SPEEDING UP THE WEB CRAWLING PROCESS ON A MULTI-CORE PROCESSOR USING VIRTUALIZATION

Hussein Al-Bahadili[1], Hamzah Qtishat[2], Reyadh S. Naoum[2]

[1]Faculty of Information Technology, Petra University, Amman, Jordan
hbahadili@uop.edu.jo
[2]Faculty of Information Technology, Middle East University, Amman, Jordan
hamzah.qtishat@yahoo.com,rnaoum@meu.edu.jo

ABSTRACT

A Web crawler is an important component of the Web search engine. It demands large amount of hardware resources (CPU and memory) to crawl data from the rapidly growing and changing Web. So that the crawling process should be a continuous process performed from time-to-time to maintain up-to-date crawled data. This paper develops and investigates the performance of a new approach to speed up the crawling process on a multi-core processor through virtualization. In this approach, the multi-core processor is divided into a number of virtual-machines (VMs) that can run in parallel (concurrently) performing different crawling tasks on different data. It presents a description, implementation, and evaluation of a VM-based distributed Web crawler. In order to estimate the speedup factor achieved by the VM-based crawler over a non-virtualization crawler, extensive crawling experiments were carried-out to estimate the crawling times for various numbers of documents. Furthermore, the average crawling rate in documents per unit time is computed, and the effect of the number of VMs on the speedup factor is investigated. For example, on an Intel® Core™ i5-2300 CPU @2.80 GHz and 8 GB memory, a speedup factor of ~1.48 is achieved when crawling 70000 documents on 3 and 4 VMs.

KEYWORDS

Web search engine; Web crawler; virtualization; virtual machines; distributed crawling; multi-core processor; distribution methodologies; processor-farm methodology.

1. INTRODUCTION

A Web search engine is designed to help finding and retrieving information stored on the Web, where it enables users to search the Web storage for specific content in a form of text meeting certain criteria and retrieving a list of files that meet those criteria [1, 2].The size of the Web is currently witnessing an exponential growth, having over billions of pages, and it continues to grow rapidly at millions of pages per day. Such growth poses a huge challenge for today's generic Web search engines like Google and Yahoo[3].

Web search engine consists of three main components; these are: (1) Web crawler, (2) document analyzer and indexer, and (3) search processor [1]. When a user enters a query into a search engine (known as keywords), the Web search engine processes the query, then examines its index and provides a listing of best-matching Web pages according to its criteria, usually with a short summary containing the document's title and sometimes parts of the text [4].

The Web crawler is one of the most important and time consuming component of the Web search engine [5, 6]. It demands large amount of hardware resources (CPU, memory, and disk storage) to

DOI : 10.5121/ijwsc.2013.4102

crawl data from the rapidly growing and changing Web. The crawling process should be performed continuously in as short as possible time to maintain highest updatability of it search outcomes. Despite the fact that powerful computers and efficient crawling software are currently in use by Web crawlers, the largest crawls cover only 30–40% of the Web, and refreshment of the Web takes weeks to a month [3].

In order to speed up the crawling process, there are two main approaches. The first approach makes use of high-performance computers or distributed crawling on multiple computers interconnected in local or wide area networks or combination of both [7-12]. The second approach makes use of fast and efficient crawling computational architecture, algorithms, and models, and recognizes and crawl relevant sites and pages for a limited time [13]. This way, the performance bottlenecks on CPU and memory can be relaxed and fairly good speedups can be achieved.

Fortunately, there is an impressive gain in processor performance, due to advances in hardware technologies and also to innovation in processor architecture (i.e., how the processor is designed and organized to perform its computational tasks) [14]. One distinguish development is the introduction of parallelism in the architecture of the processors in the form of pipelining, multitasking, and multithreading leading to significant performance enhancement [15]. As a result of that a new type of relatively low-cost and powerful multi-core processor is emerged and widely-used in current computers. A multi-core processor has two or more microprocessors (cores) each with its own memory cache on a single chip [16]. The cores can operate in parallel and run programs much faster than a traditional single-core chip with a comparable processing power. Most manufacturers, such as Intel, offer now up to seven-core processor, which is known as i7-core processors; and as technology advances, the number of cores will continue to increase. Another major component of a computer is the software, which is categorized into operating system (OS) and application software. Current computer software is significantly improved to efficiently utilize the processing power of the emerged multi-core processors [16].

Due to its high speed, multi-core processors can play a big role in speeding up the crawling process for both standard and special Web search engines, and can be used as a major building block in constructing cost-effective high speed crawling system. This paperdevelopsand evaluates the performance of a new approach to improve the multi-core processor utilization and consequently speeding up the crawling process on such processors. The new approach utilizes the concept of virtualization, in which the multi-core processor is decomposed into a number of virtual machines (VMs) that can operate in parallel performing different crawling tasks on different data (Webpages or Websites).

In order to evaluate the performance of the new approach, two types of crawlers are developed. The first one runs on a multi–core processor with no virtualization (i.e., no VMs are installed on), therefore it is called no virtualization crawler (NOVCrawler). The second one runs on a multi-core processor with a number of VMs installed on each performing part of the crawling process; therefore, it is called distributed VM-based crawler (DVMCrawler). The parallel (distributed) programming methodology that is used in porting NOVCrawler to run efficiently on the VMs is the processor-farm methodology. Extensive crawling experiments were carried-out to estimate the speedup factor achieved by DVMCrawler over NOVCrawler for various numbers of crawled documents. Furthermore, the average crawling rate in documents per unit time is computed, and the effect of the number of VMs on the speedup factor is investigated.

This paper is divided into eight sections, where this section provides an introduction to the main theme of the paper, and the rest is organized as follows. Section 2 provides the reader with some background on two main topics covered in this paper, namely, Web search engine and virtualization. Review of some of the most recent and related work is presented in Section 3. The

developed VM-based distributed crawling system is described in Section 4. The main tools used in this work and the implementations of NOVCrawler and DVMCrawler are described in Section 5 and 6. Section 7 evaluates the performance of the developed VM-based distributed crawling system through a number of extensive crawling experiments. The obtained results are thoroughly discussed in Section 7. Finally, in Section 8, based on an in-depth study and analysis of the results some conclusions are drawn and a number of recommendations for future work are pointed-out.

2. BACKGROUND

This section provides a brief background on two of the main topics underlined in this paper, namely, the Web search engine and the virtualization.

2.1. Web Search Engine

Web search engines are an information retrieval systems designed to help finding information stored on the Web quickly [1]. These Web search engines execute user search queries at lightning speed and the search results are relevant most of the times, especially, if the user frame his search queries right. Figure 1 outlines the architecture and main components of a standard Web search engine model [17].

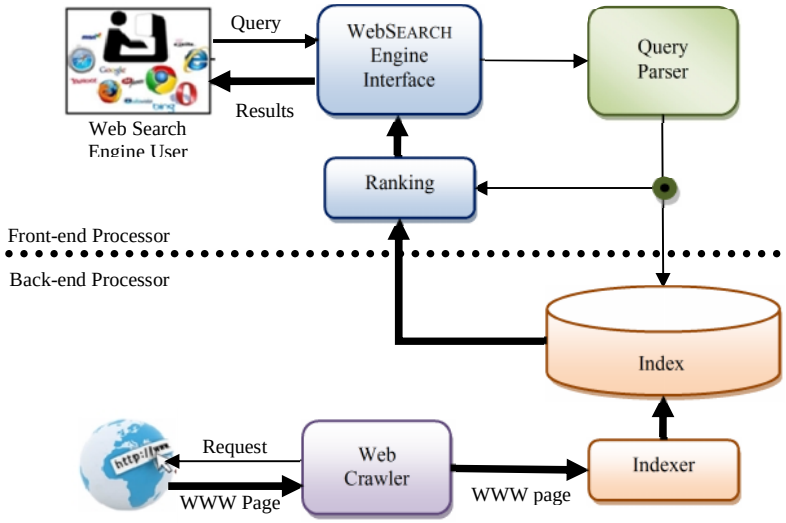


Figure 1. Main components of standard search engine model.

Web search engines can be classified into generic and specialized Web search engines. Example of generic Web search engines include: MSN, Yahoo, Google, etc. Examples of specialized search engines include: Truveo and Blinkx TV for finding video content from all video sharing Websites, Omgili for searching in discussions happening on public Internet forums and online mailing lists, Pipl for extracting personal information about any person from the Web, Ask Blog for searching content published on blogs (and other sites that offer RSS feeds). Another Web search engines with special features also include: TinEye, Wolfram Alpha, Alexa, Agame, BTJunkie, ChaCha, Crunch Base, Scour, Spokeo, Tweepz, and Address Search.

Standard Web search engine consists of Web crawler, document analyzer and indexer, and searching process [6, 17]. Consequently, Web search process can be broken up into three main sub processes as shown in Figure 2; these are:

- Crawling process (CP), which runs on the crawling processor (CPR) (CP → CPR).
- Analysis and indexing process (AIP), which runs on the analysis and indexing processor (AIPR) (AIP → AIPR).
- Searching process (SP), which runs on the search processor (SPR) (SP → SPR).

Since, we concern with the Web crawling process, in what follows, we shall briefly describe the Web crawler, its minimum requirements, and the general Web crawling strategies.

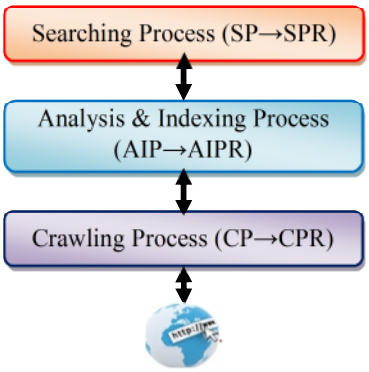


Figure 2.Web search process model.

2.1.1. Web Crawler

The Web crawler is an important component of the search engine [6]. It is a program that is used to read the Meta tags specified by the Website creator, find, download, parse content, and store pages in an index file. A crawler usually needs initial Web addresses as a starting point in order to index the information about these Websites. These addresses are the Websites Uniform Resource Locators (URLs), and they are called the seed URLs. After specifying these URLs, the crawler finds the hyperlink text and Meta tags in all pages of the Website until the text finishes [5]. Given a set of seed URLs, the crawler repeatedly removes one URL from the seeds, downloads the corresponding page, extracts all the URLs contained in it, and adds any previously unknown URLs to the seeds [2].It is important to know at this stage that most of the Websites are very large so it can take long time to crawl and index all the data. Furthermore, Websites change their contents frequently, so it is necessary to carefully consider this frequent change as when to revisit the page again in order to keep the index updatability. Further requirements for any crawling system may include: flexibility, high performance, fault tolerance, maintainability, configurability, etc [5].

Web crawlers are classified into specific crawler architecture and general-purpose (open source) crawlers. Examples of specific crawler architecture include: Googlebot is the Google crawler, Yahoo! Slurp is the Yahoo crawler, Bingbot is the Microsoft's Bing webcrawler (It is developed as a replacement to MSNbot), FAST crawler is a distributed crawler developed for Fast Search & Transfer, PolyBot is a distributed crawler, RBSE, WebCrawler, World Wide Web Worm, WebFountain, and WebRACE. Examples of general-purpose (open-source) crawlers, such as: Aspseek, GNU Wget, Datapark Search, GRUB, ICDL, Heritrix, HTTrack, PHP-Crawler, mnoGoSearch, Nutch, Open Search Server, tkWWW, YaCy, Seeks, and Larbin.

2.1.2. General Crawling Strategies

There are many crawling strategies that have been developed since the engagement of the Internet, such as [13]:

- *Breadth-first crawling*: a crawl is carried out from a list of Web pages (initial URLs or seeds)to build a wide Web archive similar to that of the Internet Archive. A breadth-first exploration is started by following hypertext links leading to those pages directly connected with these initial URLs. In fact, it is not as easy as it is explained above, Websites are not really browsed breadth-first and various restrictions are actually applied, e.g., limiting crawling processes to within particular sites, or downloading the pages believed most interesting first.
- *Repetitive crawling*: once pages have been crawled, some systems continue to perform the process periodically so that indexes remain updated. This may basically be achieved by launching a second crawl concurrently. Various heuristics exist to overcome this problem, such as: re-launching the crawling process of pages frequently, domains or sites considered important to the detriment of others. However, it can be easily recognized that a good crawling strategy is crucial for maintaining a constantly updated index list.
- *Targeted crawling*: specialized search engines use advanced crawling process heuristics in order to target a particular type of page, e.g., pages on a specific topic/language, images, audio files, video files or scientific papers. In addition to these advanced heuristics, more generic approaches have been proposed. They are based on the analysis of the structures of hypertext links and learning techniques: the objective here being to retrieve the greatest number of pages relating to a particular subject while using minimum bandwidth.
- *Random walks and sampling*: some researches have focused on the effect of random walks on Web graphs or modified versions of these graphs through sampling in order to estimate the size of documents on line.
- *Deep Web crawling*: a lot of data accessible through the Web are currently enclosed in databases and may only be downloaded through appropriate requests or forms. Lately, this often neglected. The Deep Web is a name given to a Web that contains such datbase.

2.2. Virtualization

Virtualization refers to hiding the physical characteristics of computing resources to streamline the way in which applications, users, or other systems interact with those computing resources [18]. It allows a single physical resource (e.g., servers or storage devices) appears as multiple logical resources; or makes multiple physical resources appears as a single logical resource. Virtualization also can be defined as the process of decomposing the computer hardware recourses into a number of VMs. A VM is a software implementation of computing hardware resources on which an OS or program can be installed and run. The VM typically emulates a computing hardware resources by formation of a virtualization layer, which translates requests to the underlying physical hardware, manages requests for processor, memory, hard disk, network and other hardware resources.

There are different forms of virtualization that have been developed throughout the years as shown in Figure 3; these are [19]:

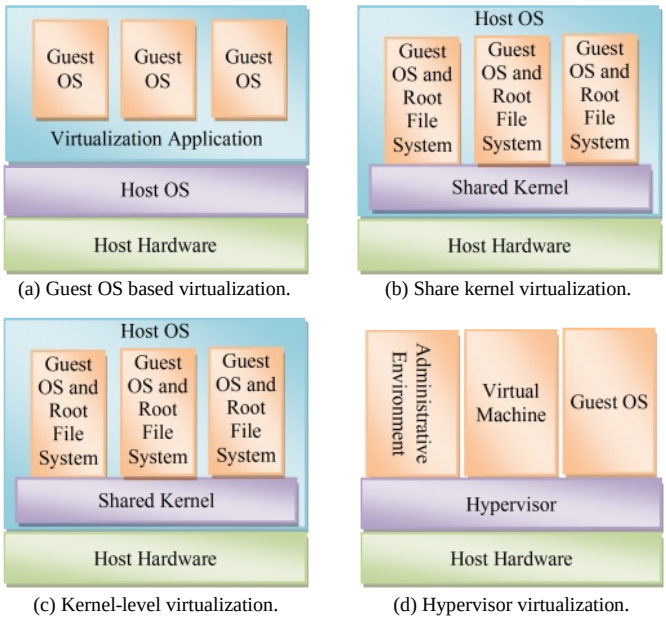
- *Guest OSbased virtualization.* Under this virtualization the physical host system runs a standard unmodified OS such as Windows, Linux, UNIX, or Mac OS. Running on this OS is a virtualization application (VA) which is executed as any application would run on the system. It is within this VA that one or more VMs are created to run the guest OSs on the host system. The VA is accountable for starting, stopping and managing the VM(s) and essentially controlling access to physical resources on behalf of the individual VM(s). Some examples of guest OS virtualization technologies include VMware Server and VirtualBox. Figure 3a provides an illustration of guest OS based virtualization.
- *Shared kernel virtualization.* Also known as system level or OS virtualization. Under this form of virtualization, each virtual guest system has its own root file system and all guest

systems share the kernel of the host OS. Examples of shared kernel virtualization include: Solaris Zones and Containers, Linux V Server, Open VZ and Free VPS. This structure is illustrated in the architectural diagram in Figure 3b.

- *Kernel-level virtualization.* Under which the host OS runs on a specially modified kernel which contains extensions designed to manage and control multiple VMs each containing a guest OS. Examples of this virtualization approach include: Kernel-based Virtual Machine and User Mode Linux. Figure 3c provides an overview of the kernel level virtualization architecture.
- *Hypervisor virtualization.* The x86 family of CPUs provide a variety of protection levels that are known as rings in which code can execute. The highest level is Ring 0, where the OS kernel normally runs. Code running in Ring 0 is said to be running in supervisor mode, kernel mode, or system space. All other codes such as applications running on the OS runs in less privileged rings, typically Ring 3.In hypervisor virtualization, a program known as a hypervisor runs directly on the host system in Ring 0to handle resource and memory allocation for the VMs in addition to providing interfaces for higher level administration and monitoring tools. Figure 3d illustrates the hypervisor approach to virtualization.

  Clearly, with the hypervisor running on Ring 0, the kernels for any guest OSs running on the system must run in less privileged Rings. Unfortunately, most OS kernels are written explicitly to run in Ring 0 for the simple reason that they need to perform tasks that are only available in that Ring, such as the ability to execute privileged instructions and directly manipulate memory. A number of different solutions to this problem have been proposed, such as: para virtualization, full virtualization, and hardware virtualization [20].

  Hypervisor virtualization solutions include Xen, VMware ESX Server and Microsoft's Hyper-V technology. In this paper, the Xen hypervisor is used as a virtualization platform and it will be discussed in more details later on.



(a) Guest OS based virtualization. (b) Share kernel virtualization.

(c) Kernel-level virtualization. (d) Hypervisor virtualization.

Figure 3. Block diagrams of various virtualization structures.

VMs can provide many advantages over directly installing OS's and software on physical hardware. Isolating applications ensure that applications and services that run within a VM cannot interfere with the host OS or other VMs. VMs can also be easily manipulated, e.g., moved,

copied, and reassigned between host servers to optimize hardware resource utilization. Administrators can take advantage of virtual environments to simply backups, disaster recovery, new deployments and basic system administration tasks. VMs also comes with several important management considerations, many of which can be addressed through general systems administration best practices and tools that are designed to manage VMs [18].

3. LITERATURE REVIEW

This section presents a review on most recent work related to reducing the crawling processing time. The reviewed work is presented in chronological order from the old to the most recent. Choet al. [21] studied the sequence in which the crawler should visit the URLs it has seen, in order to obtain important pages first. Obtaining important pages quickly can be very useful when a crawler cannot visit the entire Web in a reasonable amount of time. They defined several importance metrics, ordering schemes, and performance evaluation measures for this problem. They also experimentally evaluated the ordering schemes on the Stanford University Web. Their results showed that a good ordering scheme can help crawler obtains important pages significantly faster than one without.

Chakrabartiet al. [22] described a hypertext resource discovery system called a Focused Crawler (FC), which aims to selectively seek out pages that are relevant to a pre-defined set of topics. The topics are specified using exemplary documents not using keywords. Rather than collecting and indexing all accessible Web documents to be able to answer all possible ad-hoc queries, a FC analyzes its crawl boundary to find the links that are likely to be most relevant for the crawl, and avoids irrelevant regions of the Web. This leads to significant savings in hardware and network resources, and helps keeping updated crawler.

Chung and Clarke [23] proposed a topic-oriented approach, which partitions the Web into general subject areas with a crawler assigned to each subject area to minimize the overlap between the activities of individual partitions. They designed and examined the creation of a Web page classifier approach for use in this context; which is compared with a hash-based partitioning. The comparison demonstrated the feasibility of the topic-oriented approach, addressing issues of communication overhead, duplicate content detection, and page quality assessment.

Yanet al. [24] proposed an architectural design and evaluation result of an efficient Web-crawling system. Their design involves a fully distributed architecture, a URL allocating algorithm, and a method to assure system scalability and dynamic configurability. The results showed that load balance, scalability and efficiency can be achieved in the system. Their distributed Web-crawling system successfully integrated with Web Gather, a well-known Chinese and English Web search engine. They also suggested that their design can also be useful in other context such as digital library.

Shkapenyuk and Suel[12] described the design and implementation of a distributed Web crawler that runs on a network of workstations. The crawler scales to (at least) several hundred pages per second, is resilient against system crashes and other events, and can be adapted to various crawling applications. They presented the software architecture of the system, discussed the performance bottlenecks, and described efficient techniques for achieving high performance. They also reported preliminary experimental results based on a crawl of million pages on million hosts.

Takahashiet al.[25] described a scalable Web crawler architecture that uses distributed resources. The architecture allows using loosely managed computing nodes (PCs connected to the Internet), and may save communication bandwidth significantly. They demonstrated the importance of such

architecture, point-out difficulties in designing such architecture, and described their design. They also reported experimental results to support the potential of their crawler design.

Looet al. [26]developed a distributed Web crawler that harnesses the excess bandwidth and computing resources of clients to crawl the Web. Nodes participating in the crawl use a Distributed Hash Table (DHT) to coordinate and distribute search. They investigated different crawl distribution strategies and estimated the trade-offs in communication overheads, crawl throughput, balancing load on the crawlers as well as crawl targets, and the ability to exploit network proximity. They described the implementation of the distributed crawler using PIER, a relational query processor that runs over the Bamboo DHT, and compared different crawl strategies on Planet-Lab querying live Web sources.

Hafriand Djeraba[27] developed a real-time distributed system of Web crawling running on a cluster of machines. The system is platform independent and is able to adapt transparently to a wide range of configurations without incurring additional hardware expenditure, it includes a high-performance fault manager and it can crawl several thousands of pages every second. They provided details of the system architecture and described the technical choices for very high performance crawling.

Expostoet al. [28] evaluated scalable distributed crawling by means of the geographical partition of the Web. The approach is based on the existence of multiple distributed crawlers each one responsible for the pages belonging to one or more previously identified geographical zones. For the initial assignment of a page to a partition they used a simple heuristic that marks a page within the same scope of the hosting Web server geographical location. During download, if the analysis of a page contents recommends a different geographical scope, the page is forwarded to the best-located Web server. A sample of the Portuguese Web pages, extracted during the year 2005, was used to evaluate page download communication times and overhead of pages exchange among servers. Evaluation results permit to compare their approach to conventional hash partitioning strategies.

Chau et al. [29] developed a framework of parallel crawlers for online social networks, utilizing a centralized queue. The crawlers work independently, therefore, the failing of one crawler does not affect the others at all. The framework ensures that no redundant crawling would occur. The crawlers were used to visit approximately 11 million auction users, and about 66,000 of which were completely crawled.

Ibrahim et al.[30] demonstrated the applicability of Map Reduce on virtualized data center through conducted a number of experiments to measure and analyze the performance of Hadoop on VMs. The experiments outlined several issues that will need to be considered when implementing Map Reduce to fit completely in a virtual environment.

Hsieh et al. [31] introduced extensible crawler, which is a service that crawls the Web on behalf of its many client applications. Clients inject filters into the extensible crawler; the crawler evaluates all received filters against each Web page, notifying clients of matches. As a result, crawling the Web is decoupled from determining whether a page is of interest, shielding client applications from the burden of crawling the Web them self's. They focused on the challenges and trade-offs in the system, such as the design of a filter language that is simultaneously expressive and efficient to execute, the use of filter indexing to cheaply match a page against millions of filters, and the use of document and filter partitioning to scale their prototype implementation to high document throughput and large numbers of filters. They claimed that the low-latency, high selectivity, and scalable nature of their system makes it a promising platform for taking advantage of emerging real-time streams of data, such as Facebook or Twitter feeds.

Anbukodi and Manickam[32] addressed problems of crawling Internet traffic, I/O performance, network resources management, and OS limitations and proposed a system based on mobile crawlers using mobile agent. The approach employs mobile agents to crawl the pages by identifying the modified pages at the remote site without downloading them; instead it downloads those pages which have actually been modified since last crawl. Hence it will reduce the Internet traffic and load on the remote site considerably. The proposed system can be implemented using Java aglets.

4. THE VM-BASED DISTRIBUTED WEB CRAWLING MODEL

This section describes the VM-based distributed Web crawling model. This model assumes that a multi-core processor is used as the main computing platform. The multi-core processor is divided into a number of VMs each acts as crawling processor. In particular, in this model, the crawling processor (CPR) is split into a number virtual crawling processors (v_c), one of them acts a master crawling VM (MCVM), and the rest acts as slave crawling VMs (SCVMs), each of the SCVMs access the Internet independently retrieving HTML pages and passes them to the MCVM. In this model, the SCVMs can communicate with the MCVM and also with each other under the control of the MCVM. Figure 4 shows the architecture of the VM-based distributed crawling model.

With proper resources configuration and job distribution, the model can ensure high resource utilization result in faster crawling process, and increase scalability, availability, and reliability. The new model presumes to speed up the crawling process leading to significant increase in crawling rate (pages per unit time) meeting applications and users growing needs.

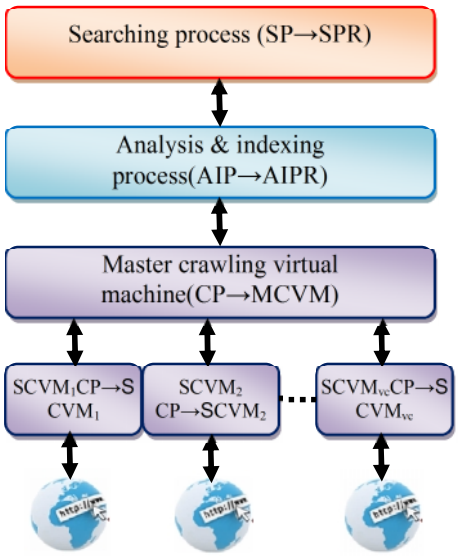


Figure 4.The architecture of the VM-based distributed Web crawling model.

5. TOOLS

This section presents a description of the implementation and evaluation of the proposed model. But, first, a brief introduction is provided for some of the tools and technologies that are used in this research, such as: Xen hypervisor, Nutch crawler, and Hadoop.

5.1. Xen Hypervisor Virtualization

Xen hypervisor is an open source standard for virtualization that offers a powerful, efficient, and secure features for virtualization of x86, x86_64, IA64, ARM, and other CPU architectures[19, 20]. It supports a wide range of guest OSs including Windows®, Linux®, Solaris®, and various versions of the BSD operating systems. The Xen hypervisor is a layer of software running directly on computer hardware replacing the OS thus allowing the computer hardware to run multiple guests OSs concurrently. The Xen hypervisor is developed and maintained by the Xen community as a free solution licensed under the GNU General Public License [33].

With Xen virtualization, a thin software layer, namely, the Xen hypervisor is inserted between the server's hardware and the OS. This provides an abstraction layer that allows each physical server to run one or more virtual servers, effectively decoupling the OS and its applications from the underlying physical server. In Xen virtualization, the same process occurs, with entire OSs taking the place of tasks. The scheduling aspect is handled by the Xen kernel, which runs on a level superior to the supervising guest OSs, and which is thus called the hypervisor. Xen hypervisor is not quite so simple OS, even the new version that has been modified to be Xen-friendly. It uses a different set of assumptions than applications, and switching between the traditional and friendly versions usually involves more complexity [33].

5.2. The Nutch Crawler

Nutch is a framework for building scalable Internet crawlers and search engines. Figure 5 illustrates the major parts as well as the workflow of a Nutch crawl [34].

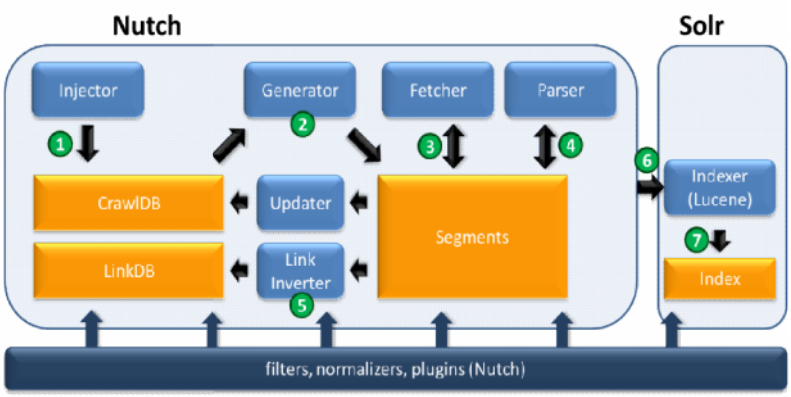


Figure 5. The workflow of the Nutch process [34].

The workflow of the Nutch crawler can be explained as follows:

1. The injector takes all the URLs of the nutch.txt file and adds them to the Nutch crawled database (CrawlDB). As a central part of Nutch, the CrawlDB maintains information on all known URLs (fetch schedule, fetch status, metadata, …).
2. Based on the data of CrawlDB, the generator creates a fetch list (FetchList) and places it in a newly created segment directory.
3. Next, the fetcher gets the content of the URLs on the FetchList and writes it back to the segment directory. This step usually is the most time-consuming one.
4. Now the parser processes the content of each Webpage and for example omits all HTML tags. If the crawl functions as an update or an extension to an already existing one (e.g. depth of 3), the updater would add the new data to the CrawlDB as a next step.

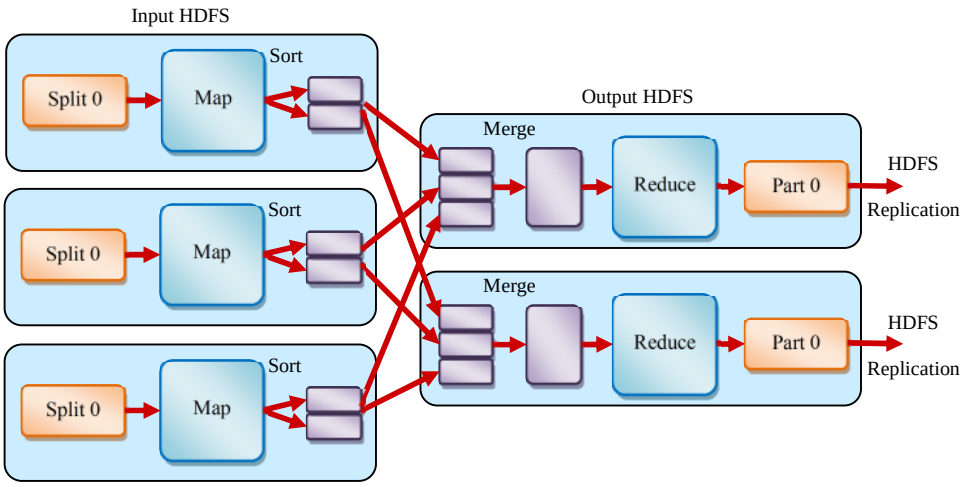
5. Before indexing, all the links need to be inverted, which takes into account that not the number of outgoing links of a Webpage is of interest, but rather the number of inbound links. This is quite similar to how Google PageRank works and is important for the scoring function. The inverted links are saved in the link database (LinkDB).
6. and (7). Using data from all possible sources (CrawlDB, LinkDB and segments), the indexer creates an index and saves it within the Solr directory. For indexing, the popular Lucene library is used. Now, the user can search for information regarding the crawled Web pages via Solr.

In addition, filters, no rmalizers and plugins allow Nutch to be highly modular, flexible and very customizable throughout the whole process. This aspect is also pointed out in the Figure 5 [34].

5.3. Hadoop

Apache Hadoop is a framework for running applications on large cluster of computing hardware [35]. The Hadoop framework transparently provides applications both reliability and data motion. Hadoop implements a computational paradigm named Map Reduce, in which the application is divided into many small computational fragments, each of which may be executed or re-executed on any node in the cluster. MapReduce works by breaking the process into two phases: the Map phase and the Reduce phase. Each phase has key-value pairs as input and output, the types of which may be chosen by the programmer. The programmer also specifies two functions: the Map function and the Reduce function. Figure 6 shows the dataflow in Map Reduce with multiple tasks [35].

Figure 6.MapReduce dataflow with multiple tasks.



When a dataset get too large for the storage capacity of a single physical machine, it becomes necessary to partition it across a number of separate machines. The file systems that manage the storage across a network of machines are called Distributed File Systems (DFSs). Because DFSsare required to communicate, all the complications of network programming take effect, which make networked DFSs more complex than regular disk file systems. One of the main challenges is making the file system allows for node failure without suffering extensive data loss.

Hadoop comes with a DFS called Hadoop DFS (HDFS) that stores data on the computing nodes, providing very high aggregate bandwidth across the cluster. Both HDFS and Map Reduce are designed so that the framework automatically handles nodes failures [35]. More details on using Hadoop with Nutch and some relevant details on how the Map Reduce paradigm is applied to a

data processing task in Nutch (e.g., link inversion, generation of fetch lists, invert, partition by host, sort randomly, and multithreading) can be found in [36].

Hadoop is meant to work with cluster of machines (at least two machines), and for the purpose of our work it should work on single machine. Fortunately, using the virtualization to create VMs within a single machine will give the Hadoop the illusion that its working on more than one machine (according to the number of VMs that have been created) and the virtualization isolation, managing resources features will support that, since every VM will work in isolation of any other VM.

6. IMPLEMENTATION

This section describes the two crawlers developed in this work, which are:

1. NOVCrawler. A non-virtualization crawler that runs on a computer comprises a multi-core processor with no VMs installed on (no virtualization).
2. DVMCrawler. A distributed VM-based crawler that runs on a computer comprises a multi-core processor with a number of VMs installed on performing distributed crawling.

6.1 Implementation of the NOVCrawler

The Web crawling process that is considered in this research and implemented in NOVCrawler can be summarized as follows:

1. Inject URLs to the Nutch crawled database (CrawlDB). This could be any number of URLs, which are injected as a seed for the crawling process. The injection process is performed only once during the whole crawling process.
2. Generate a Nutch segment by copying the top ranked URLs in the Nutch CrawlDB into the Nutch segment. The number of URLs in the segment must be limited to a specific number of URLs. The first segment will contain only the seed URLs, the ones that are injected in Step 1.
3. Fetch the Nutch segment from the Web using the Nutch Fetcher, which fetches the specific number of URLs, except the first time it only fetches the seed URLs only.
4. Parse the Nutch segment for the sake of updating the rank and the amount of URLs in the CrawlDB.
5. Update the CrawlDB by adding the new URLs and updating the ranking and the status of URLs (fetched or unfetched).
6. Read the CrawlDB to determine number of crawled documents (fetched and unfetched).
7. Repeat Steps 2 to 6 until a pre-specified number of documents is crawled from the Web.

The fetching phase (Step 3) is the only phase that uses parallel processing through threading to fetch the segment URLs from the Web, also in this phase; Nutch is using the file system of the machine directly to save data or to get the input data for the crawling processes.

We modify the default configuration of Nutch to tune and improve the performance as follows:

1. Enable redirection: Nutch by default don't follow redirection that is if the URL stored in the CrawlDB is redirected to another domain or URL, Nutch will not follow the redirection to fetch the new URL. However, we enable redirecting by re-configuring Nutch to follow new URLs.

2. Limit the number of threads in the fetching phase to a certain number: The default number of threads is 10 for each fetching phase, to better utilize this feature, in our case, we changed it to 30 threads per fetch, which is a number estimated through trial and error.

6.2 Implementation of the DVMCrawler

The distributed programming methodology used in porting NOVCrawler to run on a distributed environment, VM-based system, is the processor-farm methodology [37]. In the processor-farm methodology, the VMs are configured in master-slave architecture, where one of the VMs is run as a master (MCVM), while the other machines are run as slaves (SCVM). The program running on the master VM is called the master program, while the program running on the slave is called the slave program.

The standard relationship between the master and the slaves programs can be summarized as follows:

1. The master program reads-in the input data (URLs) and perform any require preliminary computations.
2. Send part of the URLs to the appropriate slave program (that may include transferring data (URLs) from one slave to another).
3. After completion of the fetching process, the slave program sends the output results back to the master program.
4. The master program performs any post-processing computations and then presents the final outputs.

It is clear from the above relationship that the processor-farm involves no inter-processor communications other than in forwarding data/results between the master and the slaves and the slaves between each other to make the data required for computation available in the selected slave for the task, once for all at the beginning and the end of the computation. Furthermore, slaves are only allowed to communicate if the master orders them to move data between them as required to utilize idle processors or handling failures.

In this methodology, the URL space is divided into disjoint sets (i.e., subsets), each of which is handled by a separate crawler. Each crawler parses and logs only the URLs that lie in its URL space subset, and forwards the rest of the URLs to the corresponding crawler entity. Each crawling node will have a pre-knowledge of the look up table relating each URL subset to [IP:PORT] combination that identifies all the crawler threads. This methodology also helps with avoiding the important issue of scalability as we can create VMs as many as we need.

The main challenge while using this methodology is to efficiently distribute the computation tasks avoiding overheads for synchronization, maintenance of consistency, and load balancing. Load balancing needs to be carefully considered to avoid keeping some VMs overloaded doing a lot of work while other VMs idle while waiting.

7. RESULTS AND DISCUSSIONS

In order to evaluate the performance of the VM-based distributed crawling model, a number of performance measures are computed, such as: the speed up factor (S) achieved due to the introduction of virtualization, i.e., installation of different number of VMs on the same multi-core processor. In this work, S is calculated as the ratio between the CPU time required to perform a specific crawling computation on a multi-core processor with no VMs installed on (i.e., no

virtualization) (T_s) and the CPU time required to perform the equivalent computation concurrently on the same processor with a number of VMs installed on (T_d). So that S can be expressed as [38]:

$$S = \frac{T_s}{T_d} \quad (1)$$

The speedup depends largely on the relative time spent on computation and communication. Computation is distributed between processors and communication takes place between processor and depends on how the whole computation is distributed. Thus, to obtain ultimate performance three points must be considered:

1. Load balancing. It is necessary to ensure that as many of the VMs as possible are doing useful work for as much of the time as possible.
2. Data transfer (communication time). It must be reduced to its lowest levels.
3. System topology. The way in which the various VMs are linked together can strongly affect the speedup. In theory, the hardware configuration should map the software configuration in well design system. In practice, the limitations imposed by hardware and software can make the situation rather different.

It is well recognized that the document crawling time depends on the documents content and network performance, and it is increases with increasing number of crawled documents. Therefore, we have found it is necessary to estimate the average document crawling time (t_{avg}), which is the average time required to crawl one document, and it can be expressed as:

$$t_{avg} = \frac{T_x}{W_a} \quad (2)$$

Where $W_a$is the actual number of crawled documents, and $T_x$is the crawling time. T_x is taken as T_s for no VM-based crawling and T_d for VM-based crawling. Furthermore, we define a another parameter, namely the average crawling rate (R_{avg}), which represents the average number of documents crawled per second (dps), and it is calculated by dividing the actual number of crawled document during a specific crawling time by the crawling time. In other words, R_{avg} represents the reciprocal of t_{avg}, which can be expressed as:

$$R_{avg} = \frac{1}{t_{avg}} = \frac{W_a}{T_x} \quad (3)$$

The computer used in this work comprises a high-speed single multi-core processor. The specifications of the computer are given in Table 1.

Table 1. Specification of the computing system

#	Components	Specifications
1	Processor	Intel® Core™ i5-2300 CPU @ 2.80 GHz
2	Memory	8 GB RAM
3	Hard-disk	1.5 TB Hard Drive
4	Operating System (OS)	Debian Squeeze (open source Linux-based OS for server and desktop computers

In order to compute *S*, NOVCrawler and DVMCrawler are used to crawling documents from 30 well-known Websites of different contents and capacities. A list of the Uniform Resource Locators (URLs) of these Websites is given in Table 2.

Table 2. List of initially visited Websites

#	Websites	#	Websites
1	www.aljazeera.net	16	www.en.wna-news.com
2	www.bbc.co.uk	17	www.visitabudhabi.ae
3	www.tradearabia.com	18	www.interactive.aljazeera.net
4	www.krg.com	19	www.bbcworld.mh.bbc.co.uk
5	www.iranpressnews.com	20	www.bahrainnews.com
6	www.bbc.co.uk/blogs	21	www.khilafah.com
7	www.labs.aljazeera.net	22	www.alarabiya.net
8	www.terrorism-info.org.il	23	www.blogs.aljazeera.net
9	www.live.bbc.co.uk	24	www.ameinfo.com
10	www.alarabiya.net/index/videos/default	25	www.dubai.ae
11	www.alsumaria.tv	26	www.dubaicityguide.com
12	www.pukmedia.co	27	www.dmi.ae/dubaitv
13	www.alhurriatv.com	28	www.adtv.ae
14	www.tacme.com	29	www.iraq.net
15	www.azzaman.com	30	www.efa.com.eg

Furthermore, for fare evaluation and comparison, the crawling process is perform to retrieve various number of documents ranged from 10000 to 70000 documents. The results which follow are presented with the aim of investigating the variation of S against the number of VMs, and the number of crawled documents.

7.1. NOVCrawler Performance

The crawling process is carries-out by running NOVCrawler, on the computer described in Table 1, to estimate T_s for crawling different pre-specified number of Web documents (W_o) ranging from 10000 to 70000 in step of 10000. The crawling process starts with 30 URLs given in Table 2, all will be fetched sequentially (one-by-one). For each fetched URL, a number of Web documents will be retrieved. Since each Web document may contain a number of URLs. These URLs are extracted, parsed and added to CrawlDB, i.e. updating CrawlDB. NOVCrawler keeps records of both the actual number of crawled documents (W_a) and the actual number of fetched URLs (U_a).

The crawling process is designed to continue until W_o documents are fetched (regardless of U_a). Since, the number of documents in the any fetched URL cannot be predicted; in practice, the crawling process is terminated when W_a reaches any value just above W_o (i.e., $W_a \geq W_o$). NOVCrawler is also developed to estimate the CPU time required to crawl the last 1000 documents (T_l). Furthermore, the average CPU time spent to crawling 1000 documents (T_{avg}) is calculated as $T_{avg}=1000\times t_{avg}=(1000\times T_s)/W_a$. The values of T_s, T_l, T_{avg}, and R_{avg} are shown in Figures 7 to 10, respectively.

7.2. DVMCrawler Performance

The crawling process is carries-out using DVMCrawler on the same computer described in Table 2 with different number of VMs installed on, to estimate T_d for crawling different pre-specified number of Web documents (W_o) ranging from 10000 to 70000 in step of 10000. In particular, two different VM-based distributed crawling systems are configured. The first system encompasses 3 VMs, while the second system encompasses 4 VMs. The VMs are configured in master-slave architecture, where one of the VMs is run as a master, while the other machines are run as slaves.

The crawling process starts with the same list of URLs given in Table 2, which will be fetched concurrently using the installed VMs, each VM starts with 30/(n-1) URLs. However, as we discussed above that the crawling process is terminated after we reach a number of documents just above W_o, which we referred to as W_a. The values of T_d,T_l,T_{avg}, andR_{avg} for various crawled documents (W_a) for 3 and 4 VMs, are shown in Figures 7 to 10, respectively.

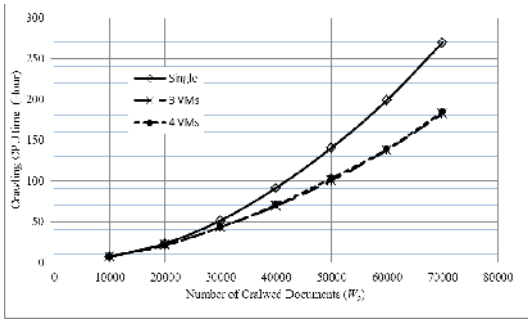


Figure 7. Variation of T_s and T_d against W_o.

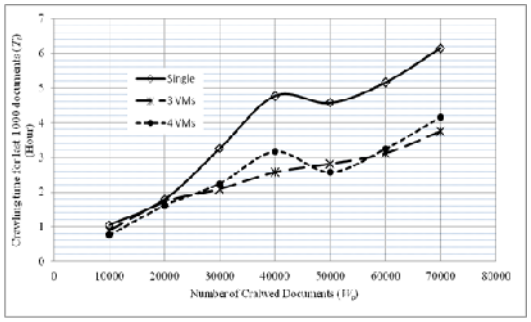


Figure 8. Variation of T_l against W_o.

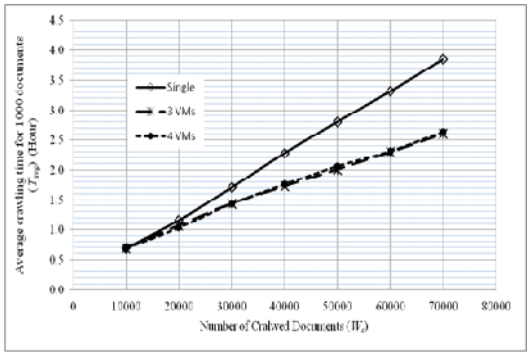


Figure 9. Variation T_{avg} against W_o.

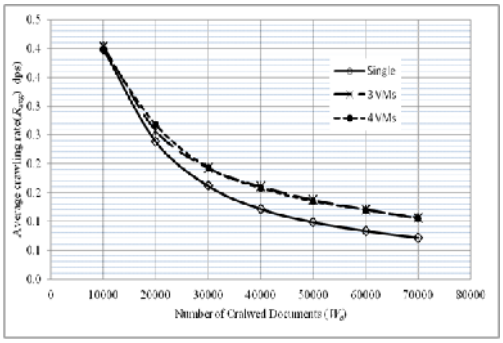


Figure 10. Variation of R_{avg} against W_o.

To perform a fare comparison, S must be calculated as a ratio between the crawling times T_s and T_d require to crawling exactly the same number of documents. Therefore, the crawling times T_s and T_d are approximated to represent the crawling time of the equivalent base W_o documents. The approximated time is calculated as $(W_o \times T_{s/d})/W_a$. The estimated S factors for the various numbers of documents are given in Figure 11.

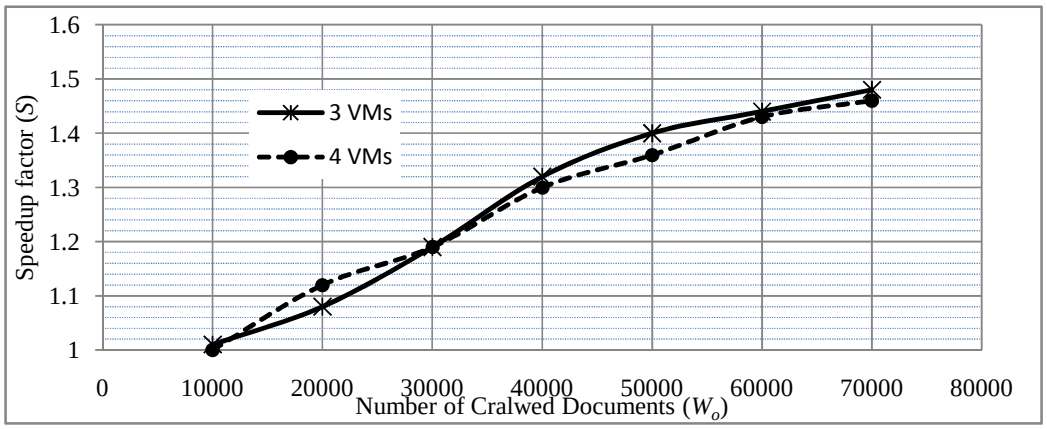


Figure 11. Variation of S against W_o.

It can be seen in Figure 7 that the crawling time T_d for the VM-based configurations is less than T_s for the no VM-based configuration. Furthermore, the difference between T_s and T_d increases as W_o increases. This demonstrates the potential of the concept of this work which provides better processor utilization through virtualization. However, the crawling time T_d for 3 and 4 VMs are almost the same. This can be due to two main reasons:

1. As the number of VMs increases, the hardware resources allocated to each VM is reduced, because the hardware resources of the system processor are distributed between more VMs.
2. For the same value of W_o, when the number of VMs increases, the amount of computation (CPU crawling time) assigns to each VM is reduced, and, on the other hand, the amount of communication (communication time) is increased. As a result of that the total crawling time is almost unchanged.

Figure 11 shows the speedup factor *S* achieved by the VM-based crawling system, which almost increases with increasing W_o. The maximum (ideal) value for *S* is equivalent to the number of VMs. However, we expect to reach a steady state (saturated value) below the ideal value due to communication overhead. It is can be easily seen in Figures 8 to 10 that the VM-based distributed crawling configurations require less crawling time for the last 1000 documents and less average crawling time per 1000 documents, and at the same time provides higher crawling rate. Once again, the results for 3 and 4 VMs are almost the same due to the reasons discussed above.

8. CONCLUSIONS

The main conclusions of this work can be summarized as follows:

1. For equivalent crawling task, a VM-based Web crawling system performs faster than the same system with no VMs installed on, i.e., reduces crawling CPU time and increases processor utilization.
2. The speedup factor, and the average crawling rate increase as the number of crawled documents increases. For example, Figure 7 shows that *S* achieved by DVMCrawler running on 3 VMs, increases from 1.01 for 10000 documents to 1.48 for 70000 documents. This demonstrates that for 70000 documents, the crawling CPU time is reduced by nearly 32%. This expects to further increase as the number of targeted crawled documents increases.
3. A VM-based system crawling system shows smaller growth in the crawling CPU time as the data grows in size as compared to using the same system as a single processor (no virtualization).
4. Our investigations on 3 and 4 VMs demonstrate that for this limited range of crawling computations (crawl a maximum of 70000 documents); the number of VMs has insignificant effect on *S* and R_{avg}, which is discussed in Section 7.
5. Creation of VMs adds an isolation layer between those VMs and the real processor hardware, which is considered as an overhead. However, when dealing with big data this overhead layer hides the OS limitations and contributes to a better utilization to the processor resources (computing powers and memory).
6. The high-performance multi-core processors with VMs installed on can play a big role in developing cost-effective Web crawlers or can be used to enhance the performance of current Web crawlers at Google, Yahoo, and other Web search engines.

Due to resources cost and time limit, in this paper, only relatively small numbers of URLs will be crawled to prove the research concept, therefore, there are a number of recommendations that can be suggested as future work; these recommendations may include:

1. Modify the distributed tool (DVMCrawler) as to assign crawling task to the master VM instead of being idle while waiting other slave VMs completing their crawling tasks, and then evaluate the performance of the modified tool and estimate the performance (speedup, efficiency, average crawling rate) enhancement.
   a. Extend the investigation and the performance evaluation for higher number of targeted crawled documents (W_o>70000), and higher number of VMs installed on the same multi-core processor (VMs>4).
2. Investigate the effect of the number of initial URLs on the performance of the VM-based crawling system.
3. Evaluate the performance of the distributed model on various multi-core processors of different specifications (speed and memory).

REFERENCES

[1] B. Croft, D. Metzler, &T. Strohman (2009). Search Engines: Information Retrieval in Practice. Addison Wesley.

[2] M. Levene (2005). An Introduction to Search Engine and Web Navigation. Addison Wesley.

[3] WorldWideWebSize (2012). The size of the World Wide Web. http://www.worldwidewebsize.com.

[4] T. Calishain. (2004). Web Search Garage. Prentice Hall.

[5] C. Olston&M. Najork (2010). Web Crawling. Now Publishers Inc.

[6] Q. Tan (2009). Designing New Crawling and Indexing Techniques for Web Search Engines. VDM Verlag.

[7] W. Gao, H. C. Lee, &Y. Miao (2006). Geographically Focused Collaborative Crawling. Proceedings of the 15th International Conference on World Wide Web (WWW'06), pp. 287-296.

[8] D. Mukhopadhyay, S. Mukherjee, S.Ghosh, S. Kar, &Y. C. Kim(2006). Architecture of a Scalable Dynamic Parallel WebCrawler with High Speed Downloadable Capability for a Web Search Engine. Proceedings of the 6th International Workshop on Multimedia Signal Processing & Transmission (MSPT'06). pp. 103-108. Jeonju, Korea.

[9] S. Tongchim, P.Srichaivattana, C.Kruengkrai, V.Sornlertlamvanich, &H. Isahara(2006).Collaborative Web Crawler over High-speed Research Network. Proceedings of the 1st International Conference on Knowledge, Information and Creativity Support Systems, pp. 302-308. August 1-4. Ayutthaya, Thailand.

[10] Boldi, P., Codenotti, B., Santini, M., & Vigna, S. (2004). UbiCrawler: A Scalable Fully Distributed Web Crawler. Software: Practice and Experience, Vol. 34, Issue 8, pp. 711 – 726.

[11] J. Cho,&H. Garcia-Molina (2002). Parallel Crawlers. Proceedings of the 11th International Conference on World Wide Web (WWW'02). May 7-11. Honolulu, Hawaii, USA.

[12] V. Shkapenyuk&T. Suel (2002).Design and Implementation of a High-Performance Distributed Web Crawler.Proceedings of the 18th International Conference on Data Engineering, pp.357-368. February 26-March 1. San Jose, CA, USA.

[13] I. Hernandez, C. R.Rivero, D. Ruiz, &R. Corchuelo (2012) An Architecture for Efficient Web Crawling. Lecture Notes in Business Information Processing, Vol. 112, pp. 228-234

[14] S. Ghoshal (2011). Computer Architecture and Organization. Pearson Education.

[15] J. Laudon L.Spracklen (2007). The ComingWave of Multithreaded Chip Microprocessors. International Journal of Parallel Programming, Vol. 35, No. 3, pp. 299-330.

[16] T. L. Floyd (2009). Digital Fundamentals. 10th Edition. Prentice Hall.

[17] H. Al-Bahadili&S. Al-Saab (2011).Development of a Novel Compressed Index-Query Web Search Engine Model. International Journal of Information Technology and Web Engineering (IJITWE), Vol. 6, No. 3, pp. 39-56.

[18] J. Smith &R. Nair (2005). Virtual Machines: Versatile Platforms for Systems and Processes. The Morgan Kaufmann Series in Computer Architecture and Design.

[19] N. Smyth (2010). Xen Virtualization Essentials. Payload Media.

[20] C. Takemura &L. Crawford (2009). The Book of Xen: A Practical Guide for the System Administrator. 1st Edition. No Starch Press.

[21] J. Cho, H. Garcia-Molina &L. Page (1998). Efficient Crawling Through URL Ordering.Proceedings of the 7th International World-Wide Web Conference (WWW 1998), April 14-18, Brisbane, Australia.

[22] S. Chakrabarti, M. van den Berg, &B. Dom (1999). Focused Crawling: A New Approach to Topic-Specific Web Resource Discovery. Computer Networks, Vol 31, Issue 11-16, pp. 1623–1640.

[23] C. Chung &C. L. A. Clarke (2002). Topic-Oriented Collaborative Crawling. Proceedings of the ACM Conference on Information and Knowledge Management (CIKM'02), November 4–9. McLean, Virginia, USA.

[24] H. Yan, J. Wang, X. Li, & L. Guo (2002). Architectural Design and Evaluation of an Efficient Web-crawling System. Journal of System and software, Vol. 60, Issue 3, pp.185-193.
[25] T. Takahashi, H. Soonsang, K. Taura, & A. Yonezawa (2002). World Wide Web Crawler. Proceedings of the 11th International Conference on World Wide Web (WWW 2002), May 7-11, Honolulu, Hawaii, USA.
[26] B. T. Loo, O. Cooper, & S. Krishnamurthy(2004). Distributed Web Crawling over DHTs. University of California, Berkeley. EECS Technical Report Identifier: CSD-04-1305. Retrieved on 12 August 2012 fromhttp://techreports.lib.berkeley.edu/accessPages/CSD-04-1305.
[27] Y. Hafri&C. Djeraba(2004). High Performance Crawling System. Proceedings of the 6th ACM SIGMM International Workshop on Multimedia Information Retrieval (MIR'04), pp. 299-306, October 10-16, New York, USA.
[28] J. Exposto, J. Macedo, A. Pina, A. Alves, &J. Rufino (2005). Geographical Partition for Distributed Web Crawling. Proceedings of the 2nd International ACM Workshop on Geographic Information Retrieval (GIR 2005), ACM Press, pp. 55-60, Bremen, Germany.
[29] D. H. P. Chau, S. Pandit, S. Wang, &C. Faloutsos (2007). Parallel Crawling for Online Social Networks. Proceedings of the International Conference on World Wide Web (WWW 2007). May 8-12. PP. 201-210. Banff, Alberta, Canada.
[30] S. Ibrahim, H. Jin, L. Lu, L. Qi, S. Wu, &X. Shi (2009). Evaluating MapReduce on Virtual Machines: The Hadoop Case. Lecture Notes in Computer Science, Vol. 5931, pp. 519-528.
[31] J. M. Hsieh, S. D. Gribble, &H. M. Levy (2010). The Architecture and Implementation of an Extensible Web Crawler. Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation (NSDI '10). April 28-30. San Jose, California, USA.
[32]S. Anbukodi&K. M. Manickam (2011). Reducing Web Crawler Overhead Using Mobile Crawler. International Conference on Emerging Trends in Electrical and Computer Technology (ICETECT 2011). pp. 926-932. March 23-24. Nagercoil, India.
[33] Xen (2012). http://www.xen.org.
[34] F. Hartl (2012). Nutch How It Works. http://www.florianhartl.com/nutch-how-it-works.html.
[35] T. White (2012). Hadoop: The Definitive guide. 3rd Edition. O'Reilly Media.
[36] H. M. Qtishat (2012). Developing a Virtual-Machine-Based Distributed Web Crawling Model. MSc Thesis. Faculty of Information Technology, Middle East University, Amman, Jordan.
[37] M. Fleury&A. Downton (2004). Pipelined Processor Farms: Structured Design for Embedded Parallel Systems. John Wiley & Sons.
[38] S. Roosta (2000). Parallel Processing and Parallel Algorithms: Theory and computation. Springer Verlag. New York, USA.

Authors

Hussein Al-Bahadili (hbahadili@uop.edu.jo) is an associate professor at Petra University. He received his PhD and M.Sc degrees from University of London (Queen Mary College) in 1991 and 1988. He received his B.Sc in Engineering from the University of Baghdad in 1986. He has published many papers in different fields of science and engineering in numerous leading scholarly and practitioner journals, and presented at leading world-level scholarly conferences. His research interests include computer networks design and architecture, routing protocols optimizations, parallel and distributed computing, cryptography and network security, data compression, software and Web engineering.

HamzaAlqtaishat (hamzah.qtishat@yahoo.com) is a senior developer at Yahoo Arabia, Amman, Jordan. He received his B.Sc degree from Al-Balqa Applied University, Jordan in 2006 and his M.Sc degree in Computer Information Systems from the Middle-East University, Amman, Jordan in 2012 under the supervision of Dr. Hussein Al-Bahadili. His current research interests are Web Search Engine Architecture and techniques, Virtual Machines, Parallel and Distributed Processing and Methodologies, Websites Development, Web Services, Network Scalability, and Social Networks.

ReyadhS. Naoum (rnaoum@Meu.edu.jo) is a professor at Middle East University, Amman, Jordan. He received his PhD and M.Sc degree from University of Manchester in 1976 and 1973.He received his B.Sc degree in Mathematics from University of Basra in 1969. He published more than one hundred papers in the field of Computer Science and Mathematicshigh standard journals, and well-recognized international conferences. His current interest includes Neural Computing and Genetic Algorithms, Image Processing, Cloud Computing, Computer Security and Instruction Detection Systems, Numerical Analysis Computation and Software.