A Cloud-based Web Crawler Architecture

Poonam Maheshwar
Management Education Research Institute, College of Engineering & Technology, (Sampla), INDIA

ABSTRACT

The cloud computing paradigm provides support for elastic resources and unstructured data, and provides pay-per-use features that allow individual businesses to run their own web crawlers for crawling the Internet or a limited web hosts. In this paper, we propose a cloud-based web crawler architecture that uses cloud computing features and the Map Reduce programming technique. The proposed web crawler allows us to crawl the web by using distributed agents and each agent stores its own finding on a PostgreSQL using amazon web services. The proposed web crawler also could store unstructured and massive amount of data on PostgreSQL.

Keywords: SQL, Web, URL

I. INTRODUCTION

The Internet has grown exponentially since the late 1960s. The Internet is used by more than 2.4 billion people.

Collecting and mining such a massive amount of content has become extremely important but very difficult because in such conditions, traditional web crawlers are not cost effective as they would be extremely expensive and time-consuming. Consequently, distributed web crawlers have been an active area of research. As the primary component of search engines, distributed web crawlers make it possible to easily gather information on the Internet. Single-process crawlers start with a list of URLs, visit these URLs, and download the corresponding webpages. Then the web crawler identifies hyperlinks in the pages and adds them to the URLs’ list in order to crawl the links in the future. A distributed web crawlers have multiple agents for crawling URLs; however, most of the distributed web crawlers need a large number of servers to crawl web content in parallel. Unfortunately, in most cases are not possible for individuals and small businesses to have multiple servers. Note that big companies, such as Microsoft, have more than a million servers which are distributed geographically.

In this paper, I introduce a scalable web crawler that employed cloud computing technology. A cloud-based web crawler allows people to collect and mine web content without buying, installing and maintaining any infrastructure. Cloud computing technology provides an effective and efficient means of using resources, it allowing users to build their virtual IT department on the cloud. It allows users to start by small resources, then additional computing and storage capacity on demand. Cloud computing provide a distributed system and support the required High Performance Computing (HPC) for distributed geographically web crawlers. In addition, cloud computing provides scalability feature to users. It allows users to add or remove crawler agents at any time on demand; therefore, users do not have concern about the size of the Internet or how much data they need to analyse, because the users can add new resources (processor and storage capacity) on demand.

II. BACKGROUND AND RELATED WORK

The cloud computing has five essential characteristics:
A. On demand self-service Services such as computing machines and/or storage devices can be scaled up or down on the users’ demand, without any interaction with the provider. This feature allows the proposed web crawler to add agents rapidly on demand.
B. Broad network access Resources are accessible over the network via any device and any platform (e.g., mobile phones, tablets, laptops, and workstations). This feature provides platform independence to our proposed web crawler.
C. Resource pooling A cloud provider pools computing resources together and allocates these resources to users based on their requirements. Cloud providers usually have multiple distributed servers in several places; so, this feature allows a web crawler to have resource pools which are distributed geographically.
D. Scalability Resources can be scaled up or down on demand. Users can get as much resources as they need at any time. This feature provides scalability to a web crawler and allows users to add/remove agents when they are not needed.
E. Measured service Consumers are charged based on the resource usage. A cloud provider monitors and measures the resource usage for all users. This is known as pay-per-use model. This feature allows users to decrease their cost by removing agents that are not needed.

Traditional web crawlers can only download a limited number of web pages in a given time. In recent years, with the exponential growth of the Internet, researchers have focused on building distributed web crawlers. A distributed web crawler has multiple servers and crawling web pages can be done in parallel to improve performance. Xu et al. introduced a User-oriented web crawler that adaptively crawls social media contents on the Internet to satisfy user’s particular online data source acquisition requirements. They developed a feedback component that can estimate the utility score of web pages. This score can be used to optimize the web crawler’s crawling decisions. They use human-labelled example web pages to train their system. Then, they determined a priority list of search results by using a ranking function to measure the utility of the individual search result pages with a prediction function. They evaluated their crawler in the context of cancer epidemiological research. Furthermore, they use identifier-seeded consistent hashing to manage their agents (which indicate each worker of distributed web crawler) and hosts (which indicate each crawling URLs). Their results show a linear relationship between the number of agents and the number of pages they can fetch. A scalable, extensible Web Crawler was developed by Haydon et al. where they introduce a scalable and extensible web crawler that can be scaled up to the entire web. Their web crawler uses hundreds of worker-threads to crawl, download and process documents and they built several protocols and processing modules for different URL’s schemas and documents. Mika et al. developed a web semantic system as part of a web crawler by cloud concept. However, in this study, the researchers did not provide an architecture for a web crawler. To the best of our knowledge, we could not find a specific cloud-based web crawler in academic literatures.

III. WEB CRAWLER REQUIREMENTS

The first requirement for a distributed web crawler is the selection of an appropriate web page partitioning scheme. The first scheme is the URL-hash-based which presents partition web pages based on URL’s hash value. Each hash is assigned to an Agent. The second scheme is site-hash-based function and assigns pages on the same website to the same agent. The third scheme is the hierarchical scheme and assigns pages based on some feature like language and region.

IV. A CLOUD-BASED WEB CRAWLER

The architecture of proposed web crawler is illustrated in Figure 1. Agent Registrar database maintains a list of agents and their host (a zone of the Internet). An agent Ai crawls a URL and adds its retrieved results (a list of found URLs).

The proposed web crawler, Cloud-based web crawler engine (CWCE), uses Amazon web services and PostgreSQL for maintaining a list of retrieved URLs from a page. The queued URLs are temporary and awaiting a fetch by the corresponding agent of its zone. I use PostgreSQL for storing permanent information about crawled pages. Across the Internet, fields differ from record to record. The Table is based on PostgreSQL database, allowing us to define a new field on-the-fly when I insert a record with a new field. For example, a record of table could have an extra field, such as an image type, and we want to add an image’s URL to the Table. In this case, Table will be dynamic and without any schema.

Partition Key: This field is the first key of partitioning of the web and represents the host of a URL, such as “cloudlab.ucmerced.edu” in “http://cloudlab.ucmerced.edu/lab” URL. This field allows us to distribute different URLs based on their hosts (URL hash based).

Row Key: As I described above, the CWCE partitions the web based on host (partition key) and then based on hash of URL(row key). This field represents the hash of a URL to provide a specific row at a partition key. In other words, URLs are distributed through different partition keys, and then are distributed through row key (the hash of URLs).

I added two fields (partition key and row key) based on MapReduce programming technique that allows to have a fully distributed system. For example, the first two rows in Table 1 will save in one partition and the third row could be saved in another partition because they have different hosts. The PostgreSQL DBMS can use different partitions to distribute data through different servers based on the partition key and, in the case of a massive size of a partition, the DBMS can distribute data based on row key. In Table 1, an agent (A1) works on a partition that crawls all links of “ssha.ucmerced.edu” and agent A2 works on another partition that crawls all links of “engineering.ucmerced.edu” host. Agent A1 crawls on two unvisited URLs, listed as follows: “http://ssha.ucmerced.edu/about” and “http://ssha.ucmerced.edu/about/contact-us”. Partition Key Row Key ssha.ucmerced.edu hash(“http://ssha.ucmerced.edu/about”) ssha.ucmerced.edu hash(“http://ssha.ucmerced.edu/about/contact-us”) engineering.ucmerced.edu hash(“http://engineering.ucmerced.edu/about”) engineering.ucmerced.edu hash(“http://engineering.ucmerced.edu/about/contact-us”) URL: This field represents a URL which is retrieved by an agent. Visit: This field represents the current status of crawling of a URL. If a URL is added to the Table and it is never crawled by an agent, the value of this field will be “False” and Partition Key Row Key URL Visit Hit String String Boolean Int32
When an agent crawls the URL, the value of this field will update to “True”. Hit: Hit represents the number of citations to a URL. Different agents retrieve the current number of hit of a URL. If an agent finds a record with the same URL, then the value of hit will be increased by 1. As illustrated in Figure 1. In this architecture, I used the Amazon web service to maintain a temporary list of URLs that requires to crawl and PostgreSQL to maintain permanent information of crawled URLs. DNS Resolver maintains a list of URLs that the web crawler is required to crawl. Each agent works on one partition. Each agent fetches a URL (Ui) from the queue (if the partition of Ui is the same as partition of the current agent), and the agent retrieves a list of URLs {Uj..Uk} from Ui. The agent checks availability of the selected URL in the queue for each Uj to Uk. If the selected URL is in the queue, the agent increases the value of Hit by 1. Otherwise, the agent adds the selected URL to the queue and adds the URL row key and the URL address in the table. If the record is found, then the agent increases the value of Hit by 1. Agent Registrar, based on a PostgreSQL database, maintains agent information such as agent name and its partition key. Page Index DB is based on PostgreSQL database because this database can collect a massive amount of URLs with a variety of fields for each record. For example, a record may have only 5 fields (as described in Table 1), or the record could have more than five fields, such as image type for image URLs. This features allows us to archive different information of crawled URLs and is useful for data mining. PostgreSQL provides a storage facility for unstructured data, such as indexing unstructured data, (e.g., binary files, video and images) from the web. For instance, collected PDF files from the Internet by a web crawler can be stored in this database. Analyser Tools and AWS provide monitoring tools for collecting information of crawler agents that are using resources, such as CPU usage and network usage. These components provide a default value for a web crawler configuration by suggesting a list of available agents and the maximum number of agents. The agent ith is denoted as Ai. Agent Ai can retrieve a list of URLs.
and A6, respectively. Agent A3 retrieves an intra-link and it is assigned to A7. The agent A7 also retrieves a link which is assigned to A3 (a recursive link between two pages on different partitions). A1.1 which is based on A1 (because they have the same host name) retrieves A8 as an inter-link and two intra-links, A1.1.1 and A1.1.2. Finally, A1.1.2 retrieves A1 as an intra-link. There are different important metrics for URLs that provide a suggestion for selecting a branch for extending further links of a page, rather than extending all retrieved URLs from one page of a URL, such as incoming URL, outgoing.

URL, enumeration link dependency, PageRank and content analysis. For instance, A1 decides which retrieved links have priority and should be selected for extending a branch, rather than extending all three retrieved links. In order to do so, link-dependent metrics for ranking each page is one of the metrics that can be used. This metric uses two parameters for each page: firstly, to retrieve an authority score of pagei which shows the number of citations (a number of URLs that point out to pagei), and second, h to compute the hub score that represents the number of citations (a number of out links to other pages). As described in equations (1) and (2), both parameters are related to each other. represents each link in crawling graph (dependency graph between links as shown in Figure 2) between Pagei and Pagej. = \sum_{E} h_{E} (1) h = \sum_{E} e_{E} (2) Distributed agents can be implemented at different levels of a computing system, such as Virtual Machines (VM), or multiple thread in one VMs. All VMs are managed by the Azure AppFabric controller. Azure does not have a limitation on the number of threads or connections that each server can support logically. However, each server in Azure could have different computational power and a server may not be able to physically handle a large number of running agents that might be created on-the-fly. In other words, theoretical maximum numbers of agents are dependent upon current cloud server configuration. In order to avoid this, the web crawler administrator sets configurations, such as maximum number of agents based on current cloud resources. If an administrator has a subscription with auto scaling up feature, the administrator would set the maximum number of agents to unlimited during the crawling process. However, unlimited agents status is not possible for an unfocused web crawler that is crawling the Internet without a limitation. Other methods, such as filtering can be implemented to avoid retrieve all links. Furthermore, Windows Azure supports scaling up and a web crawler administrator could use AWS to monitor performance counters for a number of running instances of agents (threads) and automatically add or remove servers to his Virtual Machines (VM).

V. EXPERIMENTAL RESULTS

I implemented the main component of the proposed cloud-based web crawler in Amazon web services to study its performance. The web crawler application connects to the App. In the case of lack of resources, an administrator could scale up or scale down computing resources and storage resources via the PostgreSQL. The component can run as several parallel agents in AWS on localhost server with using services. In this experimental result, I focus on 4 Available at http://cloudlab.ucmerced.edu/cloud-based-web-crawler implementation the architecture rather than increasing the hit number. I ran the web crawler by starting from one agent and then, increasing the number of parallel web-crawler agents to 2, 4, 8, 16, 24 and 32 on three small instances (VM) with a different rate of transactions on both table and queue. The different numbers of parallel agents were able to increase workload on shared resources (queue and table) that allow us to monitor scalability of the web crawler by recording the rate of transactions arrival during each hour.

The result shows that when the arrival rate of transactions increases, the response time (Average E2E Latency) remains fairly flat. The average response time for cloud table is 9.169 milliseconds for total 3,081,852 transactions and the average response time for cloud queue is 25.54 milliseconds for total 8,919,411 transactions. The result is reasonable because an agent could retrieve a URL from the same partition in which it is located. A web crawler retrieves all URLs from a host, but it could not crawl all of them because some host servers do not allow an application to crawl all the pages and the host server would return a HTTP code. A host server could return different HTTP codes to a web.

VI. ADVANTAGES OF CLOUD-BASED WEB CRAWLER

The proposed web crawler provides several advantages over traditional distributed web crawlers.

A. On demand scale up and scale down Cloud computing provides on demand scale up and scale down. This feature can be implemented on the application layer by adding a number of crawler agents or it can be implemented on cloud server by adding or reducing the number of virtual machines.

B. Geographically distributed web crawler A cloud vendor provides several reliable servers through a cloud operating system with servers located around the world. A reliable and geographically distributed server system allows a cloud-based web crawler to crawl the Internet based on the host location. This feature reduces network delay and costs in a way that is impossible in traditional systems that are usually located in one location or different locations without network reliability. Moreover, this feature allows a web crawler to decrease the Internet traffic because each crawler could work locally.

C. Cheap and faster crawler for a small businesses Frequently.
### VII. A COMPARISON BETWEEN THE PROPOSED WEB CRAWLER AND TRADITIONAL WEB CRAWLERS

<table>
<thead>
<tr>
<th>PROPOSED WEB CRAWLER</th>
<th>TRADITIONAL WEB CRAWLERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalability requires features that apply to both processing and storage. Our proposed web crawler supports both scalabilities (processing and storage) using the partition key and row key.</td>
<td>Scalability is one of the major challenges in a web crawler system.</td>
</tr>
<tr>
<td>Parallel agents can download unlimited web pages in a given time.</td>
<td>Traditional web crawlers can only download a limited number of web pages in a given time.</td>
</tr>
<tr>
<td>Distributed web crawler, each agent works separately and it does not have a bottleneck as a centralized system. Each agent writes its own results in a different partition.</td>
<td>A centralized model requires a massive rate of communication between agents and the coordinator.</td>
</tr>
<tr>
<td>when an agent writes a result on the table, other agents retrieve their relative’s data from its own partition rather than looking for a record from the whole database using PostgreSQL with two traditional tables tables to simulate the cloud-based table, and the cloud-based queue.</td>
<td>Storage: Existing web crawlers use traditional databases (relational databases) rather than SQL databases, which have completely different performance results for retrieving.</td>
</tr>
<tr>
<td>The proposed web crawler decreases communication rates and removes duplicate processing and stores in different partitions. Each agent could index a page, retrieve all URLs and add their own results to the queue and to the table. Each agent works on one host that is located in the same partition.</td>
<td>In traditional web crawlers communication rates is high because agents don’t maintain their own index table.</td>
</tr>
<tr>
<td>customer can customize, control and monitor resources.</td>
<td>customer could not customize, control and monitor resources.</td>
</tr>
</tbody>
</table>

### VIII. CONCLUSION

A web crawler plays a key role in search engines. A web crawler has highly intensive computation requirements and in order to retrieve massive amounts of data from the web and store unstructured data sets of indexed pages. A high-tech distributed technology such as Cloud Computing provides a required infrastructure for a web crawler. Cloud computing environments provide a distributed system that allows an application to use elastic resources on demand and store a massive amount of data in SQL databases with supporting unstructured data.

### REFERENCES