ISSN:0975-9646

Distributed Content Aggregation& Content Change Detection using Bloom Filters

Sornalingam Nadaraj

Abstract - In this paper, we put forward a technique for distributed aggregation of the web and content classification. Over the decade, Internet has grown exponentially in size and usage. Aggregation is way to collect the scattered information across the internet and provide it under a single hood. This paper puts forward an architecture built on Message queue architecture for running crawlers in distributed environment and classifying content. Enormous growth of websites provides lot of duplicate information and often times it is dynamic in nature. This paper describes an approach for distributed aggregation and faster scalable content classification and change detection. We have used a three-step algorithm for refreshing page content. It checks whether the content of a web page has been changed or not. Also it provides a way to find which portion of a field has been altered using Bloom filter.

Index Terms - Distributed Aggregation, Change detection, Multi-threaded server, Structural and content changes, Bloom Filter.

1 INTRODUCTION

1.1 Definition

During aggregation each site needs a separate crawler to extract the necessary information. Since web pages are different in nature we need to have separate crawler for each site. Each crawler will have its own start seed URL and template (JSON, XML). It starts with a seed URL and then follows the links on each page in a Breadth First or a Depth First method [1]. With ever increasing size of web, multiple crawlers are required to run in parallel to browse and download web pages. Companies in domains such as, Health care, Stock market, Advertising, Talent Acquisition, Search Engines etc., use their own in house crawling technology to aggregate information from thousands and thousands of websites.

The present work is divided into 4 sections as follows. In current section we have discussed about the introduction of crawlers and related work done. Section 2 discusses the proposed architecture for the distributed aggregation. In section 3 we discuss algorithms used for content classification and detecting changes in web pages and finally in section 4 we conclude our work along with future directions.

1.2 Overview

Here, we propose new design architecture for building distributed aggregation architecture. The main challenge while making such a design is to maximize the performance of the system, scalability and content classification. To achieve efficient process we need to tackle some common issues faced by the aggregation processes. The most frequent challenges in distributed aggregation system are –

1. Efficiency of the crawlers. They should not go into infinite loops and aggregate duplicate content.
2. Due to the large content of the web pages there is a possibility that we may crawl same content in different web sites.
3. Dynamic nature of websites leads to frequent changes in the content.
4. Aggregating new content, Detecting content changes (updated content), Excluding duplicate content and removing deleted content have become a big challenge. We need to have an efficient system to aggregate the reliable content from abundant pages available online.

- ***Crawler Architecture:*** Previous publications describe various architectures under which crawlers of certain current search engines work. [3] Describes the architecture of the crawling technique used by Google whereas [4] studies the Compaq SRC crawler. Although these papers describe the macro view of the crawler architecture used by them, but little insight or detail has been provided by them regarding the issues related to parallel crawlers that have been discussed above. [2] Describes a parallel crawler with multiple architectures along with metrics for evaluation.
- ***Page Update policies:*** Each crawler needs to update the pages on a periodic basis to improve the quality of the content in its databases. [6] Discusses scheduling algorithms for crawlers to index the web on a regular basis. [5] Describes the various freshness metrics used for gauging the freshness and quality of a local copy of a web page.
- ***Queue Processing:*** Queue processing allow us to achieve distributed processing, Horizontal and vertical scalability. [8] Describes the queue processing component using which we can build scalable and fault tolerance system for crawling and content classification.
- ***Content classification:*** Aggregated contents need to be classified into different categories. [7] Describes the fastest and memory efficient way to filter the content using hash functions.

Aggregation of web content consumes significant amount of resources such as network bandwidth and other resources due to the fast access of pages. To maintain the most updated content of a page we have to crawl the pages repeatedly. Thus the crawling activities of a single search engine can cause a daily load of about 60GB to the web [9].

This load will increase significantly in future as the web will grow exponentially in the future.
Distributed and parallel crawling [10] was purposed to increase the coverage and decrease the bandwidth usage, but this does not help a lot. The distribution of the crawling function was efficiently reducing the network bottleneck from the search engine's site and improves the quality of the results, but these are not at all optimized.
Page change detection Algorithms: [11] Describes and compares the different algorithms to detect the web page content changes. It describes the content classification using Tree and Hash based approach and their pros and cons. One of the biggest problems is to identify which portion of the page has been updated. In the aggregation system it plays a major role to keep the content updated.

2 PROPOSED ARCHITECTURE

The proposed architecture consists of 2 different parts,

a. Consumer to run the spiders/crawlers in parallel
b. Consumer to run the content change detection

- ***Consumer to run the spiders in parallel:*** There is a queue which will hold the spider template and start URL.

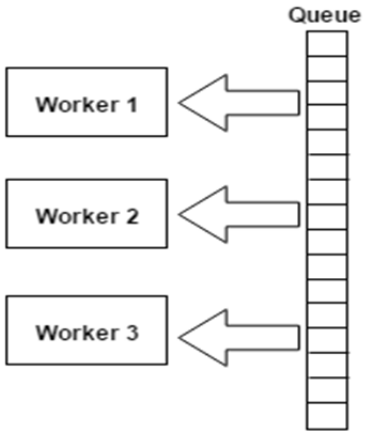

Figure 1: Publisher and Subscriber architecture to run the spider in distributed workers. Queue: It consists of Spider template and Start URL to crawl

The queue will be loaded with the messages (spider template and start URL) by the scheduler on a periodic basis. Whenever a message is available in the queue, the consumer will consume the message and deserialize the message. The consumer will run the crawler on the worker machine using the template and seed the start URL. The spider will crawl the content and directly load it into another queue. This will ensure that the spider is not tightly coupled with a machine. It will provide a way to decouple spiders and run on different machines in a distributed network. Each site will have a different pattern and we cannot use a single spider template for all the sites. Adding the spider template within the message will help the consumer to use the template and run the spider.
It will ensure that the spiders are running on different machines parallelly and aggregate the content. The task of the spider is to aggregate the content and it does not care about whether the data has reached its destination or not. This way the functionality of a spider is restricted with the aggregation process itself.

While aggregating the site we use bloom filter to find the duplicate URL's in the site, So that the spider will not go into infinite loop. The spider will begin to aggregate the page using the parent URL (start URL of a page). The spider will look for the hyper link and make the new request to the URL that are available on the page, Before making new request, it will generate hash value for all the URL's it is crawling and the generated hash key will be loaded into Bloom filter.
It follows the BFS method to aggregate the contents from various child pages.

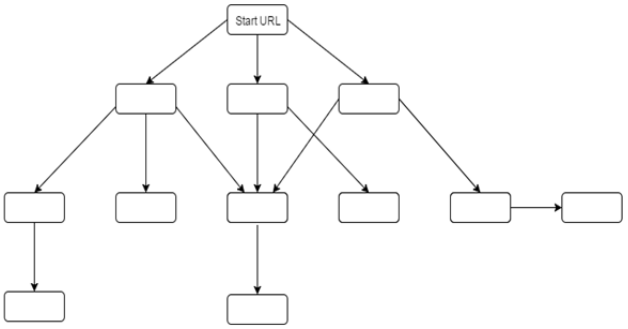

Figure 2: Example web page crawling without bloom filter.

In the proposed logic for each request/URL it will check whether the URL is already crawled or not by checking the existence of URL in the bloom filter [Figure3]. If the URL is already available in the bloom filter lookup then it will skip the URL and move to next one on the page.

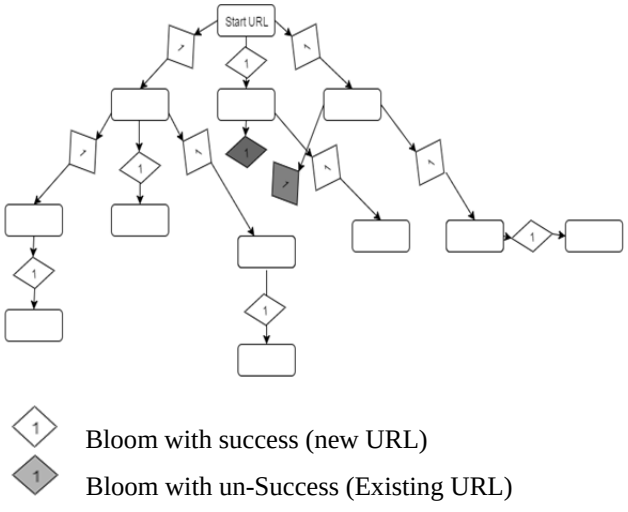

Figure 3: Example web page crawling with bloom filter.

This way the spider makes sure that the duplicate URL/Content will not be aggregated for the current execution of the spider.

Aggregated contents will be loaded into a single queue in a JSON format which will include the crawled source URL.

Each JSON represents extracted information of a single page [**Figure 4**],

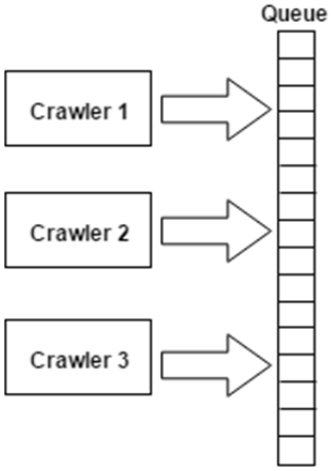

Figure 4: Crawler publishes the aggregated content to queue in JSON format.

b. Consumer to run the content change detection :

From the above **[Figure 4]** queue consumer will receive the aggregated JSON.

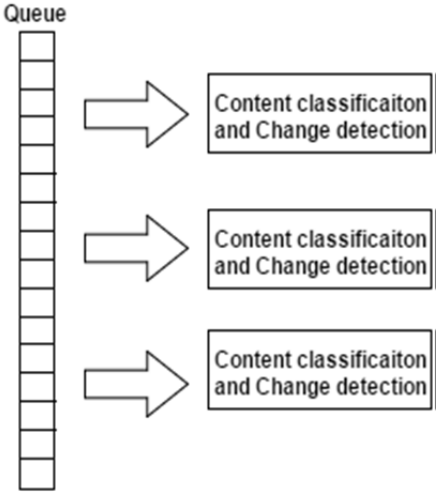

Figure 5: Consumer to find change detection and content classification.

From the above **[Figure 5]** queue, the consumer will receive the content and apply the following logic to classify the contents,

1. Consume the content from queue
2. Create Hash key for URL from the JSON content
3. Check whether the hash is already available in bloom filter file.
    a. If it is already available then, Create hash key for the crawled content.
    b. Check whether the hash is already available in the bloom lookup
        i. If yes, then it is an existing content.
        ii. If no, then content has been updated.
4. If not, Then add the URL hash value into the Bloom lookup file1
5. Also Create hash key for the crawled content and insert into Bloom filter lookup.

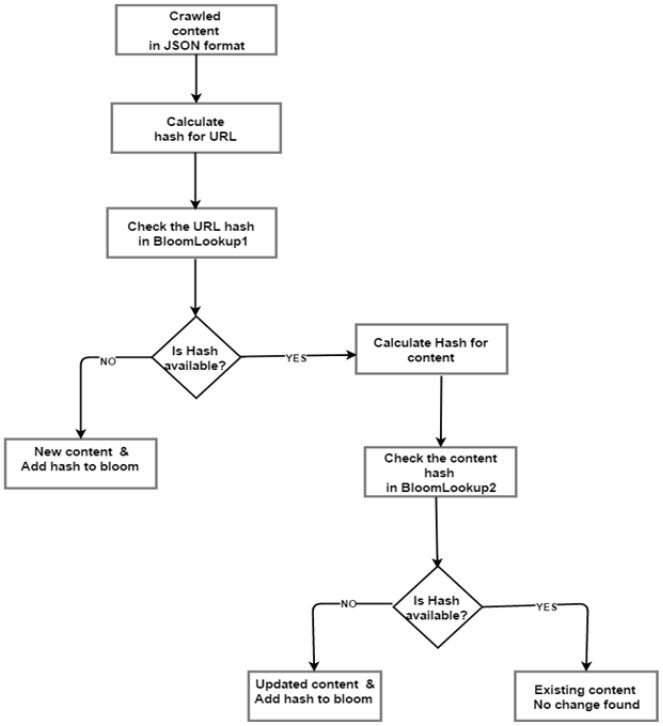

Figure 6: Content classification workflow

3 EXPERIMENTS

We used Python to build the spiders. Python tasks were used for distributed message processing. Site URLs were collected randomly.

Hardware details:

- Operating System: CentOS 7
- CPU: 2
- RAM: 2 GB

Example Aggregated JSON content:

{

'Asked' : ["today"],
'Description': ["favorite
Im using the following code to get the abbreviated(3 letters) time zone in java7. But after updating to java8, all I get is offset of that timezone rather than the abbreviation.Can anyone help?
static final ZONE_SHORT_FORMATTER = DateTimeFormat.forPattern("zzz");
String timeZoneString = US/Arizona;
String loc= DateTimeZone.forID(timeZoneString);
// loc = America/Phoenix in this case
long time = DateTimeUtils.currentTimeMillis()
Sting timeZoneShort = ZONE_SHORT_FORMATTER.withZone(loc).print(time);
Here,in timeZoneShort i get -07:00, which is the offset, rather than getting MST."],
'View' : ["219times"],
'Title' : [u"TimeZone abbreviation not working after updating to java 8
"],
'_template' : "42073c2a2cd35c84bbabdaff56e72c03c90ef6a4",
'_type' : "default",
'url' : "http://stackoverflow.com/questions/35145570/timezone-abbreviation-not-working-after-updating-to-java-8"

}

Experiment 1: Content classification

S.No	Total no of pages to aggregate	Total no of websites	Time taken to aggregate the content (*Min)*	Bloom filter lookup size	Time taken to classify the content using bloom filter *(Sec)*
1	1,000	2	15	100,000	~10
2	5,000	10	30	100,000	~42
3	10,000	13	45	100,000	~85
4	20,000	17	75	100,000	~175
5	50,000	30	180	100,000	~427

Experiment 2: Comparison between Bloom vs Lookup String based content classification.

S. No	Lookup Size	Input content count	Time taken to classify input using traditional string lookup comparison *(Sec)*	Time taken to classify input content using Bloom lookup *(Sec)*
1	1,000	1	0.001254	0.0003
2	5,000	1	0.003	0.0005
3	10,000	1	0.006	0.0008
4	20,000	1	0.2	0.001
5	50,000	1	0.35	0.002
6	100,000	1	0.7	0.008
7	1,000,000	1	1.29	0.03

4 CONCLUSION

The architecture that has been proposed in this paper has the following distinct advantages:

- It follows message based consumer and producer for processing aggregated content.
- New real time content aggregation can be achieved by this methodology.
- The algorithm for content classification and change detection is designed in such a way that it will even find the small changes in the content and faster than the traditional string lookup comparison.
- Ensures the Horizontal and Vertical scalability.
- Can dynamically attach and detach hardware resources in a distributed environment to improve the processing capacity

ACKNOWLEDGMENT

We acknowledge Mr. Vikas Aher for assistance with Distributed crawling. Mr. Udayakumar Rajendran and Mr. Jayanta Chowdhuri for advice and comments on the manuscript and for support.

REFERENCES:

[1] David Eichmann, "***The RBSE Spider – Balancing effective search against web load***", Repository Based Software Engineering Program , Research Institute for Computing and Information Systems, University of Houston – Clear Lake.

[2] Junghoo Cho & Hector Garcia-Molina, "***Parallel Crawlers***". Proceedings of the 11th international conference on World Wide Web WWW '02, Honolulu, Hawaii, USA. ACM Press. Page(s): 124 – 135.

[3] Sergey Brin and Lawrence Page, "***The Anatomy of a Large-Scale Hypertextual Web Search Engine***",In Proceedings of the Seventh World-Wide Web Conference, 1998.

[4] Heydon and M. Najork, "***Mercator: A scalable, extensible web crawler. Word Wide Web***", December 1999. Page(s):219–229

[5] Junghoo Cho and Hector Garcia-Molina, "***Synchronizing a database to improve freshness, submitted for publication***". Proceedings of the 2000 ACM SIGMOD international conference on Management of data. Volume 29 Issue 2. Page(s): 117 – 128.

[6] E. Co.man, Jr., Z. Liu, and R. R. Weber, "***Optimal robot scheduling for web search engines***". Proceedings of the 11th international conference on World Wide Web WWW '02 Honolulu, Hawaii, USA. ACM Press. Page(s): 136 – 147.

[7] Ahmadi, M. ; Comput. Eng. Lab., Delft Univ. of Technol., Delft, Netherlands ; Wong, S. "***K-Stage Pipelined Bloom Filter for Packet Classification***"

[8] Sumit Dawar1, 2, Sven van der Meer1, Enda Fallon2, John Keeney1, Tom Bennett "***Building a Scalable Event Processing System with Messaging and Policies –Test and Evaluation of RabbitMQ and Drools Expert***"

[9] J. Cho and Garcia-Molina, H. ***"The evolution of the web and implications for an incremental crawler"***. In proceedings of the Twenty-sixth International Conference on Very Large Databases, Cairo, Egypt, September 2000.

[10] Vikas, O. Chiluka, N.J. Ray, P.K. Girraj Meena Meshram, A.K. Gupta, A. Sisodia, A., ***"WebMiner--Anatomy of Super Peer Based Incremental Topic-Specific Web Crawler"***. Networking, ICN '07. Sixth International Conference on, pp.32-32, April 2007.

[11] Shobhna , Manoj Chaudhary "***A Survey on Web Page Change Detection System Using Different Approaches.***"