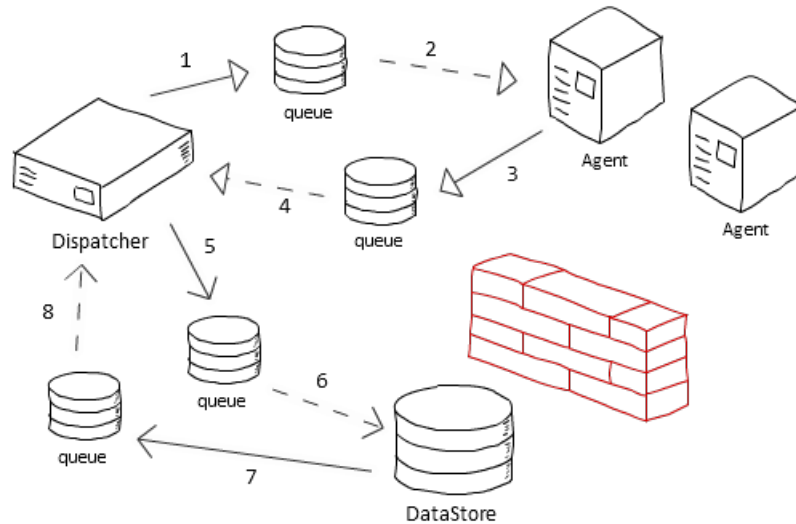


Figure 2.14: RPC over AMQP on Crawling

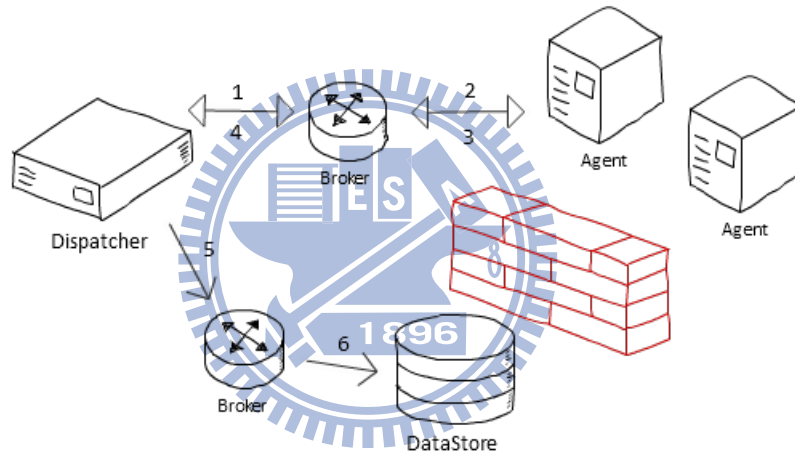


Figure 2.15: RPC over ZeroMQ on Crawling

2.16 illustrates the process mentioned above.

With this mechanism, ventilator can control the number of to-do tasks and guarantee all tasks will be processed eventually.

2.3.6 Security

We have to ensure communications between nodes in the crawling system are encrypted to prevent information leak or nodes being compromised. However, ZeroMQ does not provide encryption [97], therefore we need to implement key exchange protocol or use SSH tunnel.

Our crawling system is designed for handling massive requests with good performance. We cannot deploy complicated cryptographical mechanism such as RSA algorithm for per-message

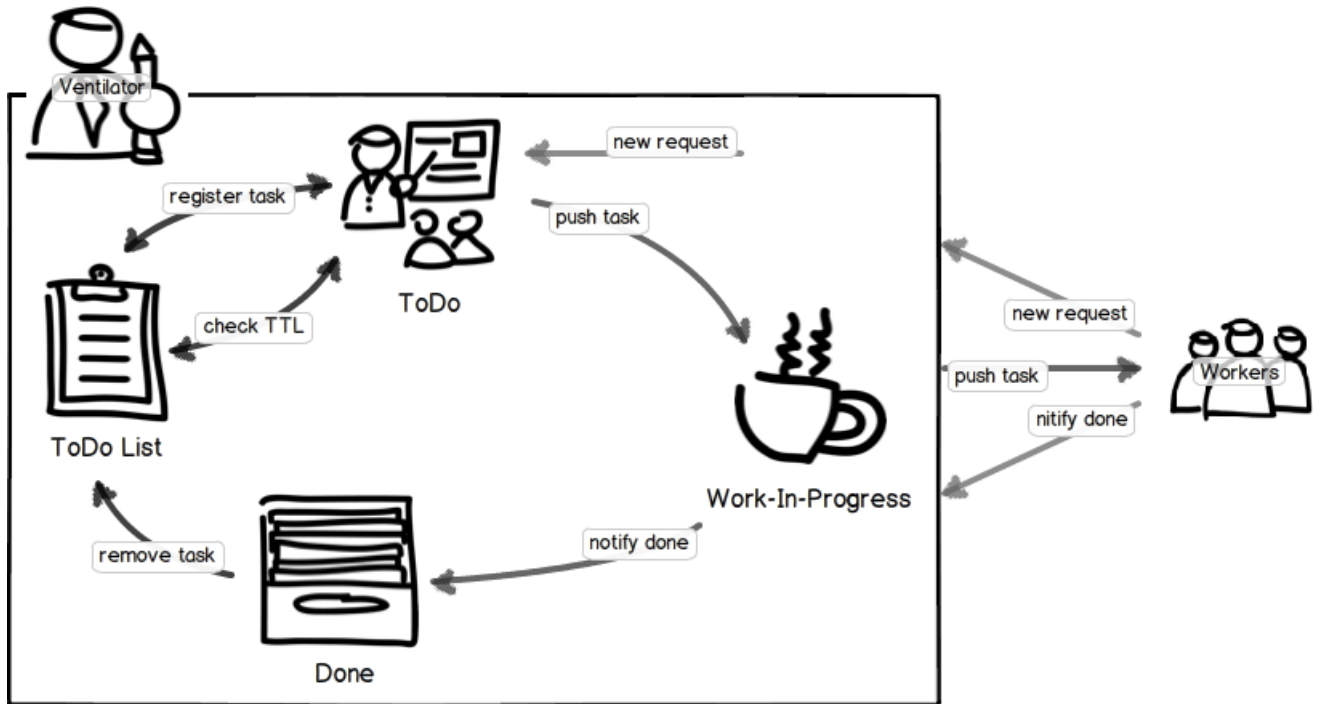


Figure 2.16: Queuing work flow

encryption/description and setup of SSH tunnel between ventilator and workers.

Therefore, we employ Advanced Encryption Standard (AES) algorithm with Intel Advanced Encryption Standard Instructions (AES-NI) hardware support as the default cryptography.

2.4 Distributed Crawling System Architecture

2.4.1 System Architecture

Similar to Chau's crawling framework [15], our system is also based on two-tier architecture to allow for simultaneous failures of agents. Figure 2.17 depicts a high-level architecture of the crawling system for this thesis. The crawling system consists of seven components as explained below.

- Agent: Installed in every worker node as a daemon process to receive commands from scheduler to start, stop, or restart the worker process, update scripts and configuration files, and increase/decrease the number of worker processes.
- Ventilator: Serves as the task dispatcher to dispatch tasks to available workers.
- Proxy: Started with the worker process in worker nodes. It is aimed to reduce TCP connections between backend ventilators.

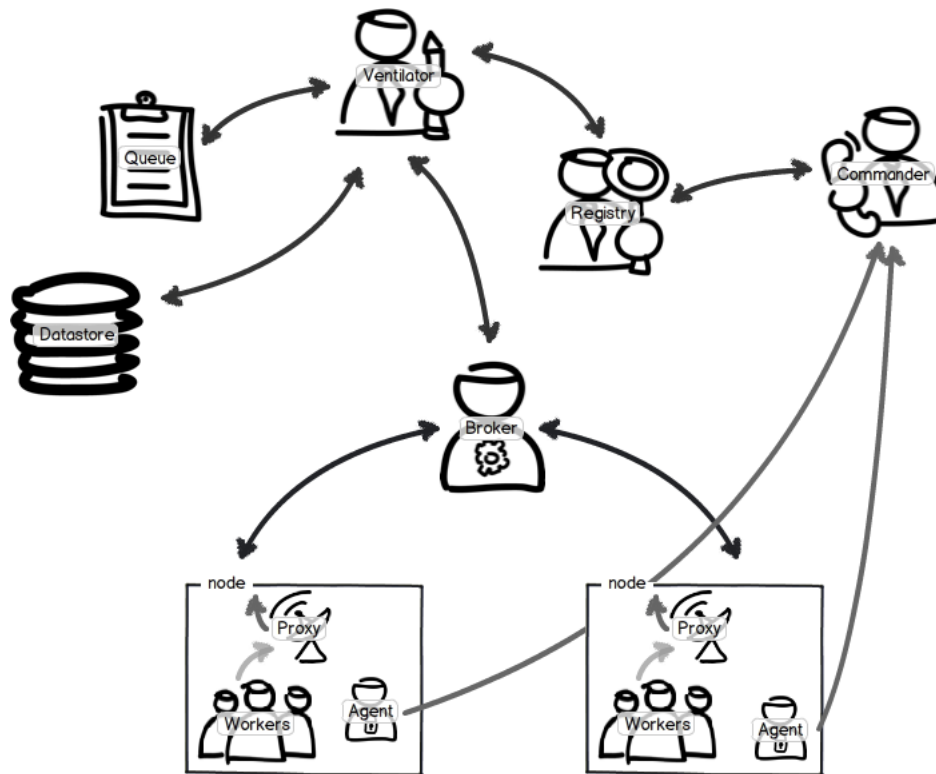


Figure 2.17: Distributed crawling architecture

- Broker: Similar to proxy, but it's started on server side to receive TCP connections from worker nodes and forward messages to ventilators by service identities as router.
- Worker: Do tasks assigned by ventilators.
- Registry: Keep track of available workers, allocate service identity for ventilators, and balance the requests from ventilators.
- Commander: Administrator send commands to control worker nodes via this role, and it could communicate with registry to adjust the total number of workers automatically.

There are several ventilators for different purposes in this system; each ventilator has a unique service identity which is allocated by the registry. For example, we want to crawl plurkers' relationship for community detection and public plurks for deriving interest topics. Then there will be two ventilators to dispatch tasks to workers and store result from crawler to specific datastore such as MongoDB and Redis in our scenario.

ZeroMQ covers several messaging patterns. We employ request-reply pattern between ventilator-worker, ventilator-registry, commander-registry, and commander-agent; publish-subscribe pattern between commander-agent. Figure 2.18 depicts the messaging pattern employed in the system. Detailed design is covered in the following section.

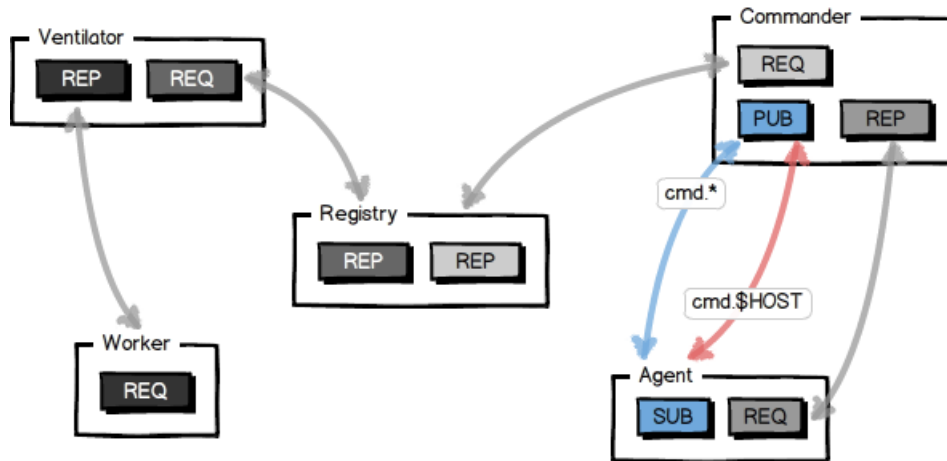


Figure 2.18: Messaging patterns between components

2.4.2 Work Flow

Figure 2.19(a) illustrates the work flow within crawling system. The flow is explained below. First, when worker becomes available, it sends an INIT message to ventilator via proxy and broker. If there are no available ventilators, worker will start to resend the INIT message until a ventilator responds.

After a worker association is established, ventilator updates worker status to registry and generates task for the worker with universally unique identifier (UUID) as task ID then sends task assignment to worker. When a worker finishes the task, it sends results along with task ID to ventilator. Ventilator then processes the results and stores it to the datastore.

Figure 2.19(b) illustrates the work flow between commander and agents. There are two messaging patterns, i.e. publish-subscribe and request-reply, between commander and agents in different scenarios. Agents subscribe to the topic with their own unique hostname and wait for specific instructions assigned by commander and broadcast generic topics.

If we want to broadcast instructions such as reboot all agent machines or fetch latest configuration files or assign specific agent to execute commands such as restart the crawling process, use the publish-subscribe channel with corresponding topic. That is, generic topic for broadcasting and specific topic for assigned agent. Moreover, if we want to execute commands and get response from agent, then use request-reply channel for receiving responses.

2.4.3 Task Queuing

As mentioned above, we introduce a new queuing mechanism and define three states to represent queuing status of a task. This mechanism is based on Redis datastore, which handles

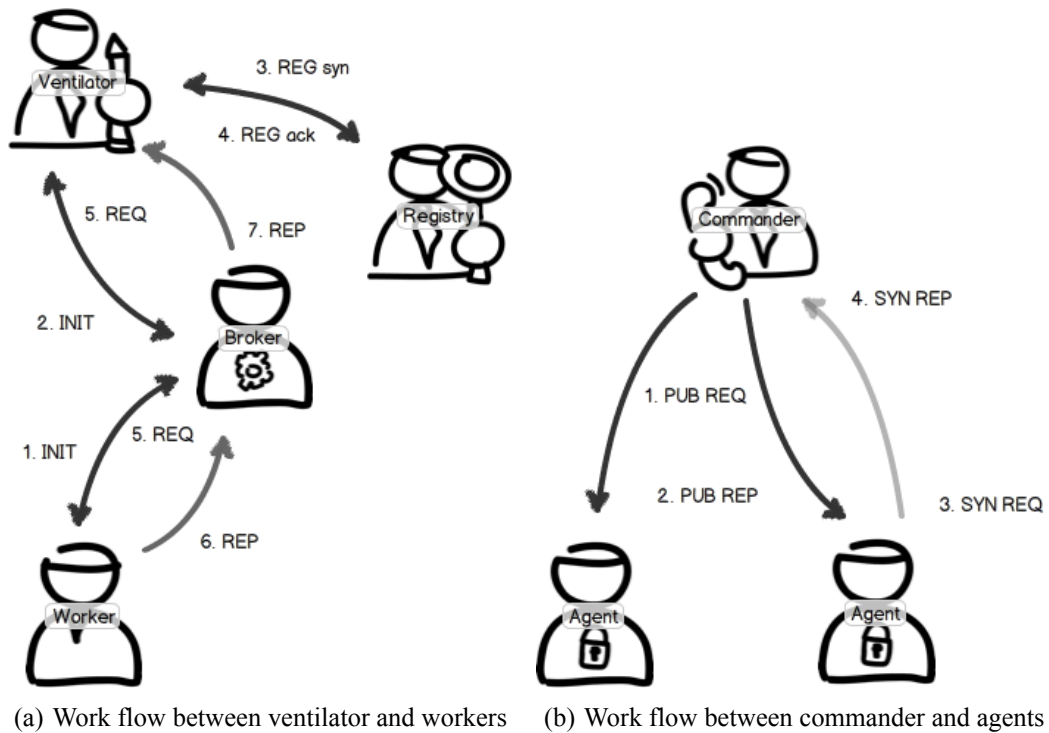


Figure 2.19: Work flow of crawling system

three data types: list, set and sorted set as shown in Figure 2.20.

Listing 2.2 illustrates how our queuing mechanism works. The execution function, upon receipt of a target `user_id`, will use `ZADD` command to add the target to the `WIP` queue with 300 seconds of `TTL` then crawl data for the target and store results into `datastore`. After crawling and storing, we add the target to the `DONE` queue by `SADD` command and remove it from `WIP` queue by `ZREM` command. If there is any exception during crawling or storing data, we remove the target from `WIP` queue by `ZREM` command.

The `fetch_targets` function demonstrates how to fetch new targets. First, we check if any target is in `WIP` state and has executed over `TTL` by `ZRANGEBYSCORE` command, i.e. there is something wrong while crawling data for the target. Second, we generate nine target candidates by removing and getting the first element in the `TODO` queue via `LPOP` command then return these candidates.

The `add_todo` function depicts how to add a given target `user_id` to the `TODO` queue. We check if this target is in the `WIP` queue by `ZSCORE` command and whether it is already done or not by `SISMEMBER` command first. If the target is not in the `WIP` state and not done, then push the target to `TODO` queue by `R PUSH` command.

Listing 2.2: Demonstration for queuing

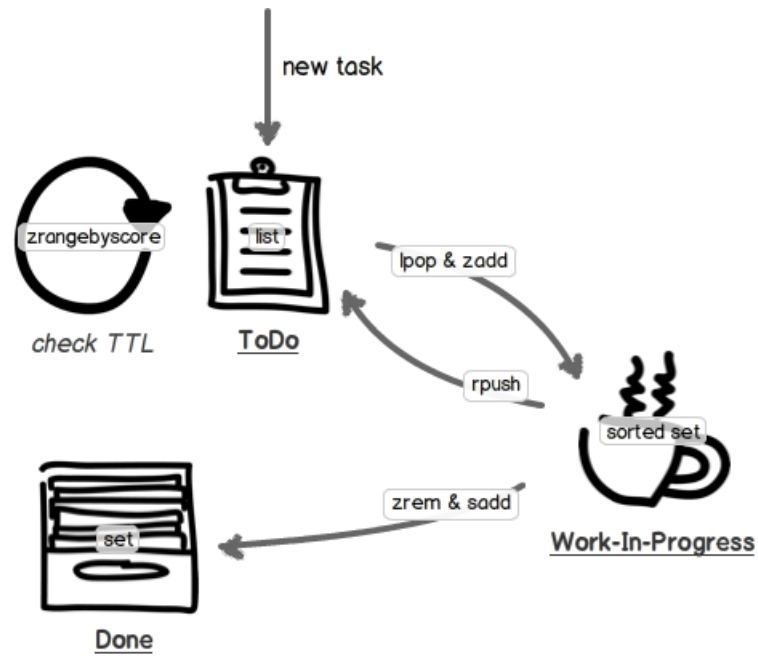


Figure 2.20: States and Redis data type of the task queue

```

import redis, time
r = redis.Redis()

def execute(user_id):
    try:
        r.zadd('WIP', user_id, int(time.time() + 300))
        CRAWL_FOR_THE_TARGET_AND_STORE(user_id)
        r.sadd('DONE', user_id)
        r.zrem('WIP', user_id)
    except:
        r.zrem('WIP', user_id)

def fetch_targets():
    targets = []
    for _ in r.zrangebyscore('WIP', 0, int(time.time())):
        targets.append(_)
    for _ in xrange(9):
        target = r.lpop('TODO')
        targets.append(target)
    return targets

def add_todo(user_id):
    if r.zscore('WIP', user_id) is None and not r.sismember('DONE', user_id):
        r.rpush('TODO', user_id)

```


Chapter 3

Implementation Details

3.1 Data Collection

3.1.1 Overview

In this section we will show how we crawl data from the Internet and how we store these data for interests derivation. There are three mechanisms for crawling data from the Internet and Plurk: (1) parsing HyperText Markup Language (HTML) source; (2) applying stateful programmatic web browsing module or (3) using application programming interface (API) provided by service provider.

Parsing HTML source is the basic mechanism for web crawling. It works by analyzing static pages' HTML source code with regular expressions (Regex) or creating Document Object Model (DOM) for parsing. However, this mechanism is unable to process a page whose content is loaded with Asynchronous JavaScript and XML (AJAX). For example, we can apply this mechanism to crawl Plurk in mobile view (Figure 3.1), but it doesn't work in the standard view.

In order to deal with AJAX, we utilize the stateful programmatic web browsing module. Generally speaking, this mechanism is based on web browser engines such as WebKit and Gecko to interpret web pages as a real web browser. Even though this mechanism can deal with most of web pages, it is much slower than parsing HTML directly. It not suitable for crawling a large number of web pages due to poor performance.

Most of web service and social network service providers such as Google, Twitter, and Plurk, etc. provide APIs for developers to access data by registering applications to the official registry. This mechanism is the most efficient way for crawling data from specific service. However it



Figure 3.1: Plurk mobile view

usually has rate limitation, i.e. only a limited number of requests in a given period of time is allowed. Besides, it can't work behind web proxy servers as anonymous page view.

We apply Plurk API for crawling Plurk data and use Spynner, a stateful programmatic web browsing module for Python, as the engine to parse keywords from Google real time trends service in this thesis.

3.1.2 Plurk API and Library

Plurk API [67] is currently available in version 2.0. Compared to version 1.0, version 2.0 is stateless (no login is required) and requests should be signed using OAuth Core 1.0a standard [64]. Version 1.0 is session-based and user account and password, instead of authorized keys, are used for authentication. Both Version 2.0 and 1.0 API return data encoded in JSON format.

Plurk officially recommends clsung's plurk-oauth [17] API library to Python developer, which depends on oauth2 [79] and httplib2 [83] library. Listing 3.1 depicts how to use plurk-oauth library to get Plurk profile. Even though the plurk-oauth is fully functional and well tested, it has poor performance and connection latency resulting from: HTTP connection overhead, performance bottlenecks in JSON library decoding and HMAC-SHA1 signing.

The HMAC procedure for OAuth consists of two phases: (1) calculate HMAC signature by the specified hash function and the given key and message, then (2) compute the Base64 encoding for the given binary signature. Listing ?? demonstrates the Python implementation of HMAC-SHA1. Besides, HMAC-SHA1 signature can also be obtained by the shell commands as follows:

```
$ echo -n "message" | openssl dgst -sha1 -binary -hmac "key" | openssl enc -base64
IIjfdNXyFGtIFGyvSWU3fp0L46Q=
```

Listing 3.1: Get Plurk profile by plurk-oauth library

```

>>> from PlurkAPI import PlurkAPI
>>> plurk = PlurkAPI(CONSUMER_KEY, CONSUMER_SECRET)
>>> plurk.authorize()
>>> print plurk.callAPI('/APP/Profile/getOwnProfile')
...truncated data...

```

Listing 3.2: Compute HMAC-SHA1 by Python standard libraries

```

>>> import hashlib
>>> import binascii
>>> trans_5C = ''.join(chr(x ^ 0x5c) for x in xrange(256))
>>> trans_36 = ''.join(chr(x ^ 0x36) for x in xrange(256))
>>> digestmod = hashlib.sha1
>>> blocksize = digestmod().block_size
>>> def hmac(key, msg):
...     if len(key) > blocksize:
...         key = digestmod(key).digest()
...     key += chr(0) * (blocksize - len(key))
...     o_key_pad = key.translate(trans_5C)
...     i_key_pad = key.translate(trans_36)
...     return digestmod(o_key_pad + digestmod(i_key_pad + msg).digest())
...
>>> h = hmac('key', 'message')
>>> print h.hexdigest()
2088df74d5f2146b48146caf4965377e9d0be3a4
>>> print binascii.b2a_base64(h.digest())[:-1]896
IIjfdNXyFGtIFGyvSWU3fp0L46Q=
>>>
>>> from hashlib import sha1
>>> import hmac
>>> h = hmac.new('key', 'message', sha1)
>>> print binascii.b2a_base64(h.digest())[:-1]
IIjfdNXyFGtIFGyvSWU3fp0L46Q=

```

3.1.3 Library Optimization

In order to improve these performance bottlenecks, we develop our enhanced patch for plurk-
oauth library.

First, we replace httplib2 [83] with urllib3 [5] for connection pooling; instead of making
connection for each request, connection pool works as a cache to make connections reused when
required, as shown in Figure 3.2. This reduces connection latency and improves throughput.

Second, as Plurk API returns data in JSON format and every request must be decoded into
dictionary type for Python or hash type for Ruby, this is one of the performance bottlenecks.

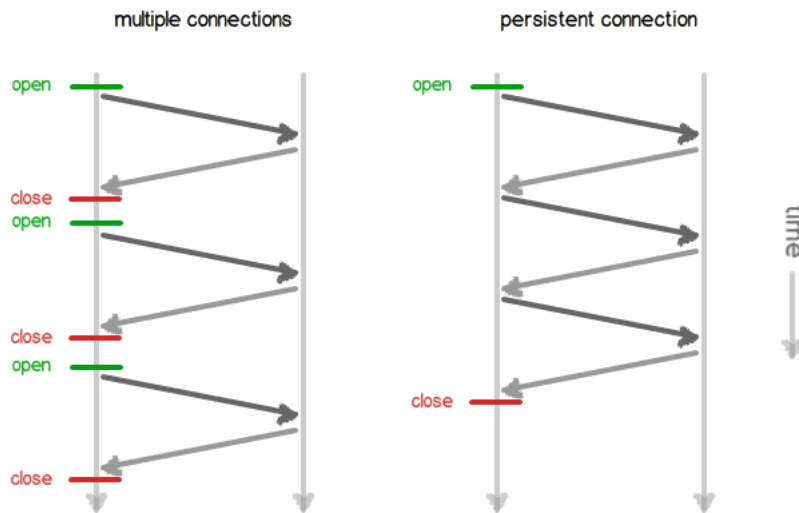


Figure 3.2: HTTP persistent connection

We benchmark and profile several Python JSON libraries (mentioned in Chapter 4) then replace Python JSON decoder included in standard library with `ujson` [42], which is a high performance C extension module for Python for the enhanced library.

Third, Python `hmac` module included in standard library is based on `hashlib` module, which calls native and optimized OpenSSL binary directly. However, this approach has poor performance because `hmac` module calls `hashlib` module just for getting hashed value and process several steps such as character translating and Base64 encoding for calculating HMACs. To address this issue, we customize an OpenSSL wrapper for HMAC-SHA1, which returns complete HMAC-SHA1 value directly.

OpenSSL is a nearly optimized C library by assembling codes with hardware acceleration instructions, and it provides several ciphers, hashing and encoding functions. We use OpenSSL as the HMAC, SHA1, and Base64 engine and integrate these OpenSSL functions to a Python extension module with Python C API. This customized extension is built with native codes. It performs 72 times faster than the version included in standard library. Detailed experimental results are given in Chapter 4.

Listing 3.3: Compute HMAC-SHA1 by OpenSSL

```
#include <string.h>
#include <openssl/evp.h>
#include <openssl/hmac.h>
#include <openssl/sha.h>

#ifdef SHA_DIGEST_LENGTH
#define SHA_DIGEST_LENGTH 20
```

```

#endif
#define B64_LEN (((SHA_DIGEST_LENGTH + 2) / 3) * 4) + 1

unsigned char* hmac_sha1(unsigned char* key, unsigned char* data)
{
    unsigned char* digest;
    unsigned char ret[B64_LEN];
    // compute HMAC digest
    digest = HMAC(EVP_sha1(), key, strlen(key), data, strlen(data), NULL, NULL);
    // encode binary digest to Base64 format
    EVP_EncodeBlock(ret, digest, SHA_DIGEST_LENGTH);
    return ret;
}

```

3.2 Preprocessing

In this section, we will show the elements of Plurk content (plurk) and explain how we apply the URL filtering and tokenization to the preprocessing of plurks.

3.2.1 A Plurk and Its Data

We can invoke Timeline series API to fetch plurk data. For example, we invoke `/APP/Timeline/getPlurk` to get data for a plurk by passing the plurk unique id or invoke `/APP/Timeline/getPublicPlurks` to get the public plurks for a plurker by passing the plurker's `user_id` or nickname. A plurk (Figure 1.5 on Page 4) is encoded as a JSON object. It will be returned as follows:

```

{
  "responses_seen": 0,
  "qualifier": "says",
  "replurkers": [],
  "plurk_id": 1003643246,
  "response_count": 0,
  "replurkers_count": 0,
  "replurkable": false,
  "limited_to": "|3461880|",
  "no_comments": 0,
  "favorite_count": 0,
  "is_unread": 0,
  "lang": "tr_ch",
  "favorers": [],
  "content_raw": "http://j.mp/JoIb2K\nhttp://youtu.be/C8HjWFPY78I\n少女時代太妍、蒂芬妮、徐玄所組成的子團體「太蒂徐」以首張專輯《Twinkle》登台已滿四周，每周都穿上不同風格的表演服的她們，26日以性感可愛的粉色系空",

```

```

"user_id": 3461880,
"plurk_type": 1,
"qualifier_translated": "說",
"replurked": false,
"favorite": false,
"content": ...truncated data...,
"replurker_id": null,
"posted": "Sun, 10 Jun 2012 15:50:08 GMT",
"owner_id": 3461880
}

```

Plurk API defines twenty two attributes for a plurk. However, in order to reduce storage size, we only include the following eight essential attributes for further processing in this thesis.

The definitions of each attributes are listed below:

```

{
  "_id": Number,
  "owner": Number,
  "qualifier": String,
  "content": String,
  "content_raw": String,
  "tags": Array,
  "posted_at": ISODate,
  "updated_at": ISODate
}

```

- `_id`: The unique plurk id, used for identification of the plurk.
- `owner`: The owner/poster of this plurk.
- `qualifier`: Qualifier is used to define the type of the plurk, which can be “says”, “asks”, “likes”, “shares”, etc.
- `content`: The formatted and filtered content, e.g. URL will be turned into text and emotions will be filtered.
- `content_raw`: The raw content as entered by user.
- `tags`: The tagging result from the filtered content, which is listed in the interest hierarchy.
- `posted_at`: The date this plurk was posted in ISODate format.
- `updated_at`: The date this plurk was formatted and filtered in ISODate format.

3.2.2 Elements of a Plurk

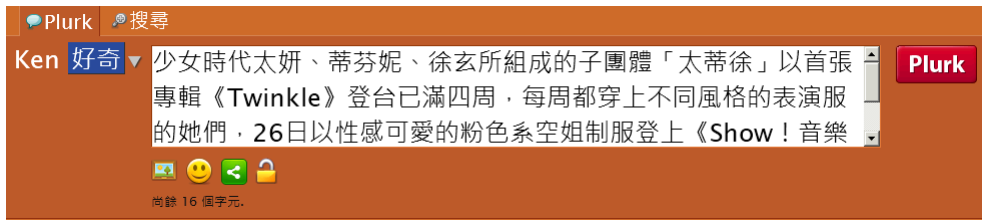
A plurk is composed of the following elements:

- Text: Text is the basic type of a plurk, which may contain Chinese, Japanese, English, or other language characters stored in Unicode.
- URL: URL may be in several types: @plurk_ID, web page, image, or file.
 - @plurk_ID: @plurk_ID identifies a Plurk user (plurker). A @plurk_ID in a plurk will be stored in the content_raw column and transformed into http://www.plurk.com/plurk_ID in the content column. Moreover, it will show the plurker’s nickname instead of account name as the link name in the Plurk page.
 - Web Page: The web page type is a hyperlink which refers to a web page. Plurk user can define the link name; if not defined, it will show the original link in the Plurk page.
 - Image: The image type is a hyperlink which refers to an image in such format as PNG, JPG, GIF etc.
 - File: The file type is similar to the image type. If the hyperlink does not refer to an image then a normal file is assumed.
- oPreview: oPreview is a special case of web page type. If the page has open graph protocol properties, the hyperlink will be transformed into a short “summary” instead of normal links. This type is convenient for plurkers to share a web page. Instead of typing URL and defining the link name, a plurker simply types URL and the page title will be displayed automatically.

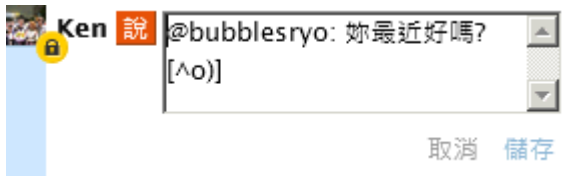
Based on the characteristics of plurk elements, we design a URL filtering mechanism and a preprocessing procedure which will be described in the following sections.

3.2.3 URL Filtering Mechanism

We give a procedure for URL filtering in order to transform URL from raw link into text content or tags which represent the subject of the URL. The pseudocode is shown in Algorithm 1.



(a) Text



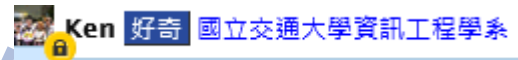
(b) @plurk_ID source



(c) @plurk_ID rendered



(d) Web page URL source



(e) Web page URL rendered



(f) Image source



(g) Image rendered



(h) File URL source



(i) oPreview source



(j) oPreview rendered

Figure 3.3: Elements of a plurk


```

Input: URL
Output: content

1 begin
2   content = null
3   if URL is shortened then
4     | URL = expand_shortened_URL(URL)
5   end
6   if URL is a web page then
7     | if Tag is available then
8       | content = keywords from predefined tags column
9     | end
10    | else if Metadata is available then
11      | content = keywords or description from metadata
12    | end
13    | else
14      | content = title of the page
15    | end
16  | end
17  | else if URL is an image then
18    | content = keywords from Google image search
19  | end
20  | else if URL is linked to Youtube then
21    | content = keywords from metadata
22  | end
23  | return content
24 end

```

Algorithm 1: URL filtering mechanism

Firstly, extract the original URL behind the short URL if necessary by detecting if any URL redirect request occurs while reading a URL. For example, the URL <http://www.ettoday.net/news/20120527/50150.htm> is shorten into <http://j.mp/JoIb2K> and posted to a plurk. In this case, we will detect the URL redirect when open the shortened URL, we then continue to open the redirected URL for reading content.

Secondly, read content from the URL. If the URL is referring to a file, then ignore it (Figure 3.4). If the URL is referring to an image, then apply this URL as a query to the image search engine such as Google Images (Figure 3.5). If the URL refers to Youtube, we get the description and keywords value from metadata in the <meta> tag (Figure 3.6). If the URL refers to a web page, we check if the metadata exists first, then we get keywords, description and title values. Else if metadata is not available, but keywords for the page are defined then get these keywords. Otherwise, we get title value from the page, as shown in Figure 3.7.

Lastly, we update filtered content from URL to the content column in datastore. For example, the URL <http://j.mp/JoIb2K> after URL filtering process will be transformd into several tags as



Figure 3.4: URL filtering for a file

follows:

```
{
  ...
  "content": "少女時代, 子團體, 徐玄, 太妍, 蒂芬妮, TaeTiSeo, Twinkle",
  "content_raw": "http://j.mp/JoIb2K",
  ...
}
```

3.2.4 Tokenization

There are no straightforward methods to tokenize a Chinese sentence because Chinese text does not have word boundaries and word is a fundamental linguistic unit. Therefore, we develop a two-step tokenization mechanism based on dictionary in this section.

Tsai [84] implemented a Chinese segmentation algorithm named MMSEG based on maximum matching algorithm and Ma [54] showed the procedures of the CKIP Chinese segmentation system, including the disambiguation algorithm for resolving segmentation ambiguities and identifying unknown words.

These two Chinese segmentation algorithms and implementations (MMSEG and CKIP) are popular among Traditional Mandarin Chinese users. However, we only care about the matching of keywords instead of the segmentations of a sentence. Therefore, we do not employ Chinese segmentation system but a maximum matching algorithm based on corpus dictionary and which is stored in a trie data structure.

Matching Algorithm with Recursively Implemented StorAge (MARISA) is a space-efficient trie data structure, while libmarisa [91] is a C++ library implementation of MARISA. We use marisa-trie [57] Python package, a Python version binding of libmarisa as the trie implementation to store interest keywords dictionary and to find the longest prefixes keyword. Listing 3.4 demonstrates how to use marisa-trie library to build a trie and find all prefixes by a given key.

Listing 3.4: Find all prefixes of a given key by marisa trie

```
>>> import marisa_trie
>>> trie = marisa_trie.Trie([u'key1', u'key2', u'key12'])
```

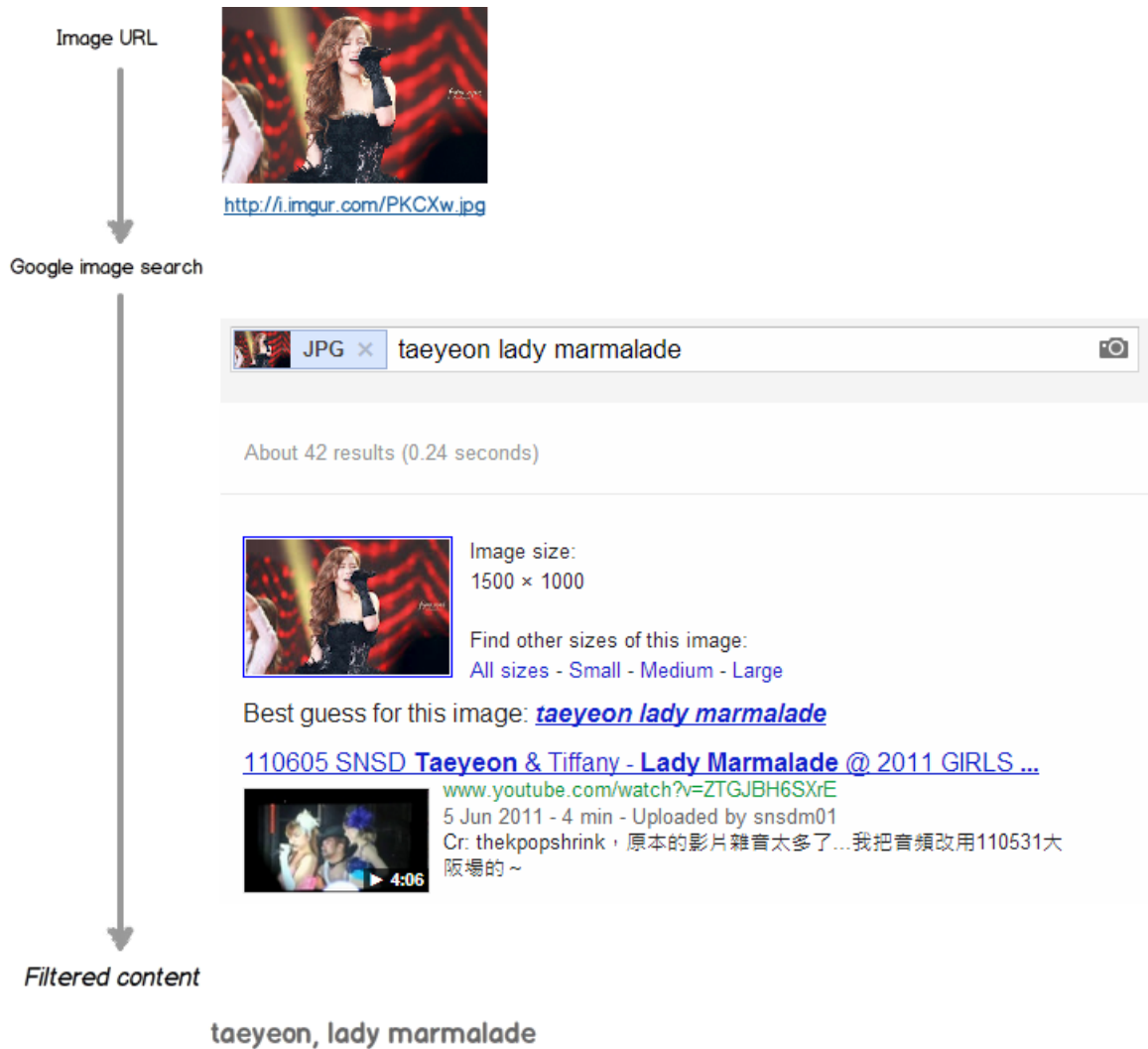


Figure 3.5: URL filtering for an image



Figure 3.6: URL filtering for a Youtube link



Figure 3.7: URL filtering for a web page

```
>>> trie.prefixes(u'key12')
[u'key1', u'key12']
```

We normalize sentences with the following five steps:

1. insert space behind CJK characters and before ASCII characters
2. insert space behind ASCII characters and before CJK characters
3. replace punctuation characters with whitespaces, replace continuous spaces with a single whitespace
4. strip the whitespaces found in the beginning or end of a sentence, and finally
5. convert case-based characters into lowercase.

Firstly, according to Unicode code charts, CJK characters (unified ideographs) are located in the range between 4E00 and 9FFF, Katakana in the range between 30A0 and 30FF, and Hiragana in the range between 3040 and 309F. We employ the Python regex extension, an alternative regular expression module to replace the re module from Python standard library and apply these ranges to define several regular expressions for normalization. This step is inspired by the



Figure 3.8: Demonstration of the normalize function

project: “為什麼你們就是不能加個空格呢？” [86]. `p_mixed_1` and `p_mixed_2` define the pattern set of Unicode characters and ASCII characters. `p_ws` defines the pattern of whitespace character by special character `\s`, which is equivalent to the set `[\t\n\r\f\v]`. `p_punctuation` defines the pattern of punctuation characters by special character `\p{P}`, which is supported by regex module. Listing 3.5 shows the normalization process and Figure 3.8 illustrates how the normalization process normalizes a sentence step by step.

Listing 3.5: Normalize sentences by regular expressions

```

import regex as re

p_mixed_1 = re.compile(ur'([\u4e00-\u9fff\u3040-\u30FF])([a-zA-Z0-9@&;=_[\$\%\^\*\-\+\(\)/])')
p_mixed_2 = re.compile(ur'([a-zA-Z0-9@&;=_[\$\%\^\*\-\+\(\)/])([\u4e00-\u9fa5\u3040-\u30FF])')
p_ws = re.compile(r'[\s]+')
p_punctuation = re.compile(ur'\p{P}+')

def normalize(ctx):
    _ = p_mixed_1.sub(r'\1 \2', ctx)
    _ = p_mixed_2.sub(r'\1 \2', _)
    _ = p_punctuation.sub(' ', _)
    _ = p_ws.sub(' ', _)
    return _.strip().lower()

```

Secondly, we generate a list of indexes of whitespace characters in the sentence, then use the index to retrieve terms. The purpose of this algorithm is to determine whether it is a CJK

term or a western term. The pseudocode is shown in Algorithm 2 and Figure 3.9 depicts the tokenization process step by step.

```

Input: context, a string
         trie, a marisa-trie
Output: terms, a set of matched keywords

1 begin
2   index = 0
3   terms = an empty set
4   while index < length(context) do
5     if context[index] is a white space then
6       | index += 1
7     end
8     match = trie.longest_prefix(context[index:])
9     if match is not null then
10    | terms.add(match)
11    | index += length(match)
12  end
13  else if context[index] in [a-zA-Z0-9] then
14    | index = next index of white spaces occurs in the context
15  end
16  else
17    | index += 1
18  end
19 end
20 return terms
21 end

```

Algorithm 2: Tokenization process

3.2.5 Plurks Preprocessing

The preprocessing procedure is carried out in three phases (Figure 3.10) as follows:

- Apply the URL filtering process to transform URL links into text for extending content.
- Apply the tokenization mechanism to extract keywords from the plurk which are included in interest keywords hierarchy.
- Transform raw content into tags, and update the records to the datastore.

MongoDB is an open source document-oriented database management systems and a part of the NoSQL family. It stores records in BSON format, which support several data types such as String, Integer, Boolean, Double, Null, Array, and Object, etc. As mentioned in subsection 3.2.1, we store the extended and filtered content from phase one into *content* field in String type



Figure 3.9: Demonstration of the tokenize function

and tokenized tags from phase three into *tags* field. We demonstrate how to find a plurk after preprocessing with a specific tag “少女時代” from MongoDB as follows:

```

> use plurk
switched to db
> db.plurks.findOne({tags: "少女時代"})
{
  "_id" : 996252142,
  "owner" : 3461880,
  "qualifier" : "wonders",
  "content": "少女時代,子團體,徐玄,太妍,蒂芬妮,TaeTiSeo,Twinkle 120526 TaeTiSeo Twinkle 少女時代太妍、蒂芬妮、徐玄所組成的子團體「太蒂徐」以首張專輯《Twinkle》登台已滿四周,每周都穿上不同風格的表演服的她們,26日以性感可愛的粉色系空",
  "content_raw" : "http://j.mp/JoIb2K\nhttp://youtu.be/C8HjWFPY78I\n少女時代太妍、蒂芬妮、徐玄所組成的子團體「太蒂徐」以首張專輯《Twinkle》登台已滿四周,每周都穿上不同風格的表演服的她們,26日以性感可愛的粉色系空",
  "tags": ["少女時代", "太妍", "蒂芬妮", "徐玄", "TaeTiSeo"],
  "posted_at" : ISODate("2012-05-28T06:09:10Z"),
  "updated_at" : ISODate("2012-07-01T14:29:15.528Z")
}

```



Figure 3.10: Demonstration of the preprocessing

3.3 Community Detection

In this section, we will introduce the algorithm for sampling network and the algorithm to find community partitions for the SNSD system to derive users' interest topics.

3.3.1 Snowball Sampling

Snowball sampling algorithm works like a pyramid scheme. It provides a fairly complete picture of the network surrounding of the sampling target.

We have to determine a depth limit for this algorithm. In practice, we usually limit the depth of sampling to two (friends-of-friends) or three (friends-of-friends-of-friends). Listing 3.6 depicts an iterative version of the algorithm with depth of two.

Listing 3.6: Iterative sampling

```
def sampling(G, user):
    for friend in read_friends(user):
        G.add_edge(user, friend)
        for friend_of_friend in read_friends(friend):
            G.add_edge(friend, friend_of_friend)
```

3.3.2 Modularity and Louvain Algorithm

The Louvain algorithm [85] is a heuristic greedy method based on modularity optimization to provide excellent results for various application to large network. This algorithm consists of two phases that are repeated iteratively: modularity optimization and community aggregation. Main idea of the processes explained by [26, 80] is restated as the pseudocode in Algorithm 3.

We put all vertices of graph G in their individual community at the beginning of this algorithm. In the first phase, for each vertex i we consider its neighbors j to compute the gain of modularity by putting i into the community of j . If no positive gain available, i stays in its original community.


In the second phase, we aggregate vertices in the same community to a new supervertice and build a new graph with supervertice as new vertice. The weight of the edge $e(V_s, V_t)$ between supervertices V_s and V_t of the new graph is calculated. Once this second phase is completed, it is reapplied to the first phase of the algorithm until a maximum of modularity is attained. Taking a graph from Tang [80] for example, the input graph will be partitioned into two communities

as shown in Figure 3.11.

We employ the `python-louvain` [82] library which is developed by Thomas Aynaud as the implementation of Louvain algorithm. This library is based on `NetworkX` [99], which is a Python scientific library for studying graphs and networks. For example, we create an undirected graph with ten vertices and apply Louvain algorithm by `python-louvain` library to find community partitions as follows:

Listing 3.7: An community detection example in NetworkX

```
>>> from community import best_partition
>>> import networkx as nx
>>> G = nx.Graph()
>>> G.add_edge(1, 2)
>>> G.add_edge(1, 3)
>>> G.add_edge(1, 4)
>>> G.add_edge(2, 3)
>>> G.add_edge(3, 4)
>>> G.add_edge(4, 5)
>>> G.add_edge(4, 6)
>>> G.add_edge(5, 6)
>>> G.add_edge(5, 7)
>>> G.add_edge(5, 8)
>>> G.add_edge(6, 7)
>>> G.add_edge(6, 8)
>>> G.add_edge(7, 8)
>>> G.add_edge(7, 9)
>>> print(G.nodes())
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> print(G.edges())
[(1, 2), (1, 3), (1, 4), (2, 3), (3, 4), (4, 5), (4, 6), (5, 8), (5, 6), (5, 7), (6, 8), (6, 7),
 (7, 8), (7, 9)]
>>> part = best_partition(G)
>>> for com in set(part.values()):
...     list_nodes = [n for n in part.keys() if part[n] == com]
...     print com, list_nodes
...
0 [1, 2, 3, 4]
1 [5, 6, 7, 8, 9]
```



Input: $G = (V, E)$

Output: S

A is the adjacency matrix of G

A_{ij} is the weight of the edge between vertex i and j

$m = \frac{1}{2} \sum_{ij} A_{ij}$ is the number of degrees of G

$C(i)$ is the community of vertex i

d_i is the degree of vertex i

$B_{ij} = A_{ij} - \frac{d_i d_j}{2m}$

$$e(v_s, v_t) = \begin{cases} \text{an edge with the weight } w(C(s), C(t)) & \text{if } s \neq t \\ \text{a self-loop edge with the weight } w(C(s), C(t)) & \text{otherwise} \end{cases}$$

$$w(C(s), C(t)) = \sum_{i \in C_s, j \in C_t} w(i, j)$$

$w(i, j)$ is the weight of e_{ij}

```
1 begin
2   for  $i \in V$  do
3     |  $C(i) = \{i\}$ 
4   end
5   while true do
6     for  $i \in V$  do
7       |  $\Delta_i(j) = B_{ij} - B_{ii}$ 
8       |  $j^*(i) = \arg \max \{\Delta_i(j) \mid j \in [n]\}$ 
9       | if  $j^*(i) \neq i$  and  $\Delta_i(j) \geq 0$  then
10        | // Set  $i$  and  $j^*$  in the same community
11        |  $C(i) = C(j^*) \cup C(i)$ 
12        |  $C(j^*) = C(j^*) \cup C(i)$ 
13      end
14    end
15    // Delete redundant elements in the collection of communities
16     $S \leftarrow \text{DelRedun}(\{C(i) \mid i \in V\})$ 
17    if  $\forall i, j^*(i) = i$  then
18      return  $S$ 
19    else
20      for  $C\{k\} \in S$  do
21        | //  $v_k$  is a supervertex
22        |  $v_k \leftarrow C\{k\}$ 
23      end
24       $l = |S|$ 
25       $V' = \{v_1, \dots, v_l\}$ 
26      for  $v_s, v_t \in V'$  do
27        |  $e_{st} = e(v_s, v_t)$ 
28      end
29       $E' = \{e_{st} \mid v_s, v_t \in V'\}$ 
30       $G = (V', E')$ 
31    end
32  end
33 end
```

Algorithm 3: Louvian algorithm

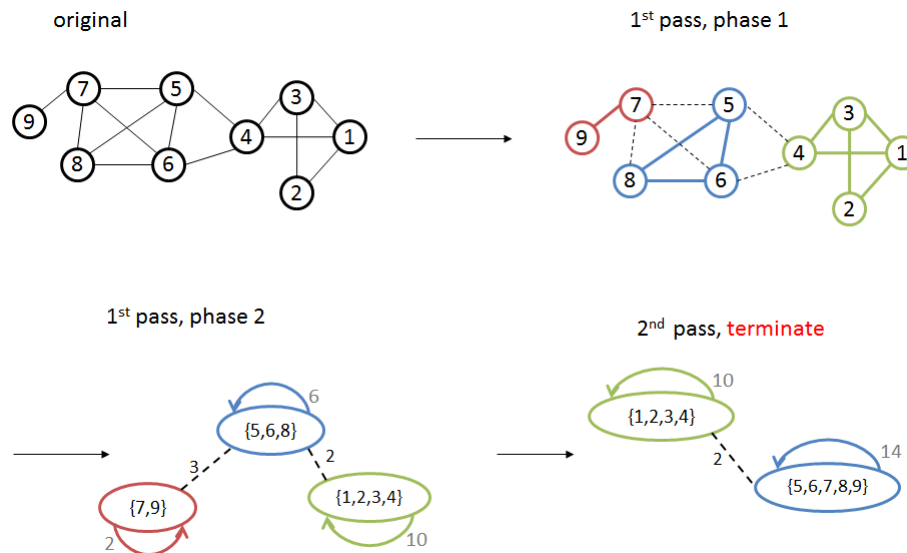


Figure 3.11: Visualization of the steps of Louvain algorithm

3.3.3 Filtering

Plurk allows users to claim his/her profile including: gender, age, location, etc., as shown in Figure 3.12. We use gender, age, privacy status (public or private), and karma (a metrics for activity) as parameters for snowball sampling algorithm to reduce the sampling range on demand.

3.4 Interest Hierarchy Model

In order to determine user's interest topics, we define an interest keyword hierarchy based on a famous BBS (Bulletin Board System) site PTT.cc in Taiwan to collect interest keywords and categorize them. In contrast to the Go!Plurk project, the new hierarchy can define detailed interests category instead of a one-level hierarchy structure.

Trees and graphs are universal data structures; however, they do not easily fit into a relational data model design. Karwin [45] introduces several hierarchical SQL models such as Adjacency List, Recursive Query, Path Enumeration (Materialized Path), Nested Sets, and Closure Table.

According to Table 3.1, Adjacency List is the most conventional but difficult to query a full tree. Recursive Query is more efficient than Adjacency List design, but MySQL does not support WITH syntax, which is defined in SQL-99. Path Enumeration is not able to check referential integrity and it stores information redundantly. Nested Sets also fail to support referential integrity and is difficult to manipulate. Closure Table is the most versatile and elegant design

(a) General settings



(b) Public profile view

Figure 3.12: Plurk profile: general information

Design	Tables	Query Child	Query Tree	Insert	Delete	Ref. Integrity
Adjacency List	1	Easy	Hard	Easy	Easy	Yes
Recursive Query	1	Easy	Easy	Easy	Easy	Yes
Path Enumeration	1	Easy	Easy	Easy	Easy	No
Nested Sets	1	Hard	Easy	Hard	Hard	No
Closure Table	2	Easy	Easy	Easy	Easy	Yes

Table 3.1: Comparison of hierarchical data model design from Ref. [45]

but requires an additional table to store the relationships.

Instead of storing relationships between ancestor and descendant like the other designs, Closure Table stores all paths through the tree, as shown in Figure 3.13. Figure 3.13(b) illustrates how the nodes are stored. Listing 3.8 shows the schema for Closure Table, and we can easily find descendants or ancestors, insert new records and delete a leaf node or subtree. For example, we use the query (Listing 3.9) to retrieve descendants of “K-Pop” and insert a new node “Kara” with id as 8 under “K-Pop” by query (Listing 3.10).

Listing 3.8: SQL schema for Closure Table

```
CREATE TABLE `taxonomies` (
  `id` INT(11) NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(64) NULL DEFAULT NULL,
```

```

PRIMARY KEY (`id`)
);

CREATE TABLE `hierarchies` (
  `ancestor` INT(11) NOT NULL,
  `descendant` INT(11) NOT NULL,
  PRIMARY KEY (`ancestor`, `descendant`),
  FOREIGN KEY (`ancestor`)
    REFERENCES `taxonomies` (`id`),
  FOREIGN KEY (`descendant`)
    REFERENCES `taxonomies` (`id`)
);

```

Listing 3.9: Retrieve descendants of “K-Pop”

```

SELECT t.*
FROM taxonomies AS t
     JOIN hierarchies AS h ON t.id = h.descendant
WHERE t.ancestor = 4;

```

Listing 3.10: Insert a new node under “K-Pop”

```

INSERT INTO hierarchies (ancestor, descendant)
SELECT h.ancestor, 8
FROM hierarchies AS h
WHERE h.descendant = 4
UNION ALL
SELECT 8, 8;

```

We extend the Closure Table design and employ MySQL as datastore for the interest keyword hierarchy. We use four tables: hierarchies, taxonomies, vocabularies, and lexicons to model the relationship. Based on Closure Table design, table hierarchies store the relationship between taxonomies. Table taxonomies defines the hierarchy structure. Table vocabularies stores the relationship between lexicons and taxonomies. Table lexicons defines the interest keywords and associated information.

We use the following SQL query to obtain the interest keywords hierarchy structure that is associated with the keyword “Girls Generation”:

Listing 3.11: Obtain hierarchy structure by Closure Table design

```

SELECT x.id, x.term, v.belongs_to,
       GROUP_CONCAT( h.ancestor ORDER BY a.id ) AS ancestors,
       GROUP_CONCAT( a.name ORDER BY a.id ) AS path
FROM lexicons AS x

```

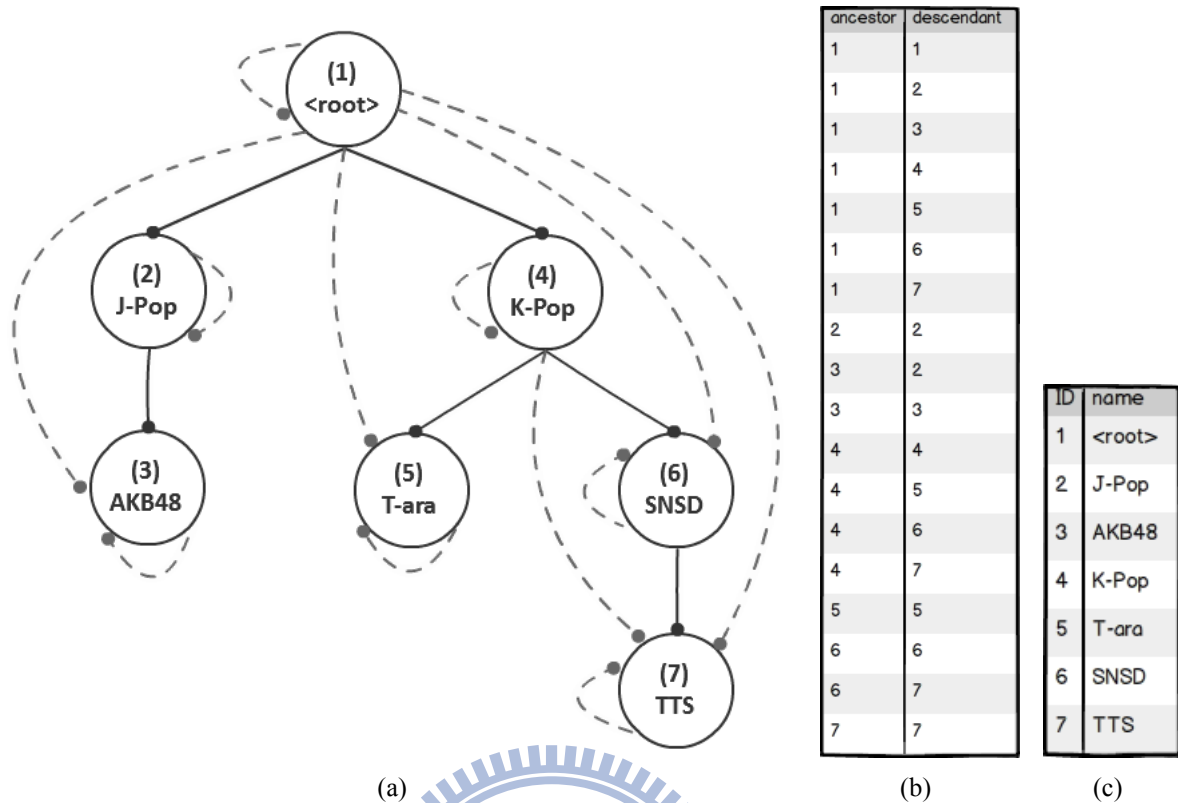


Figure 3.13: A sample Closure Table

```

JOIN vocabularies AS v ON ( v.term = x.id )
JOIN taxonomies AS t ON ( v.belongs_to = t.id )
JOIN hierarchies AS h ON ( t.id = h.descendant )
JOIN taxonomies AS a ON ( a.id = h.ancestor )
AND x.term LIKE 'Girls Generation'
GROUP BY x.term, t.id

```

3.5 Datastore Architecture

In order to store billions of records into datastore, we employ MongoDB and build a cluster with twenty nodes, donated by Delta Electronics, Inc., to improve the throughput and availability by sharding and replication methodologies.

There are three components in the MongoDB cluster: mongos, config server, and shard server, as shown in Figure 3.15. Mongos instances function as router to deal with requests from clients and serve as the access point for clients, which forward requests to the appropriate shards by coordinating with config servers. Config servers maintain the shard metadata and definitive information about the cluster including chunks, shards, and mongos processes information. Shard servers are used to store data.

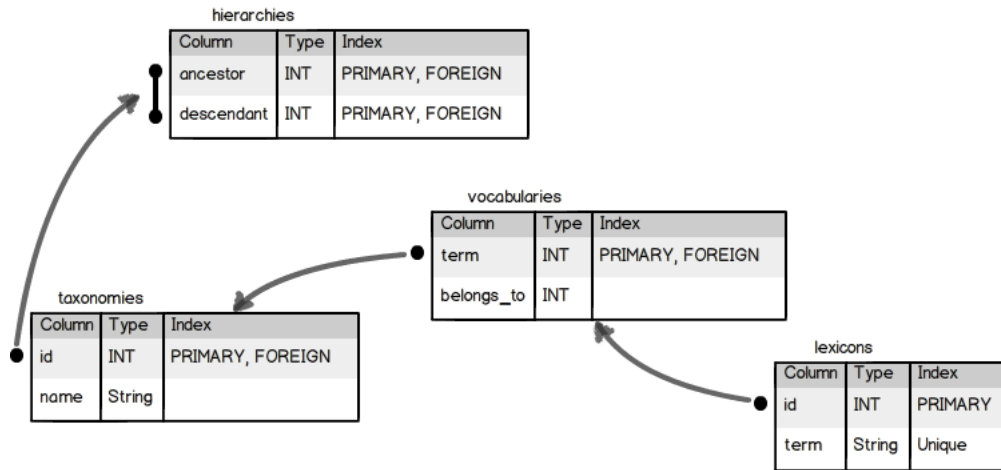


Figure 3.14: Table Relationships

MongoDB employs replica sets to achieve High Availability (HA) and auto-sharding for scaling out. Replica sets are used for data redundancy, distributing read load, and high availability (automated failover). A replication set consists of three or more nodes which are copies of each other; the set members will elect a primary node and the others as secondary nodes automatically. MongoDB drivers and mongos will detect a replication set primary changes automatically.

A shard is comprised of one or more servers in a cluster which is responsible for the same subset of data. If there is more than one server in a shard, each server has an identical copy of data and a shard is usually a replica set in production.

MongoDB's sharding is based on a shard key which determines how data will be distributed across the cluster. For example, we have an application that stores fan's personal profile of Girls' Generation, a nine-member South Korean pop girl group. Each profile document contains a *favorite_member* field, which shows who is the fan's most favorite. Its value would be "Taeyeon", "Jessica", "Sunny", "Tiffany", "Hyoyeon", "Yuri", "Sooyoung", "Yoona", or "Seohyun". We use this field as the shard key.

The collection starts with one chunk $(-\infty, \infty)$ on a shard in one shard server. Once that chunk gets big enough, it will be split into two chunks with the range $(-\infty, "Yoona")$ and $["Yoona", \infty)$. All of the profile documents with "Hyoyeon", "Jessica", "Taeyeon", or "Tiffany" will be placed into the first chunk and the rest documents with "Yoona", "Yuri", "Seohyun", "Sooyoung", or "Sunny" will be placed in the second chunk. With more and more documents added, eventually we will end up with nine chunks, i.e. one shard per member:

- $(-\infty, "Jessica")$

- [“Jessica”, “Taeyeon”)
- [“Taeyeon”, “Tiffany”)
- [“Tiffany”, “Yoona”)
- [“Yoona”, “Yuri”)
- [“Yuri”, “Seohyun”)
- [“Seohyun”, “Sooyoung”)
- [“Sooyoung”, “Sunny”)
- [“Sunny”, ∞)

The above example chooses a low-cardinality shard key which has a fixed maximum number of chunks and it will never be able to use more than that number of shards for the collection. It is recommended to choose a high-cardinality field as the shard key such as the profile unique id number in the above example or a field with MongoDB’s ObjectID datatype. In our case for storing plurks in the MongoDB, we choose the plurk’s *plurk_id* field, which is the unique id for a plurk, as the shard key.

We employ three config servers and two mongos servers, and set up nine shards with nine replicas. That is, one shard per replica set, where each replica consists of two shard servers and one arbiter in our production server layout. This is recommended by official production configuration [78]. Besides, we employ diskless architecture to deploy these twenty servers. Detailed cluster configuration is shown in Figure 3.16 and configurations are given in Appendix B.

3.6 Celery Task Queue

Computing community partitions for a user is a large computational bottleneck in our system. Thus, we employ Celery [8], which is an asynchronous distributed task queue library for Python, to utilize workstations from NCTU CSCC [103] for load-balancing.

A large web application might contain time-consuming functionalities which cannot be done or need not to be done in real-time such as making thumbnails or processing uploaded files. This type of problem is called fire-and-forget models, and the process request will be blocked until

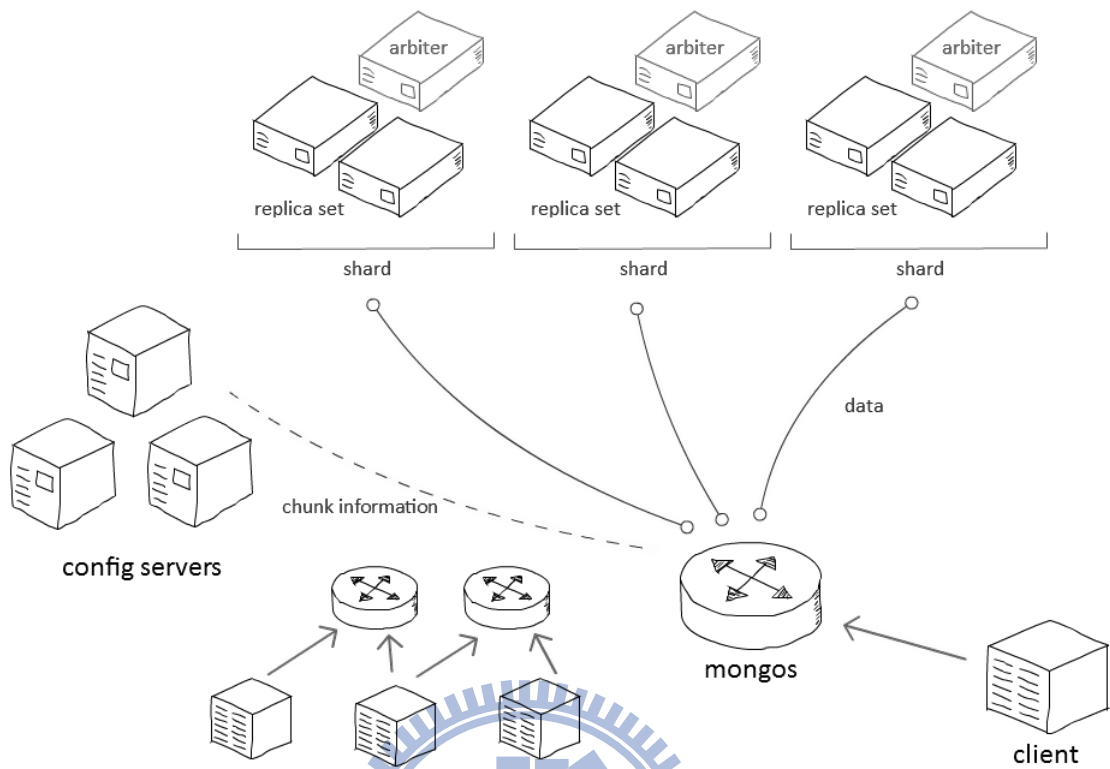


Figure 3.15: MongoDB cluster architecture

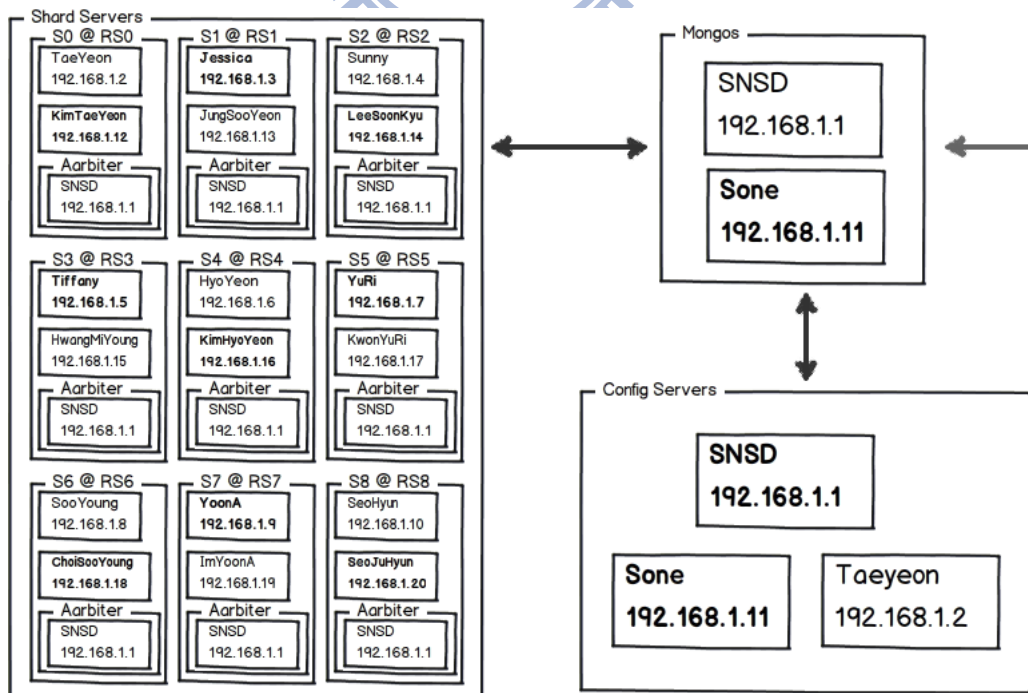


Figure 3.16: MongoDB cluster configuration

the job is done. In our case, we cannot compute every community partitions for a user in a short time which depends on the number of friends of the user. Given this limitation, we employ Celery task queue for handling jobs asynchronously to prevent blocking.

Celery has two execution units: broker and worker server. Celery requires a message broker solution for tasks queuing and storing results. It supports several popular open source choices, including: RabbitMQ, Redis, traditional RDBMS via SQLAlchemy or Django object-relational mapping (ORM), and MongoDB... etc. It also supports Amazon SQS, if you deploy Celery in Amazon Web Services (AWS) platform. Worker servers are the execution instances. You can choose multiprocessing, Eventlet, gevent, or threading modules as process pool implementation for specific purposes.

We define three trust levels for our production environments: trusted, semi-trusted, and untrusted. Trusted level means we have root privileges and unauthorized users are not allowed to login to the machine. Semi-trusted level means we can login to the machine to use shell but do not have root privileges. Besides, other user has no privileges to change file permission which belongs to us. Untrusted level is similar to semi-trusted level, but user might force to change file permissions. In general, dedicated server is in trusted level, shared UNIX-like workstation is in semi-trusted level and shared Windows workstation is in untrusted level.

Redis, a high performance open source key-value store or data structure server, is an excellent broker candidate for Celery. However, it is designed for trusted level scenario because it does not provide access control list (ACL) function for authentication, but a single global master key instead.

Even though Redis supports master-slave replication, which is suitable for scaling out reading performance by adding read-only slave nodes and connects each Redis node via Secure Shell (SSH) tunnel for basic connection security. However, we cannot employ Redis as Celery broker because Redis slaves are read-only. Celery workers shall communicate with Redis master directly. Intruders can use the SSH tunnel to apply brute-force attacks on master in the semi-trusted level node. Moreover, if the global master key is leaked or any Redis node is compromised, data would be deleted or falsified by attacker.

Given the above limitations and that we have no fully trusted machines to build a Redis server in our production environment, we choose RabbitMQ, a stable message queue based on AMQP and support ACL as broker.

3.7 Celery Cluster Layout and Worker Configurations

According to previous consideration for choosing RabbitMQ as the Celery broker, we implement a scalable Celery worker cluster as shown in Figure 3.17.

We employ two virtual machines from openstack.nctu.edu.tw with public IP address as Redis read-only slaves which serve as relationship data provider for workers from NCTU CSCC, which connect to the Redis master via SSH tunnel using autossh to manage and restart SSH tunnels automatically if needed. Besides, we set up security group (firewall) rules, as shown in Figure 3.18, so that these two Redis slaves can only provide data to specific workers from NCTU CSCC core subnet (140.113.235.0/24). After that, we set up a RabbitMQ instance with custom ACL policy then configure workers to connect to the RabbitMQ.

In the previous implementation, we choose Python multiprocessing module as process pool implementation in standard CPython interpreter. However, the computation performance is poor especially in the case of a user having a large number of friends. Therefore, we employ PyPy [98] as the alternative process pool implementation with its multiprocessing module. PyPy is a much faster interpreter for Python language and Just-in-Time compiler. Besides, it is 100% compatible with the original CPython interpreter.

However, PyPy interpreter can only execute programs in pure Python source code currently, i.e. we cannot employ PyPy to execute Python programs with CPython or Cython extension module such as PyZMQ, the Python binding library for ZeroMQ and gevent. Fortunately, NetworkX, the library which we employ to implement Louvain algorithm, is compatible with PyPy and it is much faster than the CPython interpreter.

We deploy Python programs with VirtualEnv, a Python tool used to create isolated environments that we can install Python packages without interfering with either the other VirtualEnvs or the system wide packages. VirtualEnv support CPython and PyPy interpreter and it can help users build their Python environment without root privileges.

CSCC workstations run two types of operating systems: FreeBSD and Gentoo Linux. Their home directories are mounted from a NetAPP centralized storage server, as shown in Figure 3.19. We have to make two VirtualEnvs for PyPy for these two types of OS to execute Celery worker. However, the latest VirtualEnv version 1.8.2 is not capable of building a PyPy virtual environment on FreeBSD 8.3, so we can only prepare a CPython environment on FreeBSD workstations.

Given the above limitation, we have to route the task manually by adding a new queue for FreeBSD workstations for load-balancing. First, define two Celery queues (CELERY_QUEUES): default and bsd, and define two routing keys for these queues: pypy and python27. Second, define the default queue name (CELERY_DEFAULT_QUEUE), default routing keys (CELERY_DEFAULT_ROUTING_KEY) and define default route (CELERY_ROUTES) for tasks.

Listing 3.12: Celery Routing Configuration

```
from kombu import Queue

CELERY_QUEUES = (
    Queue('default', routing_key='pypy'),
    Queue('bsd', routing_key='python27'),
)
CELERY_DEFAULT_QUEUE = 'default'
CELERY_DEFAULT_ROUTING_KEY = 'pypy'

CELERY_ROUTES = {
    'tasks.communities': {
        'queue': 'default',
        'routing_key': 'pypy',
    },
}
```

Third, start Celery worker process on FreeBSD workstations and Gentoo Linux workstations, for instance, bsd1 and linux1 with the `-Q` option to determine the queue consumed by the Celery worker process. For example, bsd1 will consume queue bsd, linux1 will consume queue default, and the worker on main server, random, will consume both default and bsd in case of FreeBSD workstations failure or we have to scale-in our system by disconnecting CSCC workstations.

```
[liic@linux1 ~]$ celery worker -Q default --autoscale=16,8
[liic@bsd5 ~]$ celery worker -Q bsd --autoscale=16,8
[ken@random ~]$ celery worker -Q default,bsd --autoscale=20,2
```

In the end, as CSCC limits the amount of memory consumption for every user defined in `/etc/login.conf` for FreeBSD and `/etc/security/limits.conf` in Gentoo Linux, but does not limit the use of `/tmp` storage space, we utilize the storage space as local cache of relationship data from Redis slaves by employing `FileSystemCache` caching function from `Werkzeug` library.

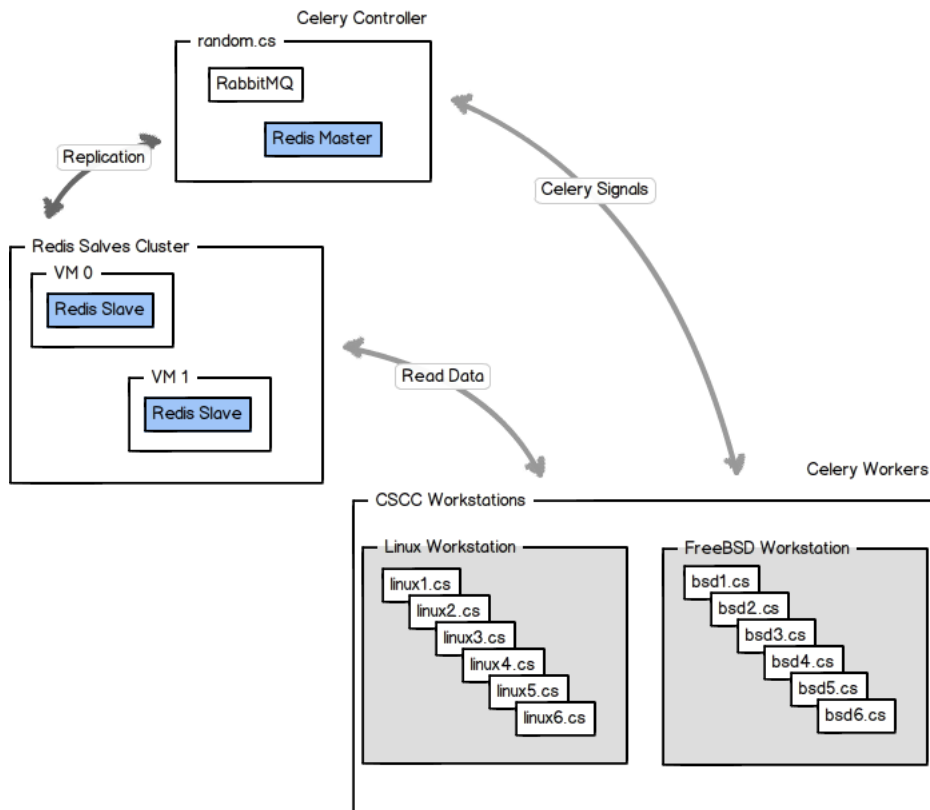


Figure 3.17: Celery cluster architecture

Edit Security Group Rules ✕

Security Group Rules Delete Rules

	IP Protocol	From Port	To Port	Source	Actions
<input type="checkbox"/>	TCP	22	22	140.113.0.0/16 (CIDR)	Delete Rule
<input type="checkbox"/>	TCP	6379	6379	140.113.207.175/32 (CIDR)	Delete Rule
<input type="checkbox"/>	TCP	6379	6379	140.113.235.0/24 (CIDR)	Delete Rule
<input type="checkbox"/>	ICMP	-1	-1	140.113.207.175/32 (CIDR)	Delete Rule
<input type="checkbox"/>	ICMP	-1	-1	140.113.235.0/24 (CIDR)	Delete Rule

Displaying 5 items

Add Rule

IP Protocol: From Port: To Port: Source Group: CIDR:

Figure 3.18: OpenStack security group configurations

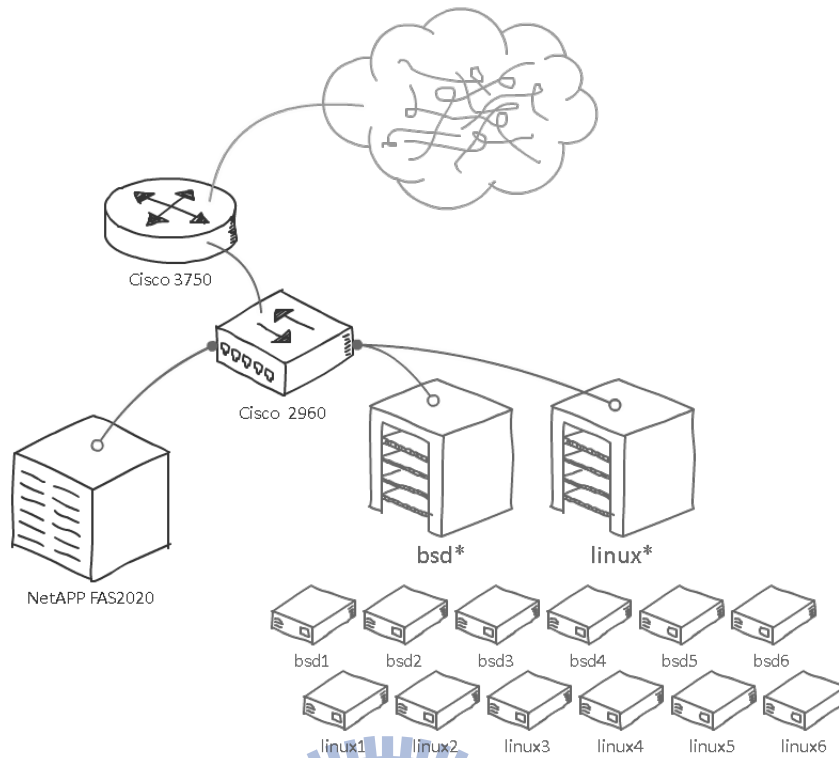


Figure 3.19: CS workstation cluster architecture

3.8 Delta Cluster Deployment

Delta Electronics, Inc. donated 80 servers in two racks to NCTU in early 2012 (Figure 3.20). Unlike normal rack servers, the size of Delta cloud server chassis is customized, as shown in Figure 3.21, and it does not provide VGA port, i.e. you cannot attach a monitor to view the console message by a server but shut it down and plug a VGA card on it, as shown in Figure 3.22. Moreover, these servers are designed for building commodity clouds and employ cost-effective hardware instead of x86 rack servers. Therefore, it is insufficient for administrator to install OS on these server boxes one-by-one because there are a large number of server boxes to be set up and if a server down, it would be difficult to determine whether it is a hardware damage or just kernel panic.

Hence, it is reasonable to employ preboot execution environment (PXE) to build a diskless environment for these server boxes, as shown in Figure 3.19. This scenario boots depend on a remote bootable image and the local storage is used to store data without system files. For example, we can just pull it out of cabinet and plug a new one back into it when a server box fails or plug server boxes and power them on when you need to power on servers to expand a cluster. We have to set up DHCP, TFTP, and Network File System (NFS) or Network Block



Figure 3.20: Servers and racks donated by Delta, Inc.

Device (NBD) as a network storage implementation for PXE procedure. The detailed installation steps and configurations are described in Appendix A.



Figure 3.21: A single Delta server

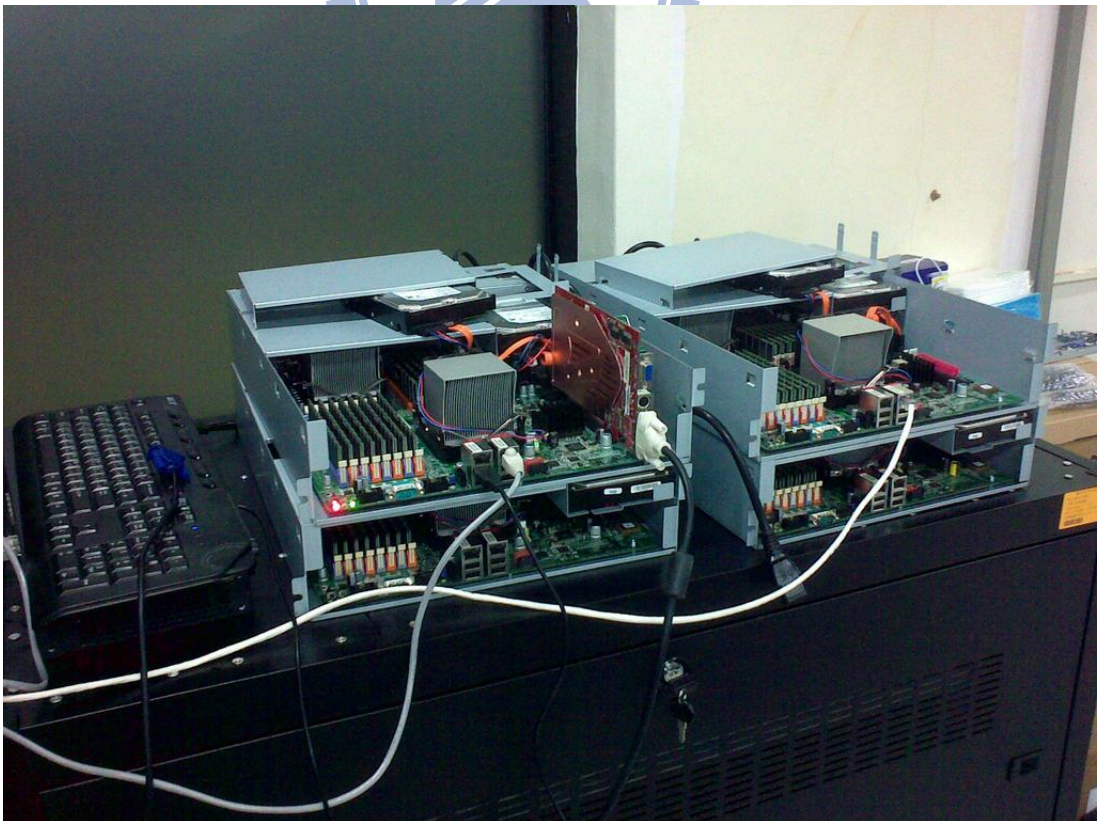


Figure 3.22: Delta server with VGA card

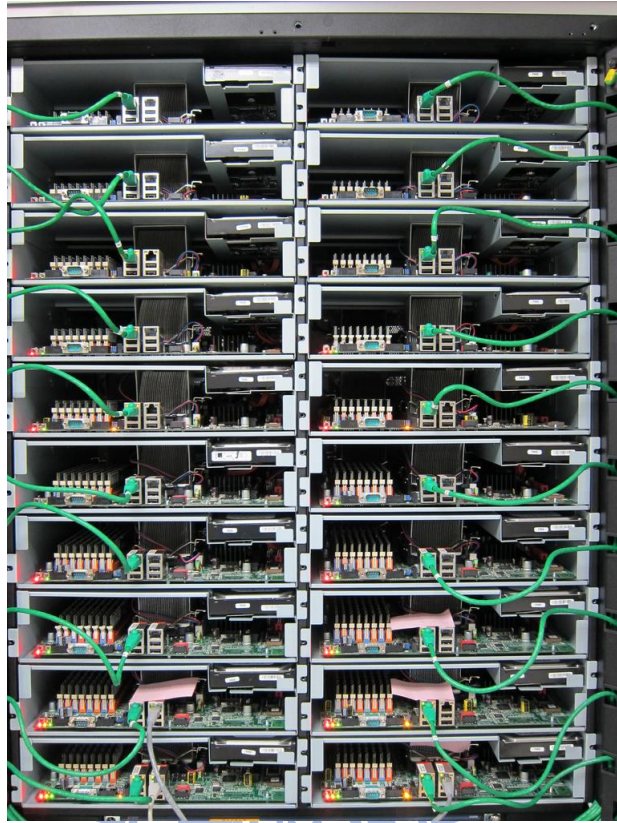


Figure 3.23: Servers installed in rack

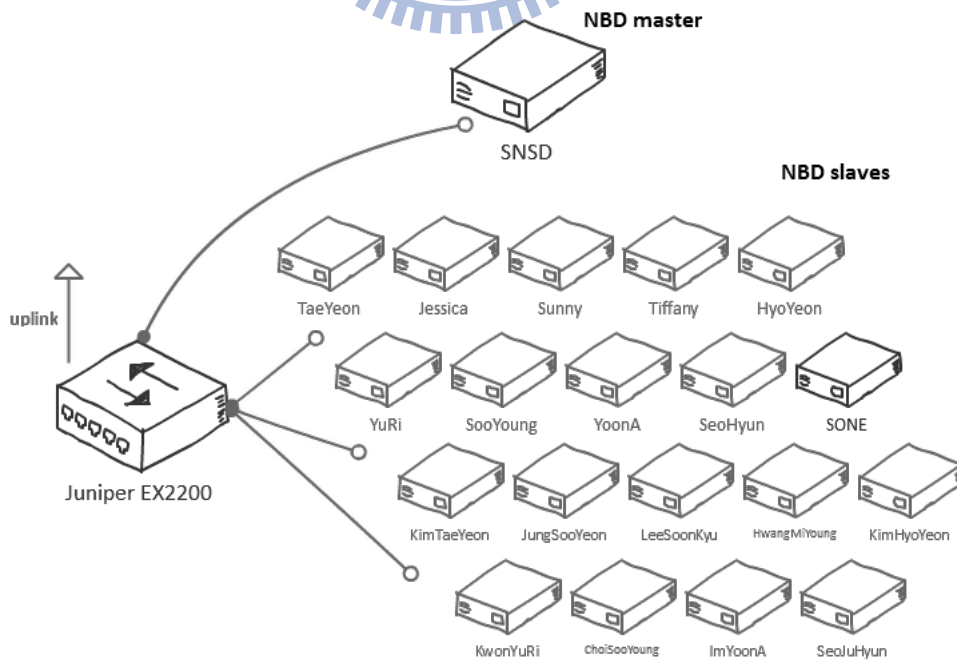


Figure 3.24: Delta cluster architecture

Chapter 4

Experiments

In this chapter, we will depict our experimental environments, performance benchmarks for libraries and systems, and present the website we build for the SNSD system.

4.1 Environment

We deploy several machines to accomplish such systems as the SNSD system, distributed crawling system, and the website for the SNSD system. Roles and specifications of these machines are listed in Table 4.1.

Our server `random.cs` plays the most important role. The website and the message queue for Celery are run on top of it. Besides, it is also the crawling controller: ventilator, broker, registry and commander components are run on top of it as well. Delta cloud cluster, running Arch Linux [44] operating system, is set up as a MongoDB cluster with replication and sharding features. We employ this cluster to store plurks for SNSD system.

CS workstation cluster is composed of twelve workstations, which is a mixture of Gentoo

Host name	CPU	Memory	OS	Role
<code>random.cs</code>	Intel Xeon X5650	48GB	Arch Linux	Website, Crawling master
<code>Delta cloud</code>	AMD Opteron 270	16GB	Arch Linux	MongoDB, Crawling agent
<code>bsd[1-5].cs</code>	Intel Xeon E5520	8GB	FreeBSD	Celery worker, Crawling agent
<code>bsd6.cs</code>	Intel Xeon E5405	12GB	FreeBSD	Celery worker, Crawling agent
<code>linux[1-5].cs</code>	Intel Xeon E5520	8GB	Gentoo Linux	Celery worker, Crawling agent
<code>linux6.cs</code>	Intel Xeon E5405	12GB	Gentoo Linux	Celery worker, Crawling agent
<code>oracle-[0-1]</code>	Intel Xeon E5-2620	8GB	Arch Linux	Redis slave
<code>ken.cs</code>	Intel Xeon E5462	32GB	Arch Linux	Celery worker, Crawling agent
<code>master-i7</code>	Intel Core i7-2600K	16GB	Arch Linux	Celery worker, Crawling agent

Table 4.1: Machine specifications and roles

Linux [29] and FreeBSD [81] operating systems. These workstations are provided by NCTU CSCC for students. This cluster is used for scaling out computation power by running Celery workers.

Two virtual machines (oracle-0, oracle-1) from openstack.cs.nctu.edu.tw work as another cluster to run Redis instances as read-only slaves for requests from CS workstation cluster. Other machines such as ken.cs and master-i7 perform the role of Celery workers.

Delta cloud cluster, CS workstation cluster (bsd[1-6].cs, linux[1-6].cs), and other machines listed above can be consolidated into random.cs and the SNSD system, crawling system and website will still work properly while scaled in.

4.2 Performance Benchmarks

4.2.1 Python JSON Libraries

In order to find out the fastest available Python JSON library in terms of decoding performance, we test the following eight popular libraries: yajl [51], cJSON [22], czjson [93], simplejson [13], ujson [42], anyjson [73], jsonlib [41], and the JSON library from Python standard library [68]. The best one will be the replacement library for Plurk API.

We choose three types of data encoded in JSON format from Plurk API as benchmark dataset: profile (/APP/Profile/getPublicProfile), friends (/APP/FriendsFans/getFriendsByOffset), and timeline (/APP/Timeline/getPublicPlurks). Dataset will contain Unicode strings and large lists, and we prepare five different sizes for friends type.

Before performance testing, we did functional testing to verify that all candidate libraries do the encoding (serializing an object into a string) and decoding (deserializing a string into an object) well as expected. The cJSON, czjson, and jsonlib library do not pass the verification. Therefore, we will not consider these three libraries in the following tests.

Our benchmark results indicate that anyjson, simplejson, and yajl are faster than standard library as shown in Figures 4.1 thru 4.3. However, anyjson and yajl consume more than 420 MB memory during the testing, and they take 400 MB more than standard library, as shown in Figure 4.4. The ujson performs the best in encoding and decoding tests. It consumes much less memory, only 12.9 MB. As such, ujson is selected to replace the standard library in our implementation.

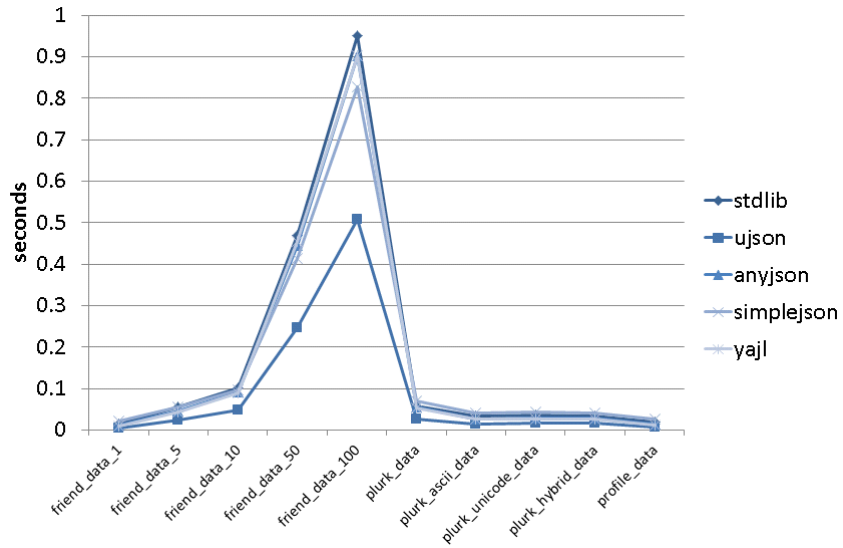


Figure 4.1: Encoding performance

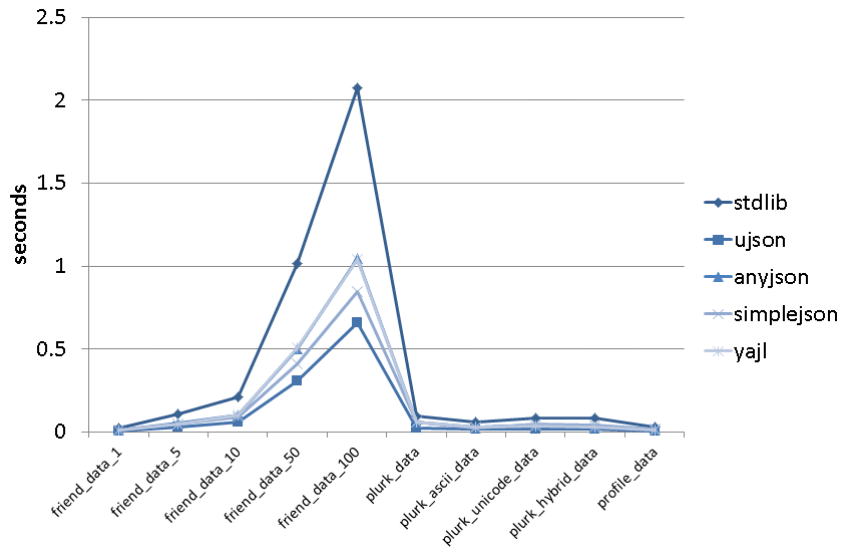


Figure 4.2: Decoding performance

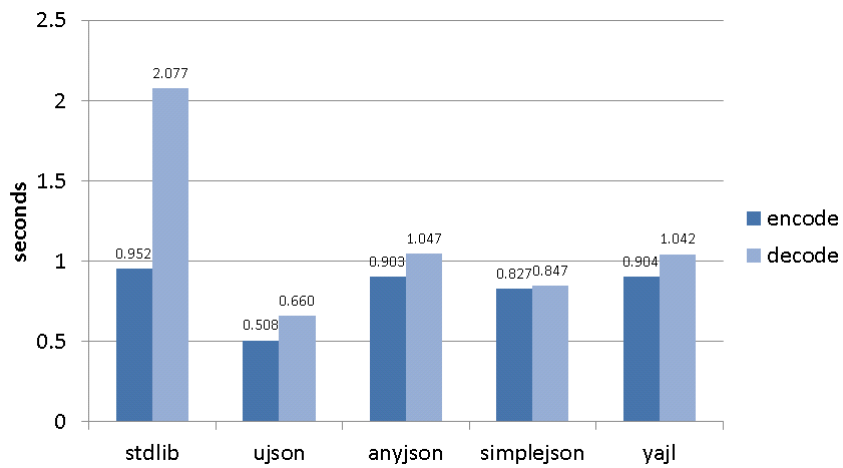


Figure 4.3: Big data performance

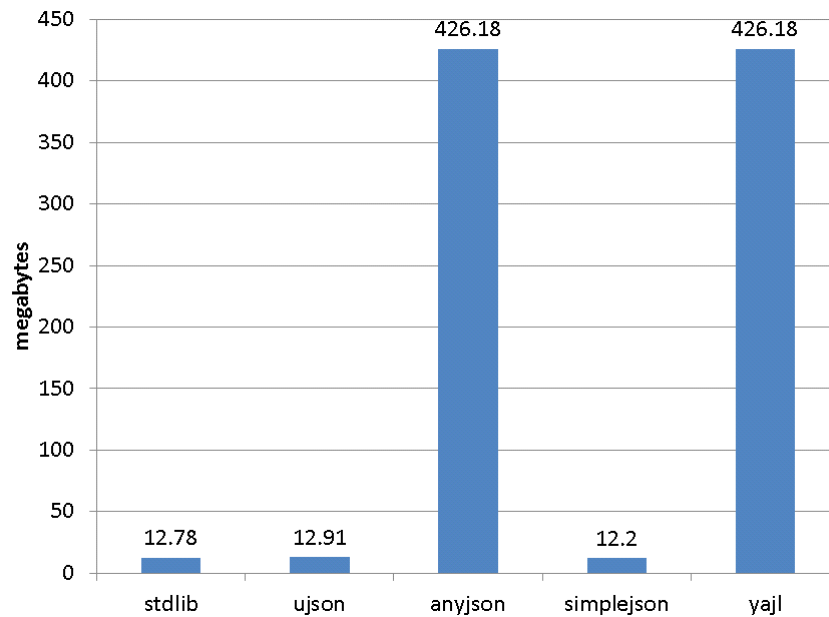


Figure 4.4: Memory usage of JSON libraries

4.2.2 Python Serialization

We need to choose a serialization format for distributed crawling system and Celery to communicate with workers and dispatcher. Instead of encoding performance, we prefer higher decoding performance and encoded data size because there are tens of workers but few dispatcher or controller in distributed crawling system and Celery task queue.

There are five candidates in this experiment: JSON (implemented by ujson), MessagePack, cPickle (Pickle implemented in C), Marshal, and BSON. We randomly select one million plurks as dataset for benchmark.

According to our benchmark results, BSON performs the worst, and others' performances are close. MessagePack is the fastest one as shown in Figure 4.5. In memory usage part, MessagePack consumes least memory in encoding but most in decoding, as shown in Figure 4.6. The major performance difference among the candidates is encoded data size: Dataset in JSON is close to the baseline of 195 MB, MessagePack takes only 144 MB and Pickle takes 240 MB, as shown in Figure 4.7. We employ MessagePack as the serialization format considering the decoding performance and encoded data size.

4.2.3 HMAC-SHA1

Python is a popular general-purpose, high-level scripting language and is regarded as a “glue language”. Even though the execution performance of Python language is much poorer than C/

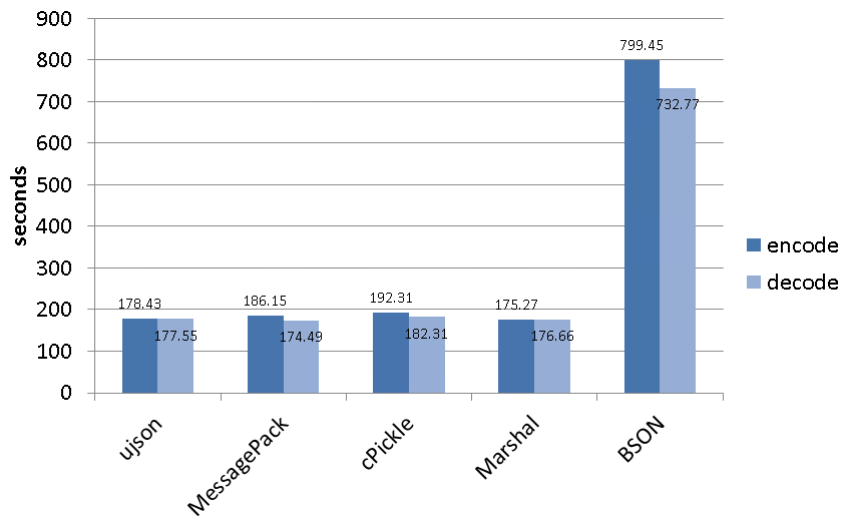


Figure 4.5: Serialization performance

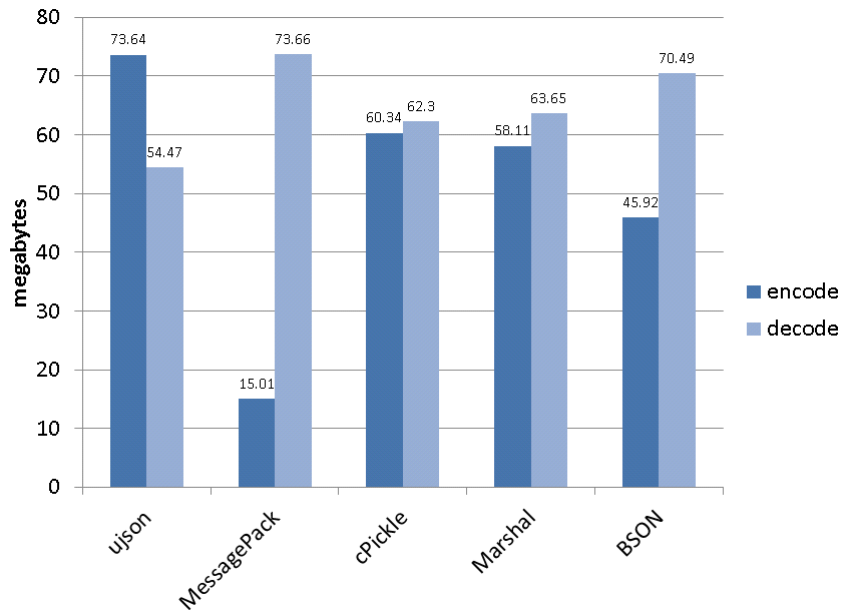


Figure 4.6: Memory usage of serialization libraries

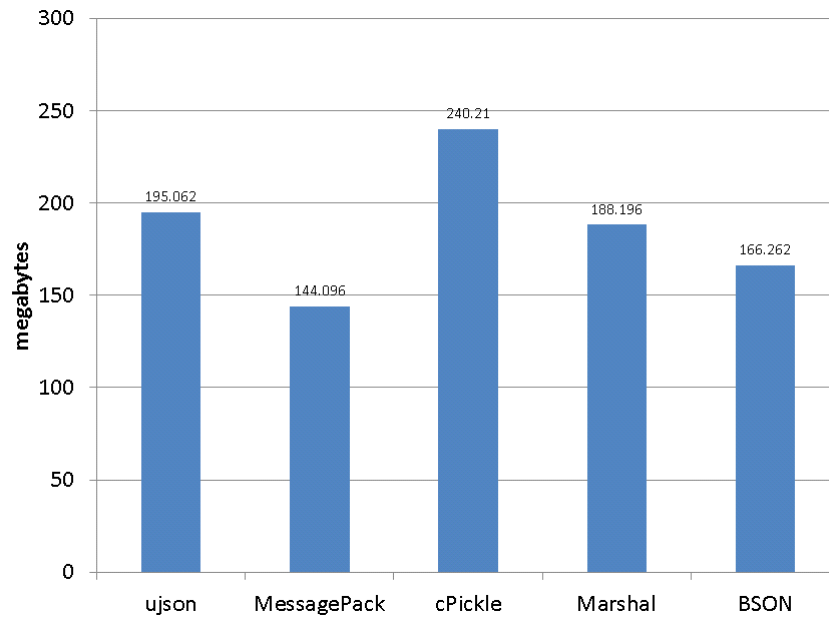


Figure 4.7: Encoded data size

C++ or Java language. It is easily extensible with C/C++ codes to improve computational performance.

According to Python performance tips [101], we can improve performance dramatically by rewriting performance-critical codes in C extension. Examples [69, 100, 53] demonstrate the way to build a Python extension written in Python C-API or Cython, a language for writing Python C extension as easily as Python codes.

In order to accelerate the HMAC-SHA1 procedure, we benchmark the following three implementations and integrate the fastest one into the enhanced Plurk API library: (1) Python standard libraries, (2) M2Crypto extension and (3) customized extension module with OpenSSL library.

Python standard library provides two options for calculating Base64 encoding: `binascii` and `base64`. The `binascii` module is implemented in C and it contains a number of low-level functions to convert data between binary and ASCII-encoded representations. The `base64` module is implemented in Python and it provides encoding and decoding functions as specified in RFC 3548. The `base64` module calls `binascii` module for encoding or decoding input, then translates the alternative alphabet for the '+' and '/' characters in encoded/decoded data.

M2Crypto, a Python wrapper library for OpenSSL, provides several features such as RSA, HMACs, and symmetric ciphers, etc. However, M2Crypto does not provide Base64 encoding function, i.e. we can only use M2Crypto to calculate HMAC signature and we have to encode the binary signature by Python. Listing 4.1 illustrates the usage of M2Crypto library and computing

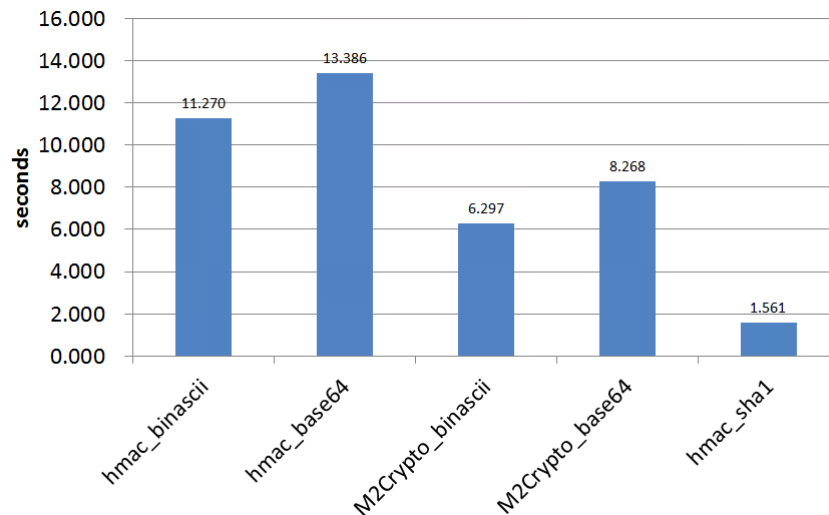


Figure 4.8: HMAC-SHA1 performance

of Base64 encoding by base64 and binascii module.

Listing 4.1: Compute HMAC-SHA1 by M2Crypto

```

>>> from M2Crypto.EVP import HMAC
>>> import base64
>>> import binascii
>>> hmac = HMAC('key', 'sha1')
>>> hmac.update('message')
>>> digest = hmac.digest()
>>> print base64.encodestring(digest)[: -1]
IIjfdNXyFGtIFGyvSWU3fp0L46Q=
>>> print binascii.b2a_base64(digest)[: -1]
IIjfdNXyFGtIFGyvSWU3fp0L46Q=

```

We calculate one million HMAC-SHA1 with the three implementations mentioned above. For standard library approach and M2Crypto approach, we do additional test for base64 and binascii module. As shown in Figure 4.8, our customized module is the fastest and the approach based on Python standard library is the slowest. Besides, the encoding performance of binascii module is better than base64 module due to the translation operations as described above.

4.2.4 Python Plurk API Library

We enhanced the plurk-oauth library for crawling performance. Performance are measured for the following four different concurrency models: single thread, multi-threading, multi-processing, and gevent. In this experiment, we randomly choose 1,000 plurkers and time the duration of crawling these users' profiles.

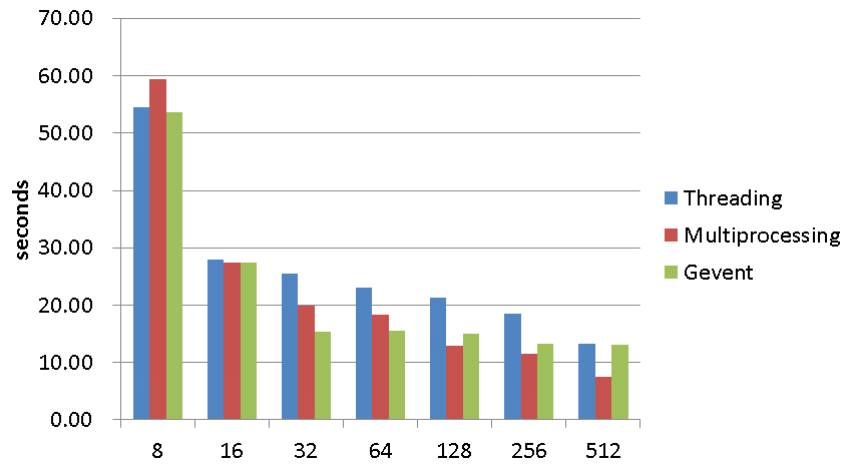


Figure 4.9: Original API library

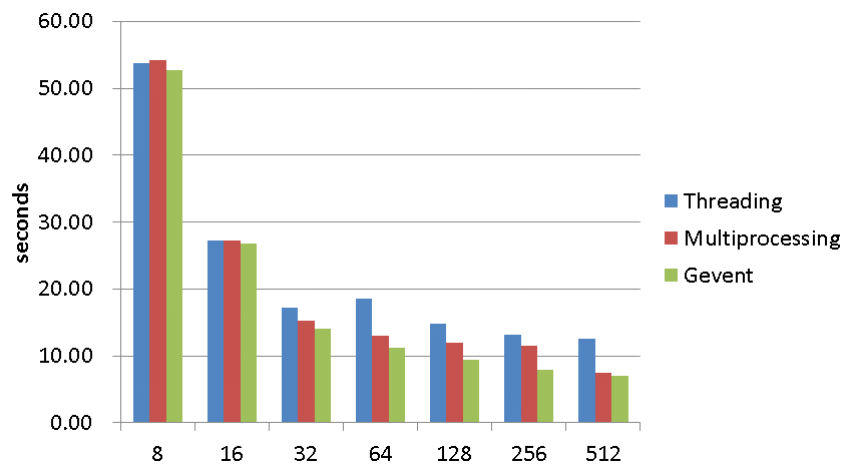


Figure 4.10: Enhanced API library

With original library, multi-processing model performs well since worker pool size larger than 16 and keeps the performance trend on increasing pool size. Gevent performs better than multi-threading model. However, the performance reaches a ceiling when pool size gets larger than 32, as shown in Figures 4.9 and 4.11.

Enhanced library improves about 2% than original library in single thread model, 6% in multi-threading model, 0.2% in multi-processing model, and 85.3% in gevent model, as shown in Figure 4.10.

4.2.5 Redis Connection

There are three binding modes for Redis: listening to all interfaces, local loopback (listen 127.0.0.1), and domain socket. According to the official documentation [35], domain socket is the fastest and local loopback is faster than listening to all interfaces. In this experiment, we

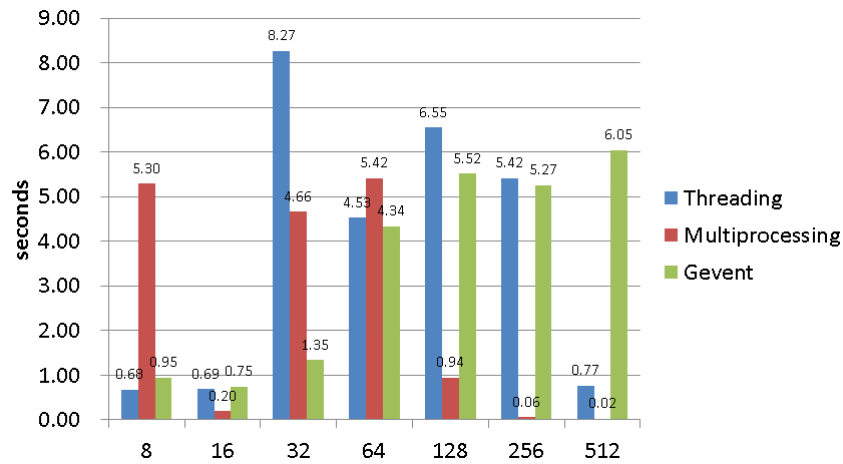


Figure 4.11: Improvements

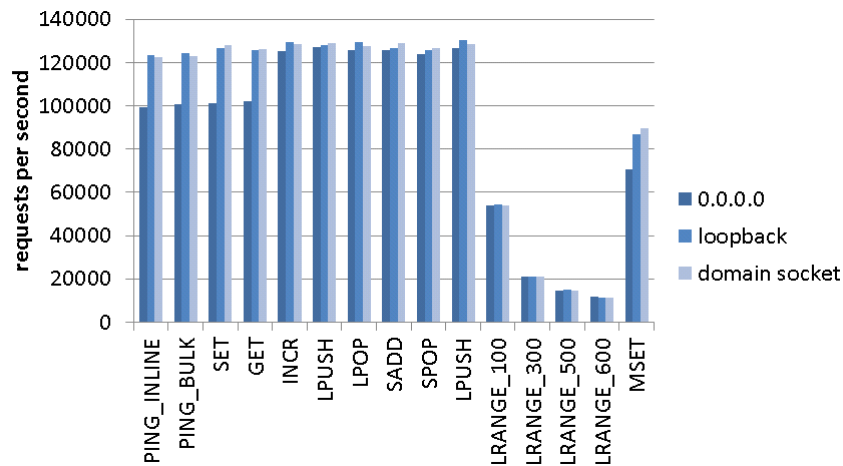


Figure 4.12: Redis binding modes

employ official benchmark tool: redis-benchmark and try to verify the performance for the three binding modes. Furthermore, due to security concern, it is recommended to make requests to Redis server via SSH tunnel, and we will also measure the connection overhead for both normal remote connection and SSH tunnel.

According to the benchmark results shown in Figure 4.12 and 4.13, domain socket and local loopback perform equally well, and both of them are much faster than listening to all interfaces mode. Besides, normal remote connection is about two times faster than SSH tunnel in the benchmark. Given the benchmark results, we connect Redis master and read-only slaves via SSH tunnel for lower traffic replication and synchronization. Read-only slaves accept requests via normal remote connections for higher traffic lookup and queries.

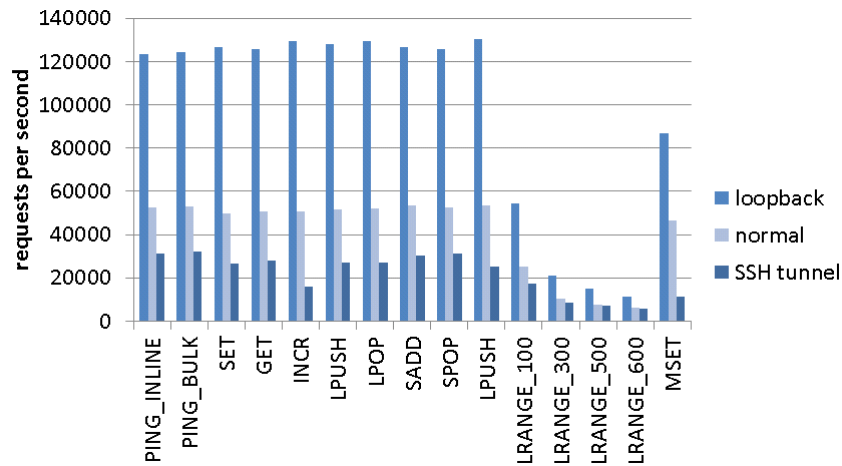


Figure 4.13: Redis remote connection types

4.3 Interest Derivation

In this section, we do an experiment to measure how many interest keywords might be guessed right for a private plurker.

Firstly, we randomly sample 100 public plurkers then aggregate their interest keywords directly from their public plurks and select top-64 frequent keywords to represent the plurker’s interests.

Secondly, we regard these plurkers as private and derive their interest keywords from communities. For each community, we select top-32 frequent keywords to represent the common topics for the community and aggregate top-64 frequent keywords to from communities to represent the plurker’s interests.

Finally, we calculate the number of matching interest keywords by counting intersections of results from the above two scenarios, as shown in Listing 4.2, to evaluate the precision about “guessing” interest keywords for a plurker. According to Figure 4.14, we can guess right about 36 keywords for these plurker, i.e. more than half of interest keywords are hit in this experiment.

Listing 4.2: Counting the cardinality of two sets

```
>>> def intersect(a, b):
...     return list(set(a) & set(b))
...
>>> public = {'snsd': 9, 'nctu': 2, 'helena': 21}
>>> private = {'snsd': 18, 'tts': 3, 'nctu': 1001}
>>> intersect(public, private)
['snsd', 'nctu']
```

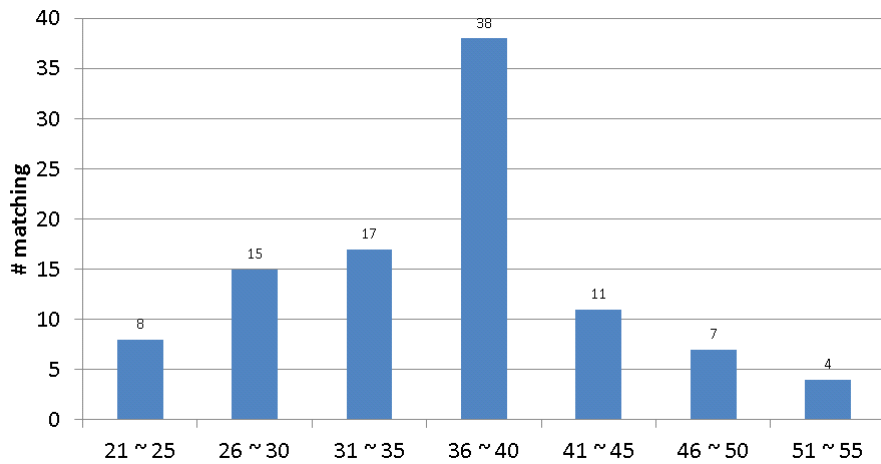


Figure 4.14: Result of interest derivation

4.4 Website Implementation

The website for SNSD system is based on Flask[7] web framework; besides, we employ Celery as task queue, D3.js[56] as visualization engine, and Twitter Bootstrap CSS framework [52] for this website. We provide three functionalities including: interest hierarchy view, interest tag cloud and focusable community view in pack and treemap layout.

Firstly, interest hierarchy view is rendered in tree layouts from D3.js. It includes customized script for making nodes collapsible. The interest hierarchy is collapsed when the page rendered. Website users are allowed to focus on particular hierarchy structure of the plurker by the collapsible function as shown in Figure 4.15.

Secondly, interest tag cloud is rendered in cloud layout by jasondavies[39]. This function is designed for website user to view plurker's interest keywords with frequency. The more frequent a keyword occurs, the bigger it will show in the cloud as in Figure 4.16.

Lastly, focusable community view is rendered in pack and treemap layout from D3.js. When communities for a plurker are rendered, the community view is zoomed out. That is, website user can view the community overview as shown in Figure 4.17 and Figure 4.18, and he/she can click a community to zoom in then focus on the community members as shown in Figure 4.19 and Figure 4.20.

Pack layout is suitable for browsing a plurker with less than 300 friends after filtering with parameter filter as shown in Figure 4.21. Treemap layout is intended for viewing a community with tens of members. Besides, this function can help plurkers find someone he/she might know but not his/her friend yet. Because the rendered communities are sampled by the snowball

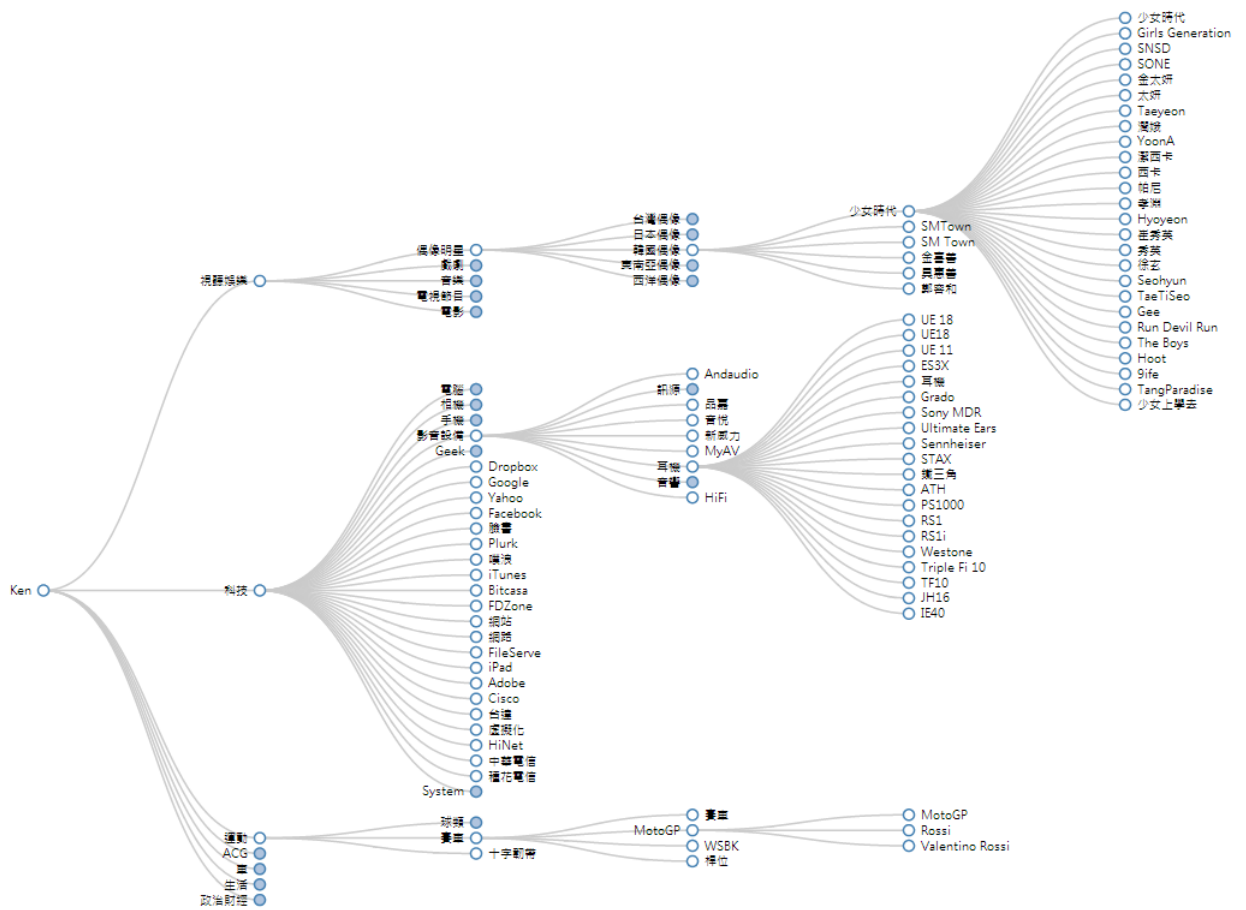


Figure 4.15: Interest keywords hierarchy

sampling algorithm with the plurker's friends and friends-of-friends, this sampling range could involve someone he/she might already know.

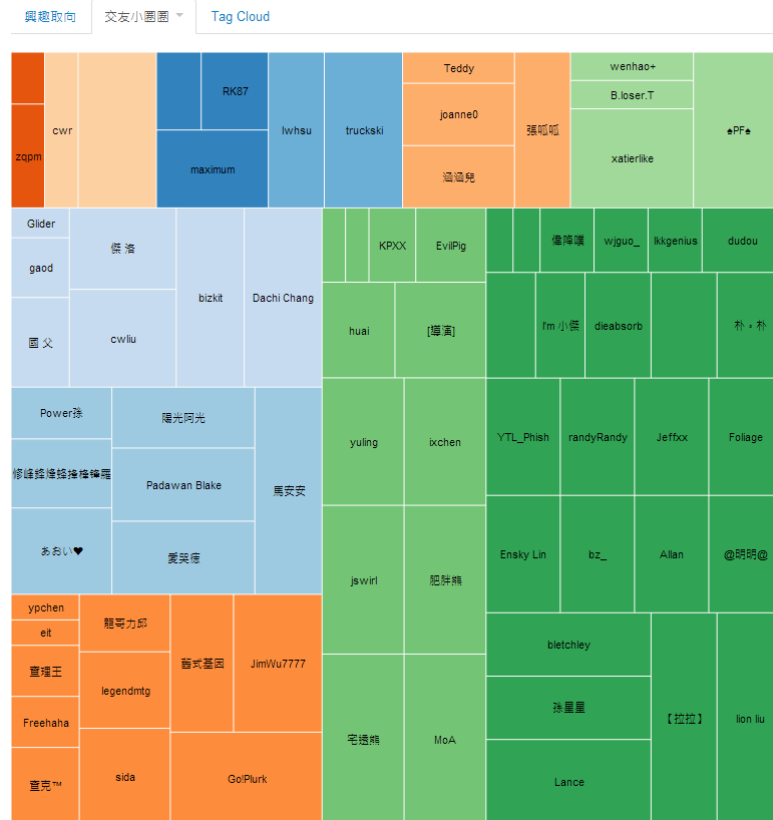


Figure 4.18: View communities in treemap layout

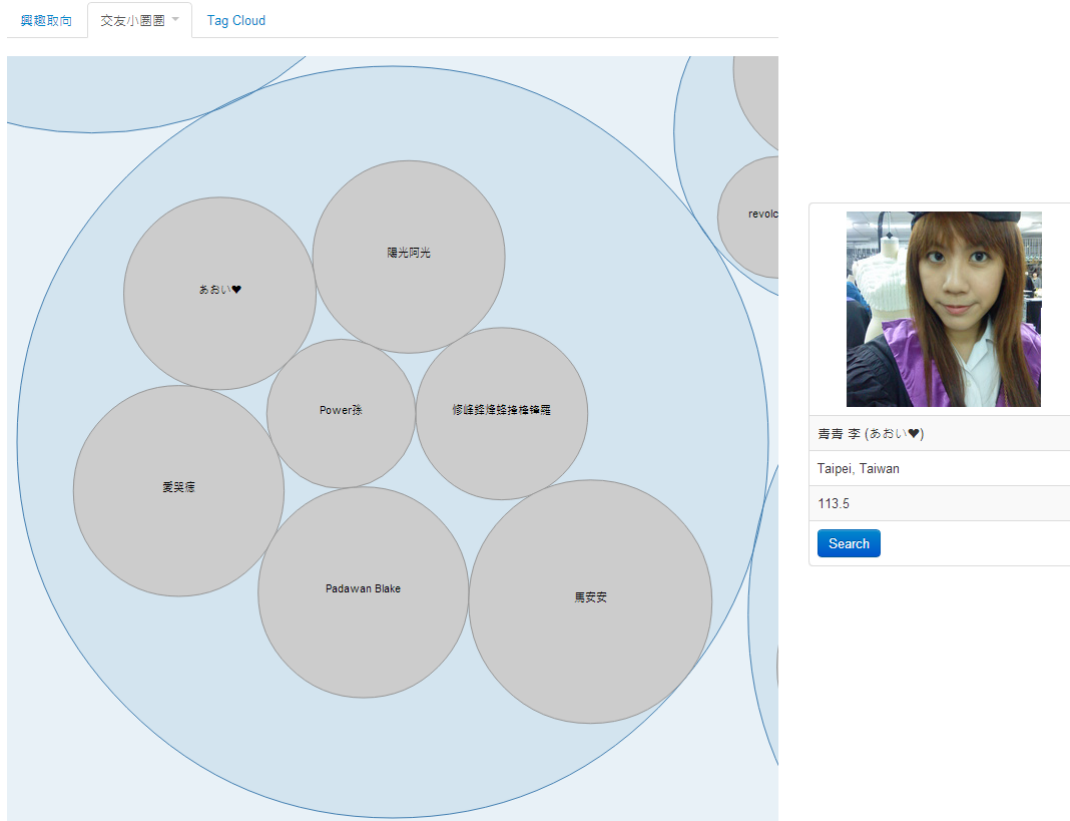


Figure 4.19: Focus community on pack layout

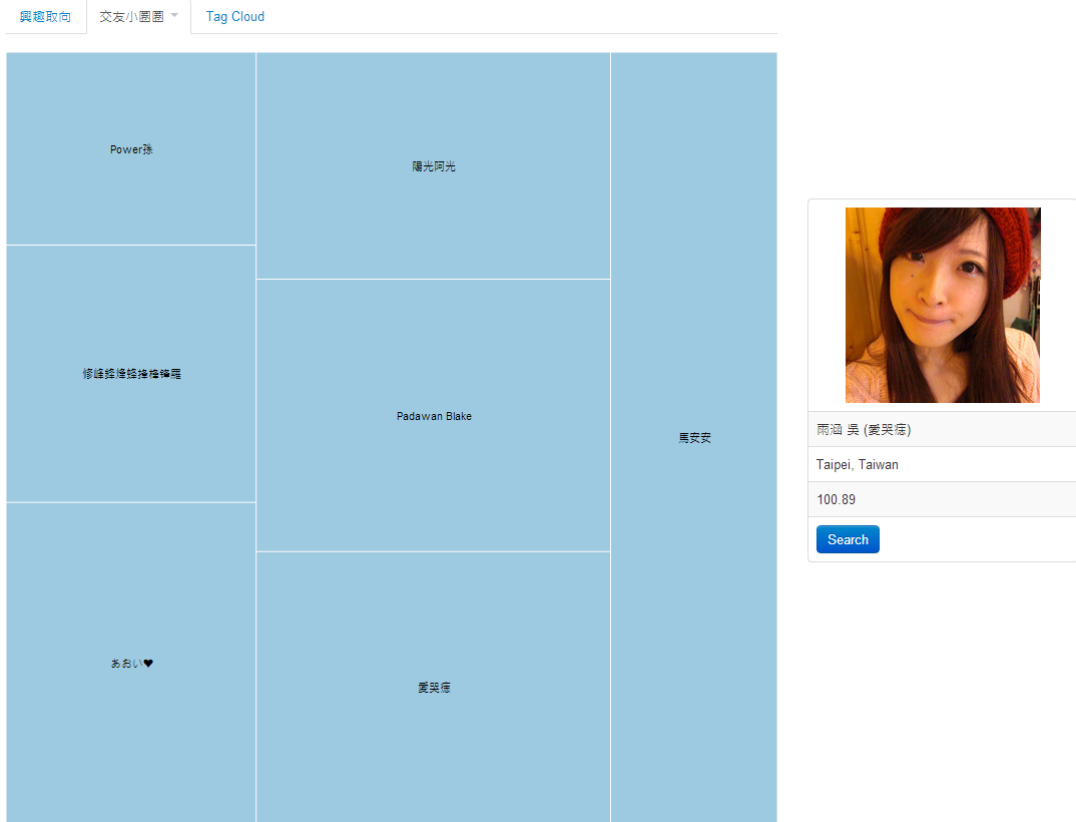


Figure 4.20: Focus community on treemap layout

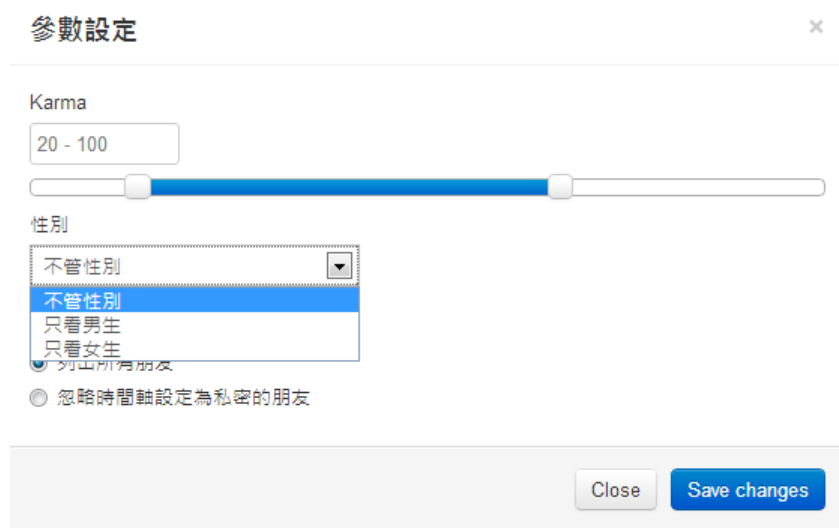


Figure 4.21: Parameter filter

Chapter 5

Conclusions and Future Works

We build an online SNSD system for Plurk users to find interest topics/keywords and relationship, develop a new crawling framework based on ZeroMQ, and patch the Plurk API for performance enhancements. Moreover, we build a website with Flask framework. Users can view the interests and relationship with a browser.

This system can be further expanded or enhanced in the following areas. First, to provide administrator with manageable UI to maintain the interest keyword hierarchy and consider synonyms, hypernyms and hyponyms for the hierarchy definition. For example, the term “SNSD”, “少女時代”, and “소녀시대” are the synonyms of “Girl’s Generation”, these terms should belong to only one category in the hierarchy instead of defining more than two categories with the same meaning.

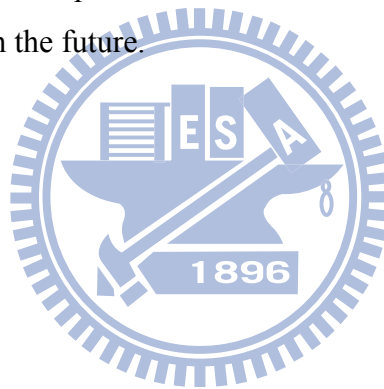
Second, apply the SNSD system to Twitter for western language and Sina weibo for mainland China, and consider fans relationship in interest derivation. With independent interest keywords hierarchy, our design allows the SNSD system to work with social networking services other than Plurk. What needs to do is to develop a new scraper for crawling conversation and relationship data from users and fulfill the interest keywords definitions in the hierarchy if the service is not using traditional Chinese characters.

Third, accelerate community detection with the algorithm proposed by Fortunato [26], Rosvall and Bergstrom [72]. The algorithm, based on random walk, achieved high performance in Fortunato’s test. Furthermore, if we still implement the algorithm by Rosvall and Bergstrom or refactor Louvain algorithm with NetworkX library, we might try to use Intel C compiler (ICC) and Math Kernel Library (MKL) to compile the SciPy and NumPy libraries which are employed by NetworkX for better performance [87].

Fourth, our customized Python C-extensions for accelerating crypto computation are based on OpenSSL. Though OpenSSL is already optimized by several hardware acceleration instructions such as SSE3, SSE4, AES-NI, or AVX etc., it is not fast enough. According to Intel's report [90, 34], the Intel Integrated Performance Primitives library is much faster than OpenSSL. The IPP can be considered to optimize our programs for higher throughput.

Fifth, consider user's interactivities in conversations for interest topics derivation. We currently only consider friend relationship to compute community partitions and derive user's interest topics. However, users will discuss with each other in a plurk/conversation or thread, we can utilize these information to derive user's interest topics and work as another filtering mechanism and derivation parameter.

Finally, Plurk provides poor searching function for users to search public and his/her own plurks. For the SNSD system, we have already stored most of public plurks, and we may allow plurkers to provide his/her private plurks to us via Plurk OAuth API and our system may serve as a full-text search engine in the future.



Bibliography

- [1] 10gen, Inc. “MongoDB”, 2012. Available: <http://www.mongodb.org>
- [2] 10gen, Inc. “PyMongo Documentation”, 2012. Available: <http://api.mongodb.org/python/current/>
- [3] Adams, P. (2011). *Grouped: How Small Groups of Friends are the Key to Influence on the Social Web*, New Riders Press.
- [4] Alexa Internet, Inc. “Plurk.com Site Info”, 2012. Available: <http://www.alexa.com/site-info/plurk.com>
- [5] Andrey Petrov. “shazow/urllib3”, 2012. Available: <https://github.com/shazow/urllib3>
- [6] Arenas, A., J. Duch, et al. (2007). “Size reduction of complex networks preserving modularity.” *New Journal of Physics* 9(6): 176.
- [7] Armin Ronacher. “Flask (A Python Microframework)”, 2012. Available: <http://flask.pocoo.org>
- [8] Ask Solem. “Homepage | Celery: Distributed Task Queue”, 2012. Available: <http://celeryproject.org>
- [9] Banker, K. (2011). *MongoDB in Action*, Manning Pubs Co Series. Manning Publications.
- [10] Beazley, D. (2010). “An Introduction to Python Concurrency.” from <http://www.slideshare.net/dabeaz/an-introduction-to-python-concurrency>.
- [11] Beazley, D. M. (2009). *Python essential reference*, Addison-Wesley Professional.
- [12] Blood, R. (2000). “Weblogs: a history and perspective.” *Rebecca’s Pocket* 7(9): 2000.

- [13] Bob Ippolito. “simplejson/simplejson”, 2012. Available: <http://github.com/simplejson/simplejson>
- [14] Brandes, U., D. Delling, et al. (2006). “Maximizing modularity is hard.” arXiv preprint physics/0608255.
- [15] Chau, D. H., S. Pandit, et al. (2007). Parallel crawling for online social networks. Proceedings of the 16th international conference on World Wide Web. Banff, Alberta, Canada, ACM: 1283-1284.
- [16] Cheng, A., M. Evans, et al. (2009). ”Inside Twitter: An in-depth look inside the Twitter world.” Unpublished report by Sysomos, inc.
- [17] Cheng-Lung Sung. “clsung/plurk-oauth”, 2012. Available: <https://github.com/clsung/plurk-oauth>
- [18] Chen, K. J. and S. H. Liu (1992). Word identification for Mandarin Chinese sentences. Proceedings of the 14th conference on Computational linguistics-Volume 1, Association for Computational Linguistics.
- [19] Cho, J. and H. Garcia-Molina (2002). Parallel crawlers. Proceedings of the 11th international conference on World Wide Web. Honolulu, Hawaii, USA, ACM: 124-135.
- [20] Conway, D. (2010). “Mining and Analyzing Online Social Graph Data.Students” from <http://www.drewconway.com/zia/?p=2151>.
- [21] Crockford, D. (2006). ‘The application/json media type for javascript object notation (json).’
- [22] Dan Pascu. “python-cjson 1.0.5”, 2012. Available: <http://pypi.python.org/pypi/python-cjson>
- [23] Denis Bilenko. “gevent: A coroutine-based network library for PythonSpeed”, 2012. Available: <http://www.gevent.org>
- [24] D’Monte, L. (2011). “Swine flu’s tweet tweet causes online flutter.” Business Standard.
- [25] Ellison, N. B., C. Steinfield, et al. (2007). ‘The Benefits of Facebook “Friends:” Social Capital and College Students’ Use of Online Social Network Sites.’ Journal of Computer-Mediated Communication 12(4): 1143-1168.

- [26] Fortunato, S. (2010). "Community detection in graphs." *Physics Reports* 486(3): 75-174.
- [27] Gaonkar, S., J. Li, et al. (2008). *Micro-Blog: sharing and querying content through mobile phones and social participation*, ACM.
- [28] Gaudeul, A. and C. Peroni (2010). "Reciprocal attention and norm of reciprocity in blogging networks." *Economics Bulletin* 30(3): 2230-2248.
- [29] Gentoo Foundation, Inc. "Gentoo Linux – Gentoo Linux News", 2012. Available: <http://www.gentoo.org>
- [30] Girvan, M. and M. E. J. Newman (2002). "Community structure in social and biological networks." *Proceedings of the National Academy of Sciences* 99(12): 7821-7826.
- [31] Google (2011). *Protocol Buffers: Google's Data Interchange Format*.
- [32] Guido van Rossum, P. J. E. (2005). "Coroutines via Enhanced Generators." from <http://www.python.org/dev/peps/pep-0342>
- [33] Guido van Rossum, T. P. (2003). "Extensions to the pickle protocol." from <http://www.python.org/dev/peps/pep-0307>
- [34] Gururaj Nagendra. "Boosting Cryptography Performance with Intel®Libraries", 2012. Available: <http://software.intel.com/en-us/articles/boosting-cryptography-performance-with-intel-libraries>
- [35] "How fast is Redis? – Redis", 2012. Available: <http://redis.io/topics/benchmarks>
- [36] Hughes, D. J., M. Rowe, et al. (2012). "A tale of two sites: Twitter vs. Facebook and the personality predictors of social media usage." *Computers in Human Behavior* 28(2): 561-569.
- [37] iMatix Corporation. "The Intelligent Transport Layer - zeromq", 2012. Available: <http://www.zeromq.org>
- [38] Jansen, B. J., M. Zhang, et al. (2009). "Twitter power: Tweets as electronic word of mouth." *Journal of the American society for information science and technology* 60(11): 2169-2188.

- [39] Jason Davies. “jasondavies/d3-cloud”, 2012. Available: <https://github.com/jasondavies/d3-cloud>
- [40] Jesse Noller, R. O. (2008). “Addition of the multiprocessing package to the standard library.” from <http://www.python.org/dev/peps/pep-0371>.
- [41] John Millikin. “jsonlib in Launchpad”, 2012. Available: <https://launchpad.net/jsonlib>
- [42] Jonas Tärnström “esnme/ultrajson”, 2012. Available: <http://github.com/esnme/ultrajson>
- [43] “JSON”, 2012. Available: <http://json.org>
- [44] Judd Vinet, Aaron Griffin. “Arch Linux”, 2012. Available: <http://www.archlinux.org>
- [45] Karwin, B. and J. Carter (2010). SQL Antipatterns: Avoiding the Pitfalls of Database Programming, Pragmatic Bookshelf.
- [46] Kegel, D. (2006). “The C10K problem.” from <http://www.kegel.com/c10k.html>.
- [47] Ken Lee, Bryan Cheng, Sean Lee. “Go!Plurk 喋浪興趣分析機”, 2012. Available: <http://goplurk.cse.tw>
- [48] Krawczyk, H., M. Bellare, et al. (1997). RFC 2104: HMAC: Keyed-hashing for message authentication, IETF, February.
- [49] Kwak, H., C. Lee, et al. (2010). What is Twitter, a social network or a news media? Proceedings of the 19th international conference on World wide web. Raleigh, North Carolina, USA, ACM: 591-600.
- [50] Lin, V. (2010). “Talk about Coroutine and Gevent.” from <http://blog.ez2learn.com/2010/07/17/talk-about-coroutine-and-gevent>
- [51] Lloyd Hilaiel. “yajl”, 2012. Available: <http://lloyd.github.com/yajl>
- [52] Mark Otto, Angus Droll. “Bootstrap”, 2012. Available: <http://twitter.github.com/bootstrap>
- [53] Matthew Perry. “A quick Cython introduction”, 2012. Available: <http://blog.perrygeo.net/2008/04/19/a-quick-cython-introduction>

- [54] Ma, W. Y. and K. J. Chen (2003). Introduction to CKIP Chinese word segmentation system for the first international Chinese Word Segmentation Bakeoff, Association for Computational Linguistics.
- [55] Martin Sústrik. “ØMQ: The Theoretical Foundation - 250bpm”, 2012. Available: <http://d3js.org>
- [56] Michael Bostock. “D3.js - Data-Driven Documents”, 2012. Available: <http://d3js.org>
- [57] Mikhail Korobov. “marisa-trie 0.3.7”, 2012. Available: <http://pypi.python.org/pypi/marisa-trie>
- [58] National Digital Archives Program, Taiwan. “中文斷詞系統”, 2012. Available: <http://ckipsvr.iis.sinica.edu.tw>
- [59] Newman, M. E. J. and M. Girvan (2004). “Finding and evaluating community structure in networks.” *Physical Review E* 69(2): 026113.
- [60] Newman, M. E. J. (2004). “Fast algorithm for detecting community structure in networks.” *Physical Review E* 69(6): 066133.
- [61] Newman, M. E. J. (2004). “Analysis of weighted networks.” *Physical Review E* 70(5): 056131.
- [62] Newman, M. E. J. (2006). “Modularity and community structure in networks.” *Proceedings of the National Academy of Sciences* 103(23): 8577-8582.
- [63] Nick Mathewson, Niels Provos. “libevent”, 2012. Available: <http://libevent.org>
- [64] OAuth Core Workgroup. “OAuth Core 1.0a”, 2012. Available: <http://oauth.net/core/1.0a>
- [65] OASIS. “AMQP”, 2012. Available: <http://www.amqp.org>
- [66] O’Higgins, N. (2011). *MongoDB and Python: Patterns and Processes for the Popular Document-oriented Database*, O’Reilly Media.
- [67] Plurk Inc. “Plurk API 2.0”, 2012. Available: <http://www.plurk.com/API>
- [68] Python Software Foundation. “18.2. json —JSON encoder and decoder — Python v2.7.3 documentation”, 2012. Available: <http://docs.python.org/library/json.html>

- [69] Python Software Foundation. ‘1. Extending Python with C or C++ — Python v2.7.3 documentation’, 2012. Available: <http://docs.python.org/2/extending/extending.html>
- [70] Ralf Schmitt. “python-greenlet/greenlet”, 2012. Available: <https://github.com/python-greenlet/greenlet>
- [71] Romm-Livermore, C. and K. Setzekorn (2009). Social networking communities and e-dating services: Concepts and implications, IGI Global.
- [72] Rosvall, M. and C. T. Bergstrom (2008). “Maps of random walks on complex networks reveal community structure.” Proceedings of the National Academy of Sciences 105(4): 1118-1123.
- [73] Rune Halvorsen. “runeh / anyjson”, 2012. Available: <https://bitbucket.org/runeh/anyjson>
- [74] Russell, M. (2011). Mining the Social Web: Analyzing Data from Facebook, Twitter, LinkedIn, and Other Social Media Sites, O’Reilly Media, Inc.
- [75] Russell, M. A. (2011). 21 Recipes for Mining Twitter, Oreilly & Associates Inc.
- [76] Sadayuki Furuhashi. “MessagePack: It’s like JSON. but fast and small.”, 2012. Available: <http://msgpack.org>
- [77] Salvatore Sanfilippo, Pieter Noordhuis. “Redis”, 2012. Available: <http://redis.io>
- [78] “Sharded Cluster Architectures — MongoDB Manual”, 2012. Available: <http://docs.mongodb.org/manual/administration/sharding-architectures/>
- [79] SimpleGeo Inc. “simplegeo/python-oauth2”, 2012. Available: <https://github.com/simplegeo/python-oauth2>
- [80] Tang, L. and H. Liu (2010). “Community detection and mining in social media.” Synthesis Lectures on Data Mining and Knowledge Discovery 2(1): 1-137.
- [81] The FreeBSD Project. “The FreeBSD Project”, 2012. Available: <http://www.freebsd.org>
- [82] Thomas Aynaud. “taynaud / python-louvain”, 2012. Available: <https://bitbucket.org/taynaud/python-louvain>
- [83] Thomas Broyer. “httplib2 - A comprehensive HTTP client library in Python”, 2012. Available: <http://code.google.com/p/httplib2>

- [84] Tsai, C. H. (2000). “MMSEG: A word identification system for Mandarin Chinese text based on two variants of the maximum matching algorithm.” Available on internet at <http://technology.chtsai.org/mmseg>.
- [85] Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. “Fast unfolding of community hierarchies in large networks” arXiv (2008): 0803.0476.
- [86] Vinta. “gibuloto/paranoid-auto-spacing”, 2012. Available: <https://github.com/gibuloto/paranoid-auto-spacing>
- [87] Vipin Kumar E K. “Numpy/Scipy with Intel®MKL”, 2012. Available: <http://software.intel.com/en-us/articles/numpyscipy-with-intel-mkl>
- [88] Wilson, C., B. Boe, et al. (2009). User interactions in social networks and their implications. Proceedings of the 4th ACM European conference on Computer systems, ACM.
- [89] Wu, S., J. M. Hofman, et al. (2011). Who says what to whom on twitter. Proceedings of the 20th international conference on World wide web. Hyderabad, India, ACM: 705-714.
- [90] Yang C. “Boosting OpenSSL AES Encryption with Intel®IPP”, 2012. Available: <http://software.intel.com/en-us/articles/boosting-openssl-aes-encryption-with-intel-ipp>
- [91] Yata Susumu. “marisa-trie - Matching Algorithm with Recursively Implemented Storage”, 2012. Available: <https://code.google.com/p/marisa-trie>
- [92] Zhao, D. and M. B. Rosson (2009). How and why people Twitter: the role that microblogging plays in informal communication at work, ACM.
- [93] Zuroc. “czjson 1.0.8”, 2012. Available: <http://pypi.python.org/pypi/czjson>
- [94] 不來恩. “GoPlurk! 噗浪「興趣分析機」, 看看你是哪種噗浪人!”, 2012. Available: <http://briian.com/?p=6331>
- [95] 楊又肇. “「Go!Plurk」分析你的噗浪興趣比例”, 2012. Available: http://mag.udn.com/mag/digital/storypage.jsp?f_ART_ID=199431
- [96] “BSON - Binary JSON”, 2012. Available: <http://bsonspec.org>
- [97] “Encryption - zeromq”, 2012. Available: <http://www.zeromq.org/topics/encryption>

- [98] “PyPy”, 2012. Available: <http://pypy.org>
- [99] “Overview — NetworkX 1.7 documentation”, 2012. Available: <http://networkx.lanl.gov>
- [100] “Python Programming/Extending with C - Wikibooks, open books for an open world”, 2012. Available: http://en.wikibooks.org/wiki/Python_Programming/Extending_with_C
- [101] “PythonSpeed/ PerformanceTips - PythonInfo Wiki”, 2012. Available: <http://wiki.python.org/moin/PythonSpeed/PerformanceTips>
- [102] “Twisted”, 2012. Available: <http://twistedmatrix.com/trac>
- [103] “國立交通大學資訊工程學系 NCTU Department of Computer Science”, 2012. Available: http://www.cs.nctu.edu.tw/cht/about_cs/index1.php



Appendix A

Diskless Linux Cluster Installation

A.1 Base System

Setup network in the live installation environment

```
# ip addr add 140.113.207.149/24 dev eth0
# ip link set up eth0
# ip route add default via 140.113.207.254
# echo nameserver 8.8.8.8 > /etc/resolv.conf.head
# passwd
# /etc/rc.d/sshd start
```

Prepare hard drive

```
# fdisk -l
# cfdisk /dev/sda
# mkfs.ext4 /dev/sda1
# mkfs.btrfs /dev/sda3
# mkswap /dev/sda2 && swapon /dev/sda2
```

Mount the partitions

```
# mount /dev/sda3 /mnt
# mkdir -p /mnt/boot
# mount /dev/sda1 /mnt/boot
```

Select installation mirror

```
# mkdir -p /mnt/etc/pacman.d
# echo "Server = http://linux.cs.nctu.edu.tw/archlinux/"\$repo"/os/"\$arch" > /mnt/etc/pacman.d/
mirrorlist
```

Install the base system and bootloader

```
# pacstrap /mnt base base-devel syslinux btrfs-progs
```

Generate fstab then chroot into system

```
# genfstab -p /mnt >> /mnt/etc/fstab
# arch-chroot /mnt
```

Miscellaneous configurations

```
# sed -i 's/PermitRootLogin yes/PermitRootLogin no/g' /etc/ssh/sshd_config

# cat >> /etc/pacman.conf <<EOF
[archlinuxfr]
Server = http://repo.archlinux.fr/\$arch
EOF

# cat > /etc/ntp.conf <<EOF
server tick.stdtime.gov.tw prefer
server tock.stdtime.gov.tw prefer
server time.stdtime.gov.tw prefer
server clock.stdtime.gov.tw
server watch.stdtime.gov.tw
restrict default nomodify nopeer
restrict 127.0.0.1
driftfile /var/lib/ntp/ntp.drift
logfile /var/log/ntp.log
EOF

# echo "HOSTNAME" > /etc/hostname

# cat > /etc/vconsole.conf <<EOF
KEYMAP="us"
CONSOLEFONT=
CONSOLEMAP=
USECOLOR="yes"
EOF

# ln -s /usr/share/zoneinfo/Asia/Taipei /etc/localtime

# cat >> /etc/locale.gen <<EOF
en_US.UTF-8 UTF-8
```

```
zh_TW.UTF-8 UTF-8
zh_TW BIG5
EOF

# locale-gen

# echo "LANG=en_US.UTF-8" > /etc/locale.conf
```

Configure the network (static IP)

```
# cat > /etc/resolv.conf <<EOF
nameserver 8.8.8.8
nameserver 8.8.4.4
nameserver 140.113.235.1
EOF

# chattr +i /etc/resolv.conf

# pacman -S netcfg ifplugd
# cat > /etc/network.d/ethernet-static <<EOF
CONNECTION='ethernet'
INTERFACE='eth0'
IP='static'
ADDR='140.113.207.147'
GATEWAY='140.113.207.254'
DNS=('8.8.8.8', '8.8.4.4', '140.113.235.1')
EOF

# systemctl enable net-auto-wired.service

## disabled Ctrl+Alt+Del to reboot
systemctl mask ctrl-alt-del.target

## Enable DAEMONS
systemctl enable cronic.service ntpd.service sshd.service iptables.service syslog-ng.service

# iptables -N sshguard
# iptables -A INPUT -p tcp --dport 22 -j sshguard
# iptables-save > /etc/iptables/iptables.rules
# systemctl restart sshguard.service
```

Create an initial ramdisk environment

```
# sed -i 's/MODULES=""/MODULES="virtio_blk virtio_pci virtio_net"/g' /etc/mkinitcpio.conf
# mkinitcpio -p linux
```

Configure the bootloader

```
# /usr/sbin/syslinux-install_update -iam
# vi /boot/syslinux/syslinux.cfg
```

Root password and adding a user

```
# passwd
# adduser
# useradd -m -g wheel -G root,log -s /bin/bash ken
```

Clean up then reboot

```
# rm -rf /var/log/*
# rm /var/cache/pacman/pkg/*
# exit
# rm /mnt/root/.bash_history
# umount /mnt/boot /mnt
# reboot
```

A.2 Network Block Device (NBD) Server

Install NBD

```
# pacman -S nbd
```

Create NBD Image

```
# mkdir /nbd
# truncate -s 4G /nbd/root
# mkfs.ext4 /nbd/root
# mount /nbd/root /mnt
# export root=/mnt
# mkdir -p $root/{proc,sys,run,tmp}
# mkdir -p $root/dev/{pts,shm}
# mkdir -p "$root/var/lib/pacman"
# mount -t proc proc "$root/proc" -o nosuid,noexec,nodev
# mount -t sysfs sys "$root/sys" -o nosuid,noexec,nodev
# mount -t devtmpfs udev "$root/dev" -o mode=0755,nosuid
# mount -t devpts devpts "$root/dev/pts" -o mode=0620,gid=5,nosuid,noexec
# pacman -Syu --root "$root" --dbpath "$root/var/lib/pacman" base base-devel --arch x86_64

# sed -i 's/^HOOKS=".*"/HOOKS="base udev net nbd autodetect pata scsi sata mdadm lvm2 filesystems us-
binput fsck/g' "$root/etc/mkinitcpio.conf"
```

```
# echo 'NETWORK_PERSIST="yes"' > "$root/etc/rc.conf"

# chroot "$root" /bin/bash
# (chroot) # mkinitcpio -p linux
# (chroot) # exit

# umount "$root/dev/pts" "$root/dev" "$root/sys" "$root/proc"
```

/etc/nbd-server/config

```
[generic]
    user = nbd
    group = nbd
[nbdroot]
    exportname = /nbd/root
    authfile = /etc/nbd-server/allow
    copyonwrite = true
    postrun = rm -f %s
```

/etc/nbd-server/allow

```
192.168.1.2
192.168.1.3
192.168.1.4
192.168.1.5
192.168.1.6
192.168.1.7
192.168.1.8
192.168.1.9
192.168.1.10
192.168.1.11
192.168.1.12
192.168.1.13
192.168.1.14
192.168.1.15
192.168.1.16
192.168.1.17
192.168.1.18
192.168.1.19
192.168.1.20
```

Install PxeLinux

```
# pacman -S syslinux

# mkdir -p /nbd/boot/pxelinux.cfg
# cp /usr/lib/syslinux/pxelinux.0 /nbd/boot

# mount /nbd/root /mnt
# cp -r /mnt/boot /nbd/boot
# umount /mnt
```


/nbd/boot/pxelinux.cfg/default

```
default linux

label linux
kernel vmlinuz-linux
append initrd=initramfs-linux.img ip=:::::eth0:dhcp nbd_host=192.168.1.1 nbd_name=nbdroot root=/dev/nbd0
```

A.3 DHCP and PXE Server

Install DNSMasq

```
# pacman -S dnsmasq
```

/etc/dnsmasq.conf

```
interface=eth1
bind-interfaces
dhcp-range=192.168.1.2,192.168.1.20,12h
read-ethers
dhcp-option-force=208,f1:00:74:7e
dhcp-option-force=209,configs/common
dhcp-option-force=210,/nbd/boot/
dhcp-boot=pxelinux.0
enable-tftp
tftp-root=/nbd/boot/
```

/etc/ethers

00:16:e6:50:a6:70	192.168.1.2
00:16:e6:4d:e3:ea	192.168.1.3
00:16:e6:50:a8:ce	192.168.1.4
00:16:e6:50:be:d2	192.168.1.5
00:14:85:e8:e5:04	192.168.1.6
00:16:e6:4d:dc:b6	192.168.1.7
00:14:85:f0:7c:5c	192.168.1.8
00:16:e6:4f:7f:cc	192.168.1.9
00:16:e6:4e:db:08	192.168.1.10
00:16:e6:5b:ba:3a	192.168.1.11
00:16:e6:55:02:8e	192.168.1.12
00:16:e6:4d:e3:ec	192.168.1.13
00:16:e6:4e:d0:04	192.168.1.14
00:16:e6:52:ff:8e	192.168.1.15
00:16:e6:51:39:12	192.168.1.16
00:16:e6:51:fb:3a	192.168.1.17
00:16:e6:4e:d0:a0	192.168.1.18
00:14:85:ed:55:24	192.168.1.19
00:14:85:f3:c1:e4	192.168.1.20

Appendix B

MongoDB Cluster Installation

B.1 MongoDB Installation

Install MongoDB and numactl

```
# pacman -S mongodb numactl
```

Make directories for MongoDB

```
# mkdir -p /data/mongodb/configsvr
# mkdir -p /data/mongodb/log
# chown -R mongodb:daemon /data/mongodb
```

Creat empty logfile then change owner and group

```
# touch /data/mongodb/log/shardsvr.log /data/mongodb/log/configsvr.log
# chown -R mongodb:daemon /data/mongodb
```

Disable hugepage

```
# echo madvise > /sys/kernel/mm/transparent_hugepage/enabled
# echo madvise > /sys/kernel/mm/transparent_hugepage/defrag
# echo 0 > /sys/kernel/mm/transparent_hugepage/khugepaged/defrag
```

/etc/hosts

```
#<ip-address> <hostname.domain.org> <hostname>
127.0.0.1 localhost
::1 localhost

192.168.1.1 master loghost SNSD
192.168.1.2 slave0 TaeYeon
```

192.168.1.3	slave1	Jessica
192.168.1.4	slave2	Sunny
192.168.1.5	slave3	Tiffany
192.168.1.6	slave4	HyoYeon
192.168.1.7	slave5	YuRi
192.168.1.8	slave6	SooYoung
192.168.1.9	slave7	YoonA
192.168.1.10	slave8	SeoHyun
192.168.1.11	slave9	SONE
192.168.1.12	slave10	KimTaeYeon
192.168.1.13	slave11	JungSooYeon
192.168.1.14	slave12	LeeSoonKyu
192.168.1.15	slave13	HwangMiYoung
192.168.1.16	slave14	KimHyoYeon
192.168.1.17	slave15	KwonYuRi
192.168.1.18	slave16	ChoiSooYoung
192.168.1.19	slave17	ImYoonA
192.168.1.20	slave18	SeoJuHyun

/etc/mongodb/configsvr.conf

```

configsvr = true
quiet = true
dbpath = /data/mongodb/configsvr
logpath = /data/mongodb/log/configsvr.log
logappend = true
rest = true

```

/etc/mongodb/mongos.conf

```

quiet = true
logpath = /data/mongodb/log/mongos.log
logappend = true

```

/etc/mongodb/shardsvr.conf

```

shardsvr = true
quiet = true
dbpath = /data/mongodb
logpath = /data/mongodb/log/shardsvr.log
logappend = true
oplogSize = 100
rest = true

```

Start config servers

```
snsd# su mongod -c "numactl --interleave=all mongod -f /etc/mongodb/configsvr.conf --rest --fork"  
sone# su mongod -c "numactl --interleave=all mongod -f /etc/mongodb/configsvr.conf --rest --fork"  
taeyeon# su mongod -c "numactl --interleave=all mongod -f /etc/mongodb/configsvr.conf --rest --fork"
```

Start mongos

```
snsd# su mongod -c "mongos -f /etc/mongodb/mongos.conf --configdb snsd:27019,sone:27019,taeyeon:27019 --fork"  
sone# su mongod -c "mongos -f /etc/mongodb/mongos.conf --configdb sone:27019,taeyeon:27019,snsd:27019 --fork"  
taeyeon# su mongod -c "mongos -f /etc/mongodb/mongos.conf --configdb taeyeon:27019,snsd:27019,sone:27019 --fork"  
kimtaeyeon# su mongod -c "mongos -f /etc/mongodb/mongos.conf --configdb taeyeon:27019,snsd:27019,sone:27019 --fork"
```



Start arbiters on master

```
snsd# su mongod -c "numactl --interleave=all mongod -f /etc/mongodb/shardsvr.conf --replSet rs0 --bind_ip 192.168.1.1 --port 27020 --dbpath /data/mongodb/  
arbiter/rs0 --rest --fork"  
snsd# su mongod -c "numactl --interleave=all mongod -f /etc/mongodb/shardsvr.conf --replSet rs1 --bind_ip 192.168.1.1 --port 27021 --dbpath /data/mongodb/  
arbiter/rs1 --rest --fork"  
snsd# su mongod -c "numactl --interleave=all mongod -f /etc/mongodb/shardsvr.conf --replSet rs2 --bind_ip 192.168.1.1 --port 27022 --dbpath /data/mongodb/  
arbiter/rs2 --rest --fork"  
snsd# su mongod -c "numactl --interleave=all mongod -f /etc/mongodb/shardsvr.conf --replSet rs3 --bind_ip 192.168.1.1 --port 27023 --dbpath /data/mongodb/  
arbiter/rs3 --rest --fork"  
snsd# su mongod -c "numactl --interleave=all mongod -f /etc/mongodb/shardsvr.conf --replSet rs4 --bind_ip 192.168.1.1 --port 27024 --dbpath /data/mongodb/  
arbiter/rs4 --rest --fork"  
snsd# su mongod -c "numactl --interleave=all mongod -f /etc/mongodb/shardsvr.conf --replSet rs5 --bind_ip 192.168.1.1 --port 27025 --dbpath /data/mongodb/  
arbiter/rs5 --rest --fork"  
snsd# su mongod -c "numactl --interleave=all mongod -f /etc/mongodb/shardsvr.conf --replSet rs6 --bind_ip 192.168.1.1 --port 27026 --dbpath /data/mongodb/  
arbiter/rs6 --rest --fork"  
snsd# su mongod -c "numactl --interleave=all mongod -f /etc/mongodb/shardsvr.conf --replSet rs7 --bind_ip 192.168.1.1 --port 27027 --dbpath /data/mongodb/  
arbiter/rs7 --rest --fork"
```

```

arbiter/rs7 --rest --fork"
sns# su mongod -c "numactl --interleave=all mongod -f /etc/mongoddb/shardsvr.conf --replSet rs8 --bind_ip 192.168.1.1 --port 27028 --dbpath /data/mongoddb/
arbiter/rs8 --rest --fork"

```

Start shard servers

```

slave0# su mongod -c "numactl --interleave=all mongod -f /etc/mongoddb/shardsvr.conf --replSet rs0 --bind_ip 192.168.1.2 --fork"
slave1# su mongod -c "numactl --interleave=all mongod -f /etc/mongoddb/shardsvr.conf --replSet rs1 --bind_ip 192.168.1.3 --fork"
slave2# su mongod -c "numactl --interleave=all mongod -f /etc/mongoddb/shardsvr.conf --replSet rs2 --bind_ip 192.168.1.4 --fork"
slave3# su mongod -c "numactl --interleave=all mongod -f /etc/mongoddb/shardsvr.conf --replSet rs3 --bind_ip 192.168.1.5 --fork"
slave4# su mongod -c "numactl --interleave=all mongod -f /etc/mongoddb/shardsvr.conf --replSet rs4 --bind_ip 192.168.1.6 --fork"
slave5# su mongod -c "numactl --interleave=all mongod -f /etc/mongoddb/shardsvr.conf --replSet rs5 --bind_ip 192.168.1.7 --fork"
slave6# su mongod -c "numactl --interleave=all mongod -f /etc/mongoddb/shardsvr.conf --replSet rs6 --bind_ip 192.168.1.8 --fork"
slave7# su mongod -c "numactl --interleave=all mongod -f /etc/mongoddb/shardsvr.conf --replSet rs7 --bind_ip 192.168.1.9 --fork"
slave8# su mongod -c "numactl --interleave=all mongod -f /etc/mongoddb/shardsvr.conf --replSet rs8 --bind_ip 192.168.1.10 --fork"
slave10# su mongod -c "numactl --interleave=all mongod -f /etc/mongoddb/shardsvr.conf --replSet rs0 --bind_ip 192.168.1.12 --fork"
slave11# su mongod -c "numactl --interleave=all mongod -f /etc/mongoddb/shardsvr.conf --replSet rs1 --bind_ip 192.168.1.13 --fork"
slave12# su mongod -c "numactl --interleave=all mongod -f /etc/mongoddb/shardsvr.conf --replSet rs2 --bind_ip 192.168.1.14 --fork"
slave13# su mongod -c "numactl --interleave=all mongod -f /etc/mongoddb/shardsvr.conf --replSet rs3 --bind_ip 192.168.1.15 --fork"
slave14# su mongod -c "numactl --interleave=all mongod -f /etc/mongoddb/shardsvr.conf --replSet rs4 --bind_ip 192.168.1.16 --fork"
slave15# su mongod -c "numactl --interleave=all mongod -f /etc/mongoddb/shardsvr.conf --replSet rs5 --bind_ip 192.168.1.17 --fork"
slave16# su mongod -c "numactl --interleave=all mongod -f /etc/mongoddb/shardsvr.conf --replSet rs6 --bind_ip 192.168.1.18 --fork"
slave17# su mongod -c "numactl --interleave=all mongod -f /etc/mongoddb/shardsvr.conf --replSet rs7 --bind_ip 192.168.1.19 --fork"
slave18# su mongod -c "numactl --interleave=all mongod -f /etc/mongoddb/shardsvr.conf --replSet rs8 --bind_ip 192.168.1.20 --fork"

```

B.2 Replica Sets

Initialize replica sets

```
ken@snsd$ mongo taeyeon:27018
MongoDB shell version: 2.0.5
connecting to: taeyeon:27018/test
> var cfg = {
...   _id : "rs0",
...   members : [
...     { _id : 0, host : "taeyeon:27018", priority : 1},
...     { _id : 1, host : "kimtaeyeon:27018", priority : 2},
...     { _id : 2, host : "snsd:27020", arbiterOnly : true}
...   ]
... }
> rs.initiate(cfg)
{
  "info" : "Config now saved locally. Should come online in about a minute.",
  "ok" : 1
}
> rs.conf()
{
  "_id" : "rs0",
  "version" : 1,
  "members" : [
    {
      "_id" : 0,
      "host" : "taeyeon:27018"
    },
    {
      "_id" : 1,
      "host" : "kimtaeyeon:27018",
      "priority" : 2
    },
    {
      "_id" : 2,
      "host" : "snsd:27020",
      "arbiterOnly" : true
    }
  ]
}
SECONDARY> rs.status()
{
  "set" : "rs0",
  "date" : ISODate("2012-06-13T17:06:07Z"),
  "myState" : 2,
  "syncingTo" : "kimtaeyeon:27018",
  "members" : [
    {
      "_id" : 0,
      "name" : "taeyeon:27018",
      "health" : 1,
      "state" : 2,
      "stateStr" : "SECONDARY",
      "optime" : {
        "t" : 1339606871000,
        "i" : 1
      },
      "optimeDate" : ISODate("2012-06-13T17:01:11Z"),
      "self" : true
    },
    {
      "_id" : 1,
      "name" : "kimtaeyeon:27018",
      "health" : 1,
      "state" : 1,
      "stateStr" : "PRIMARY",
      "uptime" : 286,
      "optime" : {
```

```

        "t" : 1339606871000,
        "i" : 1
    },
    "optimeDate" : ISODate("2012-06-13T17:01:11Z"),
    "lastHeartbeat" : ISODate("2012-06-13T17:06:07Z"),
    "pingMs" : 0
},
{
    "_id" : 2,
    "name" : "snsd:27020",
    "health" : 1,
    "state" : 7,
    "stateStr" : "ARBITER",
    "uptime" : 248,
    "optime" : {
        "t" : 0,
        "i" : 0
    },
    "optimeDate" : ISODate("1970-01-01T00:00:00Z"),
    "lastHeartbeat" : ISODate("2012-06-13T17:06:07Z"),
    "pingMs" : 0
}
],
"ok" : 1
}
SECONDARY>

```

Replica sets configuration

```

var cfg = {
  _id : "rs0",
  members : [
    { _id : 0, host : "taeyeon:27018", priority : 1},
    { _id : 1, host : "kimtaeyeon:27018", priority : 2},
    { _id : 2, host : "snsd:27020", arbiterOnly : true}
  ]
}

var cfg = {
  _id : "rs1",
  members : [
    { _id : 0, host : "jessica:27018"},
    { _id : 1, host : "jungsooyeon:27018"},
    { _id : 2, host : "snsd:27021", arbiterOnly : true}
  ]
}

var cfg = {
  _id : "rs2",
  members : [
    { _id : 0, host : "sunny:27018"},
    { _id : 1, host : "leesoonkyu:27018"},
    { _id : 2, host : "snsd:27022", arbiterOnly : true}
  ]
}

var cfg = {
  _id : "rs3",
  members : [
    { _id : 0, host : "tiffany:27018"},
    { _id : 1, host : "hwangmiyoung:27018"},
    { _id : 2, host : "snsd:27023", arbiterOnly : true}
  ]
}

var cfg = {
  _id : "rs4",
  members : [
    { _id : 0, host : "hyoyeon:27018"},
    { _id : 1, host : "kimhyoyeon:27018"},

```

```

    { _id : 2, host : "snsd:27024", arbiterOnly : true }
  ]
}

var cfg = {
  _id : "rs5",
  members : [
    { _id : 0, host : "yuri:27018"},
    { _id : 1, host : "kwonyuri:27018"},
    { _id : 2, host : "snsd:27025", arbiterOnly : true }
  ]
}

var cfg = {
  _id : "rs6",
  members : [
    { _id : 0, host : "sooyoung:27018"},
    { _id : 1, host : "choisooyoung:27018"},
    { _id : 2, host : "snsd:27026", arbiterOnly : true }
  ]
}

var cfg = {
  _id : "rs7",
  members : [
    { _id : 0, host : "yoona:27018"},
    { _id : 1, host : "imyooona:27018"},
    { _id : 2, host : "snsd:27027", arbiterOnly : true }
  ]
}

var cfg = {
  _id : "rs8",
  members : [
    { _id : 0, host : "seohyun:27018"},
    { _id : 1, host : "seojuhyun:27018"},
    { _id : 2, host : "snsd:27028", arbiterOnly : true }
  ]
}

```



B.3 Sharding

Add shards

```

ken@snsd$ mongo snsd:27017
MongoDB shell version: 2.0.5
connecting to: snsd:27017/test
mongos> use admin
switched to db admin
mongos> db.runCommand({ addshard : "rs0/taeyeon:27018, kimtaeyeon:27018" });
{ "shardAdded" : "rs0", "ok" : 1 }
mongos> db.runCommand({ addshard : "rs1/jessica:27018, jungsooyeon:27018" });
{ "shardAdded" : "rs1", "ok" : 1 }
mongos> db.runCommand({ addshard : "rs2/sunny:27018, leesoonkyu:27018" });
{ "shardAdded" : "rs2", "ok" : 1 }
mongos> db.runCommand({ addshard : "rs3/tiffany:27018, hwangmiyoung:27018" });
{ "shardAdded" : "rs3", "ok" : 1 }
mongos> db.runCommand({ addshard : "rs4/hyoyeon:27018, kimhyoyeon:27018" });
{ "shardAdded" : "rs4", "ok" : 1 }
mongos> db.runCommand({ addshard : "rs5/yuri:27018, kwonyuri:27018" });
{ "shardAdded" : "rs5", "ok" : 1 }
mongos> db.runCommand({ addshard : "rs6/sooyoung:27018, choisooyoung:27018" });
{ "shardAdded" : "rs6", "ok" : 1 }
mongos> db.runCommand({ addshard : "rs7/yoona:27018, imyooona:27018" });
{ "shardAdded" : "rs7", "ok" : 1 }
mongos> db.runCommand({ addshard : "rs8/seohyun:27018, seojuhyun:27018" });
{ "shardAdded" : "rs8", "ok" : 1 }

```



```

mongos> db.runCommand({ listShards : 1 });
{
  "shards" : [
    {
      "_id" : "rs0",
      "host" : "rs0/kimtaeyeon:27018,taeyeon:27018"
    },
    {
      "_id" : "rs1",
      "host" : "rs1/jessica:27018,jungsooyeon:27018"
    },
    {
      "_id" : "rs2",
      "host" : "rs2/leesoonkyu:27018,sunny:27018"
    },
    {
      "_id" : "rs3",
      "host" : "rs3/hwangmiyoung:27018,tiffany:27018"
    },
    {
      "_id" : "rs4",
      "host" : "rs4/hyoyeon:27018,kimhyoyeon:27018"
    },
    {
      "_id" : "rs5",
      "host" : "rs5/kwonyuri:27018,yuri:27018"
    },
    {
      "_id" : "rs6",
      "host" : "rs6/choisooyoung:27018,sooyoung:27018"
    },
    {
      "_id" : "rs7",
      "host" : "rs7/imyoona:27018,yoona:27018"
    },
    {
      "_id" : "rs8",
      "host" : "rs8/seohyun:27018,seojuhyun:27018"
    }
  ],
  "ok" : 1
}
mongos>

```

Enable sharding

```

ken@snsd$ mongo sns:27017
MongoDB shell version: 2.0.5
connecting to: sns:27017/test
mongos> printShardingStatus()
--- Sharding Status ---
  sharding version: { "_id" : 1, "version" : 3 }
  shards:
    { "_id" : "rs0", "host" : "rs0/kimtaeyeon:27018,taeyeon:27018" }
    { "_id" : "rs1", "host" : "rs1/jessica:27018,jungsooyeon:27018" }
    { "_id" : "rs2", "host" : "rs2/leesoonkyu:27018,sunny:27018" }
    { "_id" : "rs3", "host" : "rs3/hwangmiyoung:27018,tiffany:27018" }
    { "_id" : "rs4", "host" : "rs4/hyoyeon:27018,kimhyoyeon:27018" }
    { "_id" : "rs5", "host" : "rs5/kwonyuri:27018,yuri:27018" }
    { "_id" : "rs6", "host" : "rs6/choisooyoung:27018,sooyoung:27018" }
    { "_id" : "rs7", "host" : "rs7/imyoona:27018,yoona:27018" }
    { "_id" : "rs8", "host" : "rs8/seohyun:27018,seojuhyun:27018" }
  databases:
    { "_id" : "admin", "partitioned" : false, "primary" : "config" }

mongos> db.runCommand({enableSharding:"plurk"})
{ "ok" : 1 }

mongos> db.runCommand({shardcollection:"plurk.plurks", key:{_id:1}})

```

```

{ "collectionsharded" : "plurk.plurks", "ok" : 1 }
mongos> printShardingStatus()
--- Sharding Status ---
  sharding version: { "_id" : 1, "version" : 3 }
  shards:
    { "_id" : "rs0", "host" : "rs0/kimtaeyeon:27018,taeyeon:27018" }
    { "_id" : "rs1", "host" : "rs1/jessica:27018,jungsooyeon:27018" }
    { "_id" : "rs2", "host" : "rs2/leesoonyoung:27018,sunny:27018" }
    { "_id" : "rs3", "host" : "rs3/hwangmiyoung:27018,tiffany:27018" }
    { "_id" : "rs4", "host" : "rs4/hyoyeon:27018,kimhyoyeon:27018" }
    { "_id" : "rs5", "host" : "rs5/kwonyuri:27018,yuri:27018" }
    { "_id" : "rs6", "host" : "rs6/choisooyoung:27018,sooyoung:27018" }
    { "_id" : "rs7", "host" : "rs7/imyoona:27018,yoona:27018" }
    { "_id" : "rs8", "host" : "rs8/seohyun:27018,seojuhyun:27018" }
  databases:
    { "_id" : "admin", "partitioned" : false, "primary" : "config" }
    { "_id" : "plurk", "partitioned" : true, "primary" : "rs6" }
    plurk.plurks chunks:
      rs6      1
      { "_id" : { $minKey : 1 } } --> { "_id" : { $maxKey : 1 } } on : rs6 Timestamp(1000, 0)

mongos> printShardingStatus()
--- Sharding Status ---
  sharding version: { "_id" : 1, "version" : 3 }
  shards:
    { "_id" : "rs0", "host" : "rs0/kimtaeyeon:27018,taeyeon:27018" }
    { "_id" : "rs1", "host" : "rs1/jessica:27018,jungsooyeon:27018" }
    { "_id" : "rs2", "host" : "rs2/leesoonyoung:27018,sunny:27018" }
    { "_id" : "rs3", "host" : "rs3/hwangmiyoung:27018,tiffany:27018" }
    { "_id" : "rs4", "host" : "rs4/hyoyeon:27018,kimhyoyeon:27018" }
    { "_id" : "rs5", "host" : "rs5/kwonyuri:27018,yuri:27018" }
    { "_id" : "rs6", "host" : "rs6/choisooyoung:27018,sooyoung:27018" }
    { "_id" : "rs7", "host" : "rs7/imyoona:27018,yoona:27018" }
    { "_id" : "rs8", "host" : "rs8/seohyun:27018,seojuhyun:27018" }
  databases:
    { "_id" : "admin", "partitioned" : false, "primary" : "config" }
    { "_id" : "plurk", "partitioned" : true, "primary" : "rs6" }
    plurk.plurks chunks:
      rs0      1
      rs4      1
      rs1      1
      rs2      1
      rs5      1
      rs6      1
      rs3      1
      { "_id" : { $minKey : 1 } } --> { "_id" : 38 } on : rs0 Timestamp(3000, 0)
      { "_id" : 38 } --> { "_id" : 834733 } on : rs4 Timestamp(6000, 0)
      { "_id" : 834733 } --> { "_id" : 11488297 } on : rs1 Timestamp(6000, 1)
      { "_id" : 11488297 } --> { "_id" : 33395723 } on : rs2 Timestamp(5000, 0)
      { "_id" : 33395723 } --> { "_id" : 50989408 } on : rs5 Timestamp(7000, 0)
      { "_id" : 50989408 } --> { "_id" : 252729929 } on : rs6 Timestamp(7000, 1)
      { "_id" : 252729929 } --> { "_id" : { $maxKey : 1 } } on : rs3 Timestamp(2000, 0)

mongos> db.printShardingStatus( true )
--- Sharding Status ---
  sharding version: { "_id" : 1, "version" : 3 }
  shards:
    { "_id" : "rs0", "host" : "rs0/kimtaeyeon:27018,taeyeon:27018" }
    { "_id" : "rs1", "host" : "rs1/jessica:27018,jungsooyeon:27018" }
    { "_id" : "rs2", "host" : "rs2/leesoonyoung:27018,sunny:27018" }
    { "_id" : "rs3", "host" : "rs3/hwangmiyoung:27018,tiffany:27018" }
    { "_id" : "rs4", "host" : "rs4/hyoyeon:27018,kimhyoyeon:27018" }
    { "_id" : "rs5", "host" : "rs5/kwonyuri:27018,yuri:27018" }
    { "_id" : "rs6", "host" : "rs6/choisooyoung:27018,sooyoung:27018" }
    { "_id" : "rs7", "host" : "rs7/imyoona:27018,yoona:27018" }
    { "_id" : "rs8", "host" : "rs8/seohyun:27018,seojuhyun:27018" }
  databases:
    { "_id" : "admin", "partitioned" : false, "primary" : "config" }
    { "_id" : "plurk", "partitioned" : true, "primary" : "rs6" }
    plurk.plurks chunks:
      rs2      122
      rs4      122
      rs1      122
      rs6      122

```

```

rs0      122
rs3      122
rs8      122
rs5      122
rs7      123
{ "_id" : { $minKey : 1 } } --> { "_id" : 38 } on : rs2 Timestamp(561000, 0)
{ "_id" : 38 } --> { "_id" : 197903 } on : rs4 Timestamp(652000, 1)
{ "_id" : 197903 } --> { "_id" : 612416 } on : rs1 Timestamp(474000, 0)
{ "_id" : 612416 } --> { "_id" : 921696 } on : rs6 Timestamp(477000, 0)
{ "_id" : 921696 } --> { "_id" : 1325680 } on : rs0 Timestamp(561000, 1)
...
{ "_id" : 599934313 } --> { "_id" : 600312646 } on : rs7 Timestamp(728000, 6)
{ "_id" : 600312646 } --> { "_id" : 600700305 } on : rs7 Timestamp(730000, 2)
{ "_id" : 600700305 } --> { "_id" : 877069913 } on : rs7 Timestamp(730000, 3)
{ "_id" : 877069913 } --> { "_id" : 878621144 } on : rs8 Timestamp(99000, 4)
{ "_id" : 878621144 } --> { "_id" : 891168102 } on : rs8 Timestamp(121000, 6)
{ "_id" : 891168102 } --> { "_id" : 892947685 } on : rs5 Timestamp(129000, 4)
{ "_id" : 892947685 } --> { "_id" : 895436846 } on : rs2 Timestamp(142000, 4)
{ "_id" : 895436846 } --> { "_id" : 898183991 } on : rs2 Timestamp(152000, 6)
...
{ "_id" : 968419263 } --> { "_id" : 968803422 } on : rs5 Timestamp(563000, 2)
{ "_id" : 968803422 } --> { "_id" : 969199788 } on : rs5 Timestamp(569000, 8)
...
{ "_id" : 1008503104 } --> { "_id" : 1010166169 } on : rs8 Timestamp(501000, 6)
{ "_id" : 1010166169 } --> { "_id" : 1011814205 } on : rs8 Timestamp(509000, 2)
...
{ "_id" : 1039107320 } --> { "_id" : 1040049038 } on : rs3 Timestamp(686000, 4)
{ "_id" : 1040049038 } --> { "_id" : { $maxKey : 1 } } on : rs3 Timestamp(686000, 5)

```

mongos>

