A Cloud-based Web Crawler Architecture

Mehdi Bahrami[1], Mukesh Singhal[2] and Zixuan Zhuang[3]

Cloud Lab
University of California, Merced, USA

[1]IEEE Senior Member, MBahrami@UCMerced.edu
[2] IEEE Fellow, MSinghal@UCMerced.edu
[3]ZZhuang@UCMerced.edu

***Abstract*—Web crawlers work on the behalf of applications or services to find interesting and related information on the web. For example, search engines use web crawlers to index the Internet. Web crawlers have several challenges, such as complexity between links and highly intensive computation requirements when a web crawler wants to retrieve complex connected links. Another issue is the storage of a massive amount of indexed links or downloaded unstructured data, such as binary files, videos or images. As the volume of information on the Internet increases rapidly and requests may search data in a variety of formats including unstructured data, no cloud-based architecture exists in the literatures for web crawlers that could effectively address both highly intensive computing and storage issues. 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 MapReduce 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 Cloud Azure Table (NoSQL database). The proposed web crawler also could store unstructured and massive amount of data on Azure Blob storage. We analyze the performance and scalability of the proposed web crawler and we describe the advantages of the proposed web crawler over traditional distributed web crawlers.**

Keywords— ***web crawler; cloud computing; big data; multimedia web crawler; cloud-based web crawler.***

I. INTRODUCTION

The Internet has grown exponentially since the late 1960s. The Internet is used by more than 2.4 billion people as reported in 2012 [1]. Globally, the Internet traffic will reach 14 gigabytes per capita by 2018, up from 5 GB per capita [2]. 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, we 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 commuting 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) [3] 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 analyze, because the users can add new resources (processor and storage capacity) on demand.

The rest of the paper is organized as follows: in Section II, we review the literature of traditional web crawlers and explain the main issues of each existing web crawler. In Section III, we introduce the requirements of a web crawler. In Section IV, we introduce our proposed cloud-based web crawler, present its architecture, and outline its implementation in Azure Cloud Platform. In Section V, we provide experimental results. In Section VI, we discuss advantages of the proposed web crawler. In Section VII, we present a comparison between the proposed cloud-based web crawler architecture and other existing web crawlers. Finally, in Section VIII, we conclude our proposed research.

II. BACKGROUND AND RELATED WORK

National Institute of Standards and Technology (NIST) defines cloud computing as "a model for enabling ubiquitous, convenient, on demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [4]. 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.

Since the first crawler deployed by Matthew Gray's Wanderer in 1993 [5], web crawlers have been an active topic of research. 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 [6] 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-labeled 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.

Codenotti et al. [7] introduced Ubicrawler, a scalable, fault-tolerant and fully distributed web crawler. This is a part of their project to collect and mine large data sets over the web. They parallelize the crawling process and decentralize the crawling tasks to implement fault tolerance and scalability. 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 Heydon et al. [8] 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 [9] 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.

In industry, some companies provide a cloud-based web crawler to their customers without any accessible documentation. For example, 80legs[1] which is based on grid computing, Prompt Cloud[2] and Scrapy Cloud[3] which are based on cloud computing, provide an API and platform to their customers to run a web crawler on the vendor's infrastructures. However, the number of parallel agents are limited to less than 10 agents and do not allow a customer to customize, control and monitor resources, as provided in our proposed web crawler.

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 [10]. 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. In our proposed web crawler, first, we partitioned the web by site-hash-based (e.g., subdomain.domain) and assigned to an agent. Second, each site is partitioned based on the URL-hash-based.

The second requirement is job division mode. Most of distributed crawlers use exchange mode, allowing agents to communicate each other and exchange information. Users are allowed to use firewall and cross-over modes when they need. The firewall mode does not accept information exchange among different agents. Agents can only crawl pages in certain partitions in this mode. In cross-over mode, agents are only allowed to follow the inter-partition links when they finish their own sessions [11]. In our proposed web crawler, we provide a new method to remove exchange communications and a coordinator. The proposed method uses the MapReduce programing technique which is described in Section IV.

Another parameter is a focused or an unfocused web crawler. Unfocused crawlers search the whole Internet, while focused crawlers only retrieve pages in certain web zone [12]. It is necessary to allow users to switch between focused and unfocused, because sometimes users need to search the whole Internet and sometimes only a limited zone.

The distributed system provided by cloud computing is a key to our web crawler and allows us to obtain scalability, fault tolerance and high performance computing.

Scalability is very important for a web crawler. As other distributed crawlers, our proposed web crawler also expects the performance to grow linearly with the numbers of requests; however, the advantage of our system is that we can scale up easily by adding agents on demand.

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 A_i crawls a URL and adds its retrieved results (a list of found URLs).

---

[1] http://80legs.com/

[2] http://promptcloud.com/index.php

[3] http://scrapinghub.com/platform

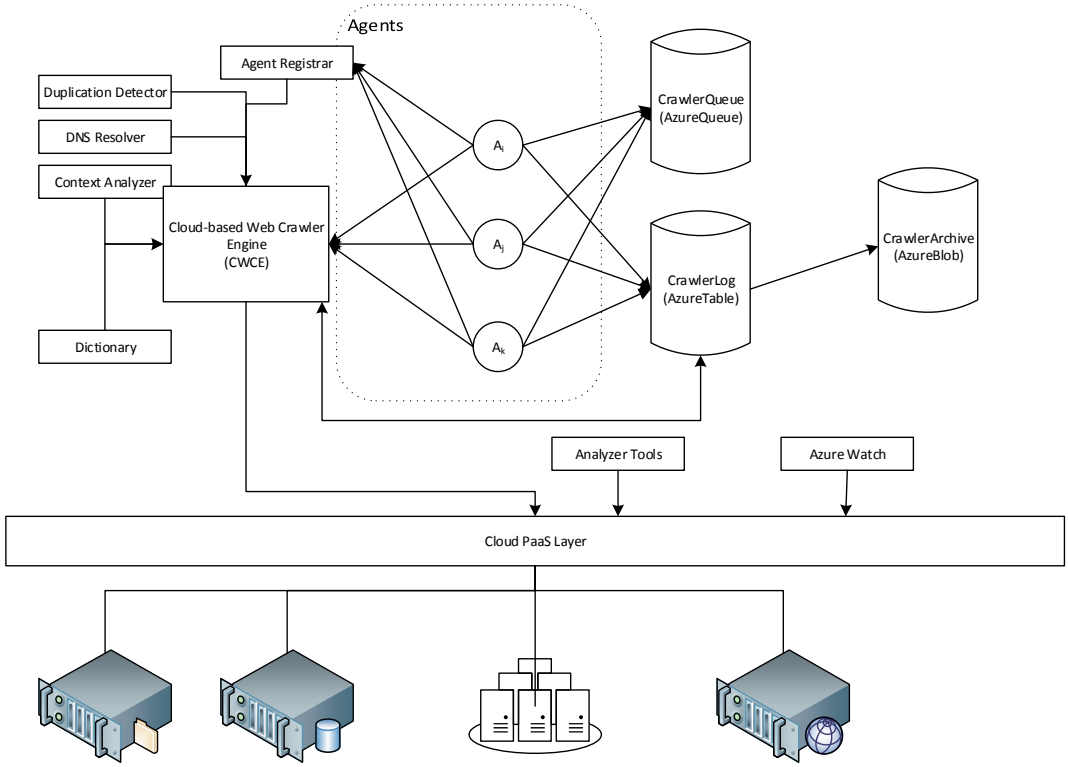

Figure 1. *An architecture of the proposed cloud-based web crawler*

The proposed web crawler, *Cloud-based web crawler engine (CWCE),* uses *Azure Cloud Queue* (*Queue*) 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. We use *Azure Cloud Table* (*Table*) for storing permanent information about crawled pages. Across the Internet, fields differ from record to record. The *Table* is based on NoSQL database, allowing us to define a new field on-the-fly when we 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.

The main fields which we use for the proposed web crawler are described in Table 1.

Table 1. *A static part of the Table*

Partition Key	Row Key	URL	Visit	Hit
String	String	String	Boolean	Int32

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 we 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).

We added two fields (partition key and row key) based on MapReduce programming technique that allows us to have a fully distributed system. For example, the first two rows in Table 2 will save in one partition and the third row could be saved in another partition because they have different hosts. The *Azure Table 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 2, an agent (A_1) works on a partition that crawls all links of "*ssha.ucmerced.edu*" host and agent A_2 works on another partition that crawls all links of "engineering.ucmerced.edu" host. Agent A_1 crawls on two unvisited URLs, listed as follows:

"http://ssha.ucmerced.edu/about" and
"http://ssha.ucmerced.edu/about/contact-us".

Table 2. *A sample records in Azure Table*

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*")

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

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.

In the following section, we review the major components of the proposed web crawler, as illustrated in Figure 1.

In this architecture, we used the Azure cloud queue (*queue*) to maintain a temporary list of URLs that requires to crawl and Azure Cloud Table (*table*) to maintain permanent information of crawled URLs.

DNS Resolver maintains a list of URLs that the web crawler is required to crawl. This list could be limited to specific URLs, providing a focused-based crawler.

Cloud-based web crawler engine (CWCE) is the main component of the proposed web crawler. The *CWCE* is responsible to start the crawling process by creating an agent. Then, based on *CWCE* instructions, an agent can add new agents for different zones of the web during the crawling process.

In the initialization step of *CWCE*, the first agent creates and fetches a URL from the *DNS Resolver*. If the fetched URL is not in the *queue* and has never been visited (if the value of the visit field of the URL in the *table*=="False"), then the URL is added to the *queue* and to the *table* as an unvisited URL (the value of the visit field of the URL in the *table*="False"). If the URL is already crawled (if the value of the visit field of the URL in the *table*=="True") the *CWCE* ignores the URL, retrieves another URL from the *DNS Resolver* and repeats all steps.

After initialization steps, the following steps are executed in an infinite loop until the *CWCE* stops:

(i) fetch the top URL from the *queue* as U_i;

(ii) if an agent registers for the partition of the fetched URL (agent has the same zone as the host of URL), then the fetched URL is ignored (because corresponding agent will crawl the URL). Otherwise, the *CWCE* controls the maximum number of agents (*Max*) and the number of alive agents (A_{alive}) from *Agent Registrar* by the following statement:

If $Max=A_{alive}$, then the *CWCE* will wait until some of agent back to alive set and $Max>A_{alive}$. Otherwise, *the CWCE* will create a new agent for the partition of the URL and the *CWCE* removes the fetched URL from the *queue*.

Each agent works on one partition. Each agent fetches a URL (U_i) from the *queue* (if the partition of U_i is the same as partition of the current agent), and the agent retrieves a list of URLs $\{U_j..U_k\}$ from U_i. The agent checks availability of the selected URL in the *queue* for each U_j to U_k. 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 the agent will check the availability of the URL partition key, 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 Microsoft SQL database, maintains agent information such as agent name and its partition key. *Page Index DB* is based on Azure table database which is a NoSQL 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 2), 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.

Azure Blob (Binary Large Object) provides a NoSQL 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.

Analyzer Tools and *Azure Watch* components 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 i*th* is denoted as A_i. Agent A_i can retrieve a list of URLs. According to A_i's finding, a new instance of an agent will crawl different zones of the web. As illustrated in Figure 2, each agent of A_i is responsible for: (i) crawling a web host to index and to find all intra-links (URLs that address to the same host) and all inter-links (URLs that address to other hosts); (ii) crawling on each intra-link and adding each inter-link to the *queue*.

For example, in Figure 2, agent A_1 retrieves three URLs, including two inter-links, which are added to the *queue* and are assigned to the agents A_2 and A_3, and one intra-link ($A_{1.1}$) which is assigned to A_1. Agent A_2 retrieves two inter-links and they are assigned to A_5 and A_6, respectively. Agent A_3 retrieves an intra-link and it is assigned to A_7. The agent A_7 also retrieves a link which is assigned to A_3 (a recursive link between two pages on different partitions). $A_{1.1}$ which is based on A_1 (because they have the same host name) retrieves A_8 as an inter-link and two intra-links, $A_{1.1.1}$ and $A_{1.1.2}$. Finally, $A_{1.1.2}$ retrieves A_1 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

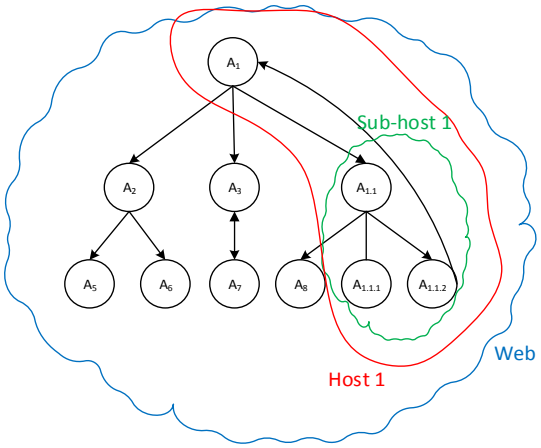

Figure 2. *An example of web crawling graph*

from one page of a URL, such as incoming URL, outgoing

URL, enumeration link dependency, PageRank and content analysis.

For instance, A_1 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, $a(i)$ to retrieve an authority score of $page_i$ which shows the number of citations (a number of URLs that point out to $page_i$), and second, $h(i)$ 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. E represents each link in crawling graph (dependency graph between links as shown in Figure 2) between $Page_i$ and $Page_j$.

$$a(i) = \sum_{(j,i)\in E} h(j) \qquad (1)$$

$$h(i) = \sum_{(i,j)\in E} a(j) \qquad (2)$$

Distributed agents can be implemented at different levels of a cloud 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 [15] can be implemented to avoid retrieve all links.

Furthermore, Windows Azure supports scaling up and a web crawler administrator could use *Azure Watch* 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

We implemented the main component of the proposed cloud-based web crawler in Microsoft Windows Azure platform to study its performance. The web crawler application connects to the AppFabric at PaaS layer of Windows Azure Platform. The AppFabric is responsible for resource requirements of the web crawler by assigning a pool of virtual machine resources, network servers and databases. In the case of lack of resources, an administrator could scale up or scale down computing resources and storage resources via the Azure dashboard.

The main component of the proposed architecture is available online[4]. The component can run as several parallel agents in Azure Platform or on localhost server with Azure Compute Emulator. In this experimental result, we focus on implementation the architecture rather than increasing the hit number.

A. Experimental Setup

In this experiment, we developed the web crawler by several parallel agents. Each agent in Microsoft .Net is called *Worker Role* (Virtual Machine instance) and it runs an infinitive loop thread that retrieves a URL from the queue and it crawls the URL. All agents use *Azure cloud-based queue* and *Azure cloud-based table* (NoSQL database) as shared resources to retrieve the next URL and save the results. The implemented web crawler is a type of a focus-based web crawler that focused on "*UCMerced.edu*" host. The first agent starts by crawling "*http://ucmerced.edu*" URL; the results (retrieved links) are added to the queue and to the table as discussed in Section IV. Other parallel agents start by fetching a URL from the Queue. The first agent and other parallel agents retrieve future links for crawling from the *queue*.

We ran several parallel agents on a virtual machine (VM) concurrently. Each VM is an emulator of a particular computer system and it is defined by CPU speed (number of cores), RAM capacity, and the size of disk. We ran our web crawler in one, two and three VMs with small instance configuration. Each small instance is defined by Microsoft as a CPU with 1 core (~1.6 GHz clock speed of an Intel i7 processor), 1.75 GB memory RAM and 70 GB disk size at Microsoft U.S. west datacenter. Figure 3 illustrates an implementation model of several parallel agents (workers role) in Windows Azure. A user has to pay for a subscription that indicates the maximum number of VMs and its configurations. Each VM could run several workers role. Currently, each VM could run the maximum of 25 worker role by Azure. Each VM is configured with a virtual CPU, virtual RAM and virtual Disk storage.

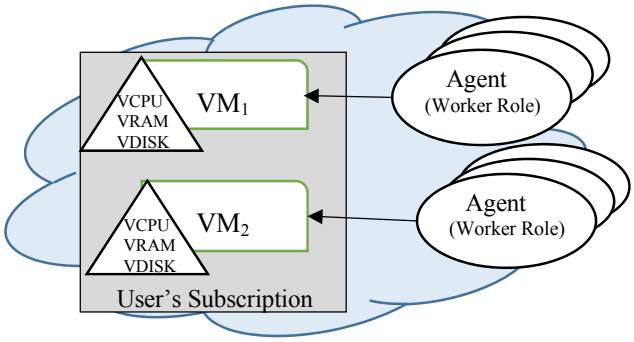

Figure 3. *A model of implementation of parallel agents on Windows Azure*

An administrator could configure the web crawler to add new agents by adding a new VM or upgrading a VM's configuration on demand. The application would run on local machine by using Microsoft Visual studio with Azure .Net 2013 SDK[5] that includes Azure emulator to run an application on a localhost. The application could run on Microsoft data centers by publishing the compiled codes. The web crawler architecture does not have any limitations on the number of agents but the number of agents based on type of cloud user's subscription is limited by Windows Azure. An administrator

[4] Available at http://cloudlab.ucmerced.edu/cloud-based-web-crawler

[5] Software Development Kit is a set of tools for developing software applications

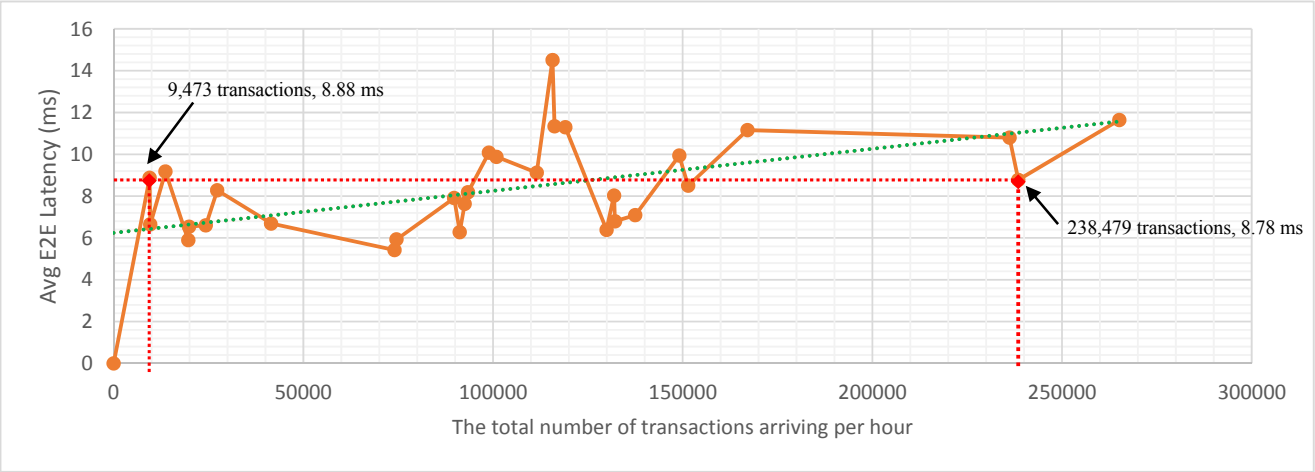

Figure 4. *Average E2E Latency for Cloud Table*

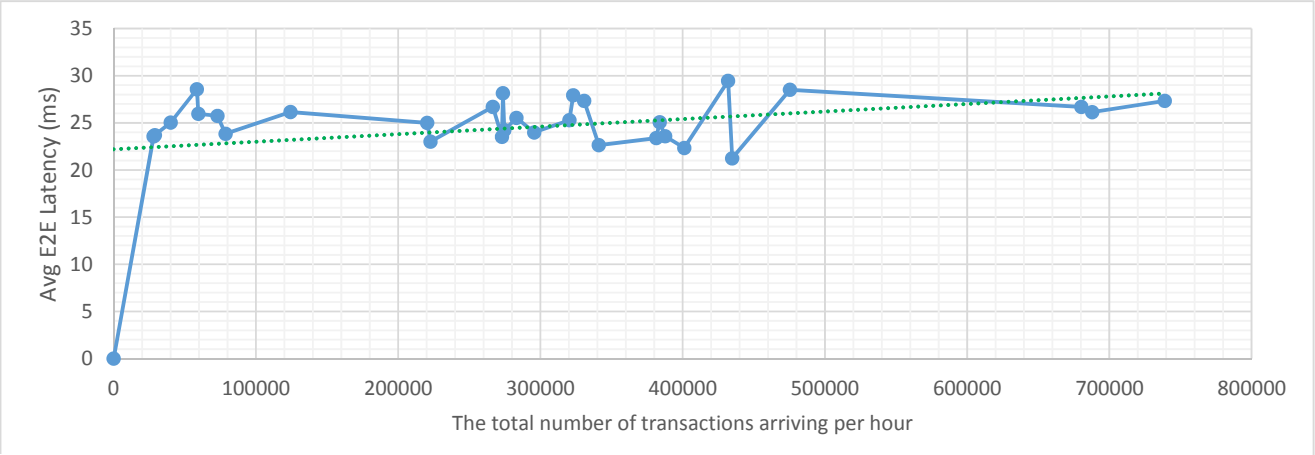

Figure 5. *Average E2E Latency for Cloud Queue*

could monitor the number of agents and resource usage and the administrator could add agents and resources on demand.

In this experiment, we 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.

B. The result of experiment

We monitored different arrival rate of transactions on both table and queue. The transactions are mixed of different queries that ran on both storages, such as insert, retrieve and delete records. The result shows in Figure 4 and Figure 5. These figures show the scalability of the web crawler for *cloud queue* and *cloud table*, respectively. In these figures, X-axis represents the rate of transactions arrival per hour on a *cloud table* (maximum ~75 transactions per second) / *cloud queue* (maximum ~222 transactions per second) and Y-axis represents the Average of End-to-End Latency for *cloud table* and *cloud queue*. The Average E2E Latency includes the required processing time of a transaction on table/queue within Windows Azure Storage to read a request, send the response, and receive acknowledgement of the response. The figures demonstrate scalability of the proposed web crawler when the rate of transactions arrival is increased.

The result shows that when the arrival rate of transactions increases, the response time (Average E2E Latency) remains fairly flat. As shown in these figures, Average E2E Latency does not increase much when using the arrival rate of transactions (workload) increases. For example, we ran 9,473 transactions in 8.8 milliseconds Average E2E Latency (red line in Figure 4) which is similar to running 238,479 transactions with 8.78 milliseconds Average E2E latency, when the arrival rate is increased 3 times.

We calculated the average response time over all the transactions for queue and table by using Equation (3).

$$Avg.\,Response\ Time = \frac{\sum Avg.E2E_i * T_i}{\sum T_i} \qquad (3)$$

where $Avg.E2E_i$ is the response time of ith hour and T_i is the arrival rate of transactions in ith hour.

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. As described in Table 3, a host server could return different HTTP codes to a web crawler. In

this experiment, we only considered pages that return the 200 response code.

Table 3. *HTTP response codes*

HTTP Code	Description
200	OK
404	Not found
302	Moved temporary
301	Moved permanently
401	Unauthorized
403	Forbidden

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. For example, the web-based crawler allows the administrator to add new virtual machines (instances) or increase the size of cloud storage that covers all databases (*Cloud Queue, Cloud Table* and *Blob*).

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. For example, a web crawler agent could run on servers in Europe to crawl European hosts, such as Microsoft.eu and another agent could run on servers in North America servers to crawl the Microsoft.com hosts. 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, a web crawler requires expensive servers and high bandwidth Internet access because it must send out a massive amount of requests and receive responses from each host. As such, many users are not able to provide the system requirements to support their own web crawler. Cloud vendors enable a small business to have their own web crawler to retrieve information of interest on the Internet. Major cloud vendors provide high power servers, a high bandwidth Internet access and a reliable network between servers.

VII. A COMPARISON BETWEEN THE PROPOSED WEB CRAWLER AND TRADITIONAL WEB CRAWLERS

All major search engines have their own web crawlers with copyright surveillance and their architectures are not public. The available web crawlers in academic literatures are Mercator [9] which is used by AltaVista[6] search engine, a Distributed Crawler (DC) [13], and Atrax [16] which is an extended version of Mercator. In the following section, we compare our proposed web crawler against these available traditional web crawlers.

Scalability is one of the major challenges in a web crawler system. Scalability requires features that apply to both processing and storage. Our proposed web crawler supports both scalabilities (processing and storage) using the methods described in Section IV.

Processing: Mercator is a centralized web crawler that collects a massive amount of communication data from parallel agents. Mercator and DC use a centralized model to collect data from parallel agents. Mercator uses random I/O to collect data. The DC improves the I/O issue by adding a cache to the web crawler. However, a centralized model requires a massive rate of communication between agents and the coordinator. In our proposed 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 (which is based on MapReduce technique). For example, in our proposed architecture, agent A_1 could find a URL that should be assigned to agent A_2, and agent A_1 will write its own results in agent A_2's partition on the *table*. Then, agent A_2 is able to look up in its partition to find a new un-crawled record without any communication with a coordinator to look up the new record (from I/O in Mercator and from cache in DC) and return the required information to the agent by the coordinator.

Storage: Existing web crawlers use traditional databases (relational databases) rather than NoSQL databases, which have completely different performance results for retrieving structured and unstructured data from a massive amount of data. In our proposed architecture, 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 (like a traditional database transaction). For instance, we developed a Microsoft SQL database with two traditional tables to simulate the *cloud-based table,* and the *cloud-based queue*. During the crawling process, we added data to both SQL and cloud storage. We monitored 18233 transactions on a query which is retrieved data from both storages. To compare the performance of both storages in a fair manner, we used the same network access for clients to the storages, and we used cloud storage in Microsoft US West datacenter and a Microsoft SQL database in a US based datacenter. We ran several local cloud emulators (clients) to run the same query on these storages. As illustrated in Figure 6, the result of this experiment shows the cloud storage provides a better performance results over the SQL database for different numbers of parallel web crawler agents.

Indeed, this performance is a hidden parameter behind all web crawler architectures because all traditional web crawlers have been evaluated by employing traditional databases. In our proposed web crawler, the web crawler is able to handle a large data set with a better performance by employing a NoSQL database over traditional web crawlers.

In other words, removing a coordinator from the web crawler is one of the key features of the proposed architecture. 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. If an agent dies

[6] AltaVista was purchased by Yahoo and shut down by Yahoo on July 8, 2013. reference: Wikipedia.org

during the crawling process (e.g., crashes, unexpected closing or network issues), the parent agent could control current crawling activities, and create or add a new agent to finish the remaining tasks. Each agent, rather than using a blind write on the *table* or request information from a coordinator, uses the host of a URL and the hash of the URL to distribute data (see Section IV). This distribution method removes duplicate tasks between parallel agents and the coordinator. By removing these tasks, the web crawler is able to decrease communication rate during the web crawling process because parallel agents use the *table* (which is partitioned based on the host of a URL) to save and retrieve data to/from a partition of the storage.

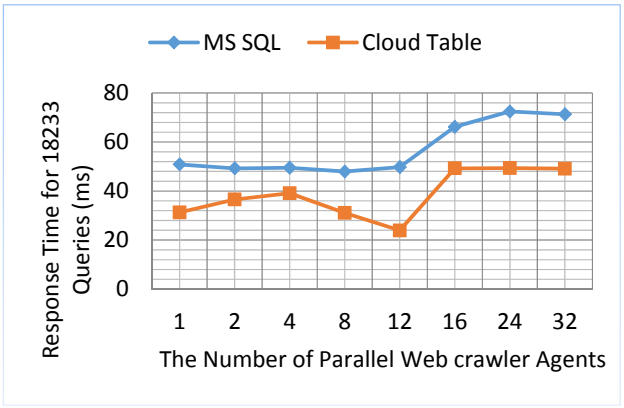

Figure 6. *A comparison between running a query on a cloud-based table and a traditional SQL database*

As we described before, although, some cloud-based web crawlers have been introduced by different companies, the number of parallel agents of these cloud based web crawler are limited to less than 10. We do not have access to their web crawler architectures but have seen that a customer could not customize, control and monitor resources, which are the features provided in our proposed web crawler.

Privacy in cloud computing is another challenge and can be implemented for an agent, such as light-weight models [17] but this paper did not consider this issue.

The summary of the comparison between the proposed web crawler and other related works is described in Table 4.

Table 4. *Comparison summary*

Web crawler	Architecture	Limitation
Mercator	Distributed	SQL, Coordinator, bottleneck
DC	Distributed	SQL, Coordinator, bottleneck
Atrax	Distributed	SQL, Coordinator, bottleneck
Parallel Crawler [13, 14]	Distributed	SQL, High communications rate
80legs, Prompt Cloud, Scrapy Cloud	Not Available	10 parallel agents
Proposed method	Distributed without coordinator	Unlimited (NoSQL, Distributed)

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 *NoSQL* databases with supporting unstructured data.

In this paper, we proposed a web crawler architecture based on cloud computing. The proposed web crawler is implemented in *Windows Azure Cloud Platform*. We used *Azure Cloud Queue* to make a connection among distributed web crawler's agents globally. Each agent provides its own results by fetching a URL from the queue and then by crawling the fetched URL. Each agent could add indexed pages by *MapReduce technique* in *Azure Table storage* which is based on a *NoSQL* database. Moreover, we suggested *Azure Blob* for collecting and storage unstructured data on cloud servers. Blob allows a web crawler to collect a massive amount of unstructured data, such as video, binary files and images. We discussed and compared our proposed web crawler architecture against traditional distributed systems and we explained the advantages of the proposed architecture.

ACKNOWLEDGEMENTS

The implementation work of cloud-based web crawler was supported by Microsoft Windows Azure through Windows Azure Educator Award. The authors would also like to thank Letha Goger at UC Merced for her feedback on this paper.

REFERENCES

[1] http://www.internetworldstats.com/stats.htm, retrieved on June 23, 2014.

[2] Cisco Visual Networking Index: Forecast and Methodology, 2013–2018, June 10, 2014. http://goo.gl/UgIFLS

[3] Dowd, Kevin. *High performance computing*. O'Reilly, 1993.

[4] Mehdi Bahrami and Mukesh Singhal, "The Role of Cloud Computing Architecture in Big Data", Information Granularity, Big Data, and Computational Intelligence, Vol. 8, Chapter 13, Pedrycz and S.-M. Chen (eds.), Springer, 2014 http://goo.gl/E1TmXu

[5] Edwards, Jenny, Kevin McCurley, and John Tomlin, "An adaptive model for optimizing performance of an incremental web crawler" Proceedings of the 10th international conference on World Wide Web. ACM, 2001.

[6] Xu, Songhua, Hong-Jun Yoon, and Georgia Tourassi, "A user-oriented web crawler for selectively acquiring online content in e-health research" Bioinformatics 30.1 (2014): 104-114

[7] P. Boldi, B. Codenotti, M. Santini, and S. Vigna, "Ubicrawler: A scalable fully distributed web crawler," Proc Australian World Wide Web Conference, vol. 34, no. 8, pp. 711–726, 2002. Online Available: citeseer.ist.psu.edu/article/boldi03ubicrawler.html

[8] Heydon, Allan, and Marc Najork. "Mercator: A scalable, extensible web crawler." World Wide Web 2.4 (1999): 219-229.

[9] Mika, Peter, and Giovanni Tummarello. "Web semantics in the clouds." Intelligent Systems, IEEE 23.5 (2008): 82-87.

[10] Ahmadi-Abkenari, Fatemeh, and Ali Selamat, "An architecture for a focused trend parallel Web crawler with the application of clickstream analysis" Information Sciences 184.1 (2012): 266-281.

[11] Liu, Jin-Hong, and Yu-Liang Lu, "Survey on topic-focused Web crawler" Application Research of Computers 10 (2007): 006.

[12] Najork, Marc, and Allan Heydon, "High-performance web crawling" Springer US, 2002.

[13] Cho, Junghoo, and Hector Garcia-Molina. "Parallel crawlers" Proceedings of the 11th international conference on World Wide Web. ACM, 2002

[14] Hsieh, Jonathan M., Steven D. Gribble, and Henry M. Levy, "The Architecture and Implementation of an Extensible Web Crawler", NSDI, 2010.

[15] Xu Zhang, Baolong Guo, Yunyi Yan, Chunman Yan ,"A novel classificatory filter for multiple types of erotic images based on Internet", International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 1, No. 1, pp. 44-50, 2011

[16] M. Najork, "Atrax: A distributed web crawler", Presentation given at AT&T Research, March 20, 2001.

[17] Mehdi Bahrami and Mukesh Singhal, "A Light-Weight Permutation based Method for Data Privacy in Mobile Cloud Computing", under review, 2015.