12-2013

A Bandwidth-Conserving Architecture for Crawling Virtual Worlds

Dipesh Gautam
University of Arkansas, Fayetteville

A Bandwidth-Conserving Architecture for Crawling Virtual Worlds

A Bandwidth-Conserving Architecture for Crawling Virtual Worlds

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science

by

Dipesh Gautam
Tribhuvan University
Bachelor of Engineering in Computer Engineering, 2005
Chosun University
Master of Science in Computer Engineering, 2008

December 2013
University of Arkansas

This thesis is approved for recommendation to the Graduate Council.

_____
Dr. Susan Gauch
Thesis Director

_____
Dr. Craig W. Thompson
Committee Member

_____
Dr. Gordon Beavers
Committee Member

ABSTRACT

A virtual world is a computer-based simulated environment intended for its users to inhabit via avatars. Content in virtual worlds such as Second Life or OpenSimulator is increasingly presented using three-dimensional (3D) dynamic presentation technologies that challenge traditional search technologies. As 3D environments become both more prevalent and more fragmented, the need for a data crawler and distributed search service will continue to grow. By increasing the visibility of content across virtual world servers in order to better collect and integrate the 3D data we can also improve the crawling and searching efficiency and accuracy by avoiding crawling unchanged regions or downloading unmodified objects that already exist in our collection. This will help to save bandwidth resources and Internet traffic during the content collection and indexing and, for a fixed amount of bandwidth, maximize the freshness of the collection. This work presents a new services paradigm for virtual world crawler interaction that is co-operative and exploits information about 3D objects in the virtual world. Our approach supports analyzing redundant information crawled from virtual worlds in order to decrease the amount of data collected by crawlers, keep search engine collections up to date, and provide an efficient mechanism for collecting and searching information from multiple virtual worlds. Experimental results with data crawled from Second Life servers demonstrate that our approach provides the ability to save crawling bandwidth consumption, to explore more hidden objects and new regions to be crawled that facilitate the search service in virtual worlds.

ACKNOWLEDGEMENTS

I express my gratitude to my advisor Dr. Susan Gauch for her precious support and guidance during my study. Her guidance and valuable suggestions shaped this thesis to this form.

I also thank my committee members Dr. Craig W. Thompson and Dr. Gordon Beavers for suggesting ways to better organize the thesis and for guiding me on which topics to emphasize.

In addition, I thank my wife Tara for her support and encouragement throughout my career. Also my appreciation goes to my son Sauvagya for finally growing healthier throughout the period of writing this thesis. Last but not least, I thank my parents and all the family members for their support and love throughout my life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Search engines revolutionized the way we discover information in the World Wide Web. It is likely that search engines could also vastly improve how we discover and use information available in 3D virtual worlds. Virtual world crawlers are programs that automatically collect object data from virtual world servers. They differ from WWW crawlers in that much of the information is visual rather than textual. Of the text content that users do see, most is embedded in texture images, note cards, and chat messages that are triggered by avatar proximity or actions (see Figure 1).

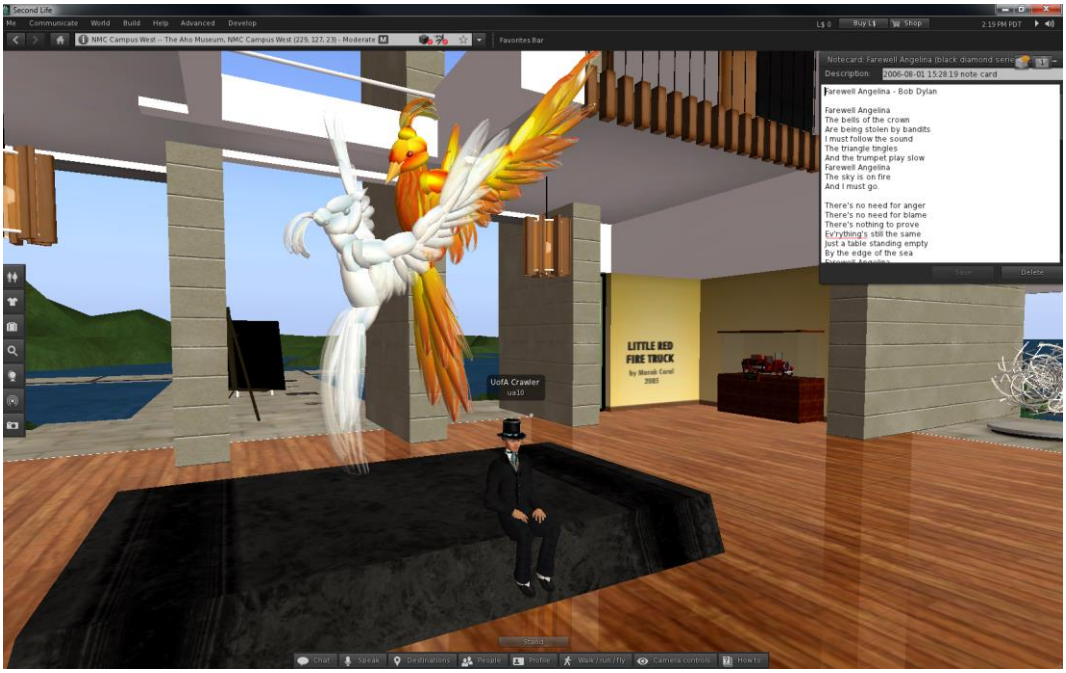

Figure 1: Content Sources in Virtual Worlds

Virtual worlds present problems in content collection that differ significantly from traditional crawlers, making this an open field for further research. Rather than simply queuing up the list of URLs to visit, a crawler needs to plan the order of location visits, deal with access permissions, perform route planning, and avoid obstacles en route. In addition, it is more difficult

to collect object content from a virtual world because most of the objects in the virtual world are labeled with tags that bear little or no semantic information. Therefore, crawling data in three-dimensional (3D) environments has additional constraints that may affect how much content can reasonably be collected. A crawler must explore an unknown search space, convert formatted content into a searchable form, and coordinate the search between multiple crawler processes. The primary difference in comparison with the crawling flat web is how a crawler navigates a space. One of the main challenges for a virtual world crawler is that content is only presented to crawlers as they move close to the content location. Therefore, a crawler must dynamically model the environment to find a path to maneuver close to any content it collects. To collect such content, the crawler must interact with the world the same way an avatar would.

Since the virtual world crawler has to 'touch' the object before collecting its information, the crawler must move around and approach objects in the region it is exploring. As it travels throughout a region, the crawler may encounter objects have already been collected by previous crawls. Thus, the crawler will consume bandwidth needlessly by collecting these redundant objects. Another problem in 3D crawling is the obstacle avoidance and banning of the avatars if the crawler tried to enter some restricted region. In this case, several dynamic objects with valuable information might be missed.

1.1 Motivation

Virtual worlds can be navigated by users who create human-emulating avatars that represent them. Most users discover content and areas of interest just by wandering around and/or getting recommendations from friends. The virtual world servers do provide some search functionality; however, it is limited in the scope of the type of search supported. Unlike the

WWW, one important drawback that provides a poor user experience is that the virtual worlds are not interlinked together. It is similar to being able to search for content at a single web site rather than using a search engine that can search for information no matter where it is hosted. In order to provide cross-site searching for virtual worlds, we need to create a crawler that can collect information from the worlds to a central location, index the content, and provide the users with efficient and effective search support.

Virtual worlds such as Second Life and OpenSim can also be navigated by crawlers that mimic the behavior of human users [4]. The contents of the virtual worlds range from text to multimedia to graphical objects. These contents are not visible to avatars other than by approaching them to within a given proximity in the regions where the contents are located. Because of the distribution of the content around a 3D virtual world, and the requirement that the crawler virtually navigate around the world(s) to collect content, virtual world crawlers are much more difficult to create than WWW crawlers, and they are also much less efficient.

In this work, we present an approach that investigates the efficiency gains possible when the virtual world crawler and the virtual world servers work collaboratively to support content collection. This scheme would also make it possible to efficiently create a virtual world search engine capable of collecting information from multiple virtual worlds and helping users find locations of interest across multiple sites.

1.1.1 Crawling in Virtual Worlds

In order to extract the information implicit in virtual worlds, the virtual worlds need to be crawled and the data converted into a searchable format. However, crawling in virtual worlds needs extra overhead in terms of bandwidth and the process to collect the data. Unlike the

WWW, the crawler actually has to log in to the server and wander around the region to discover the objects present there. So the crawler has to plan a path through the region. Moreover, the content in the virtual worlds is graphical, and we need to bridge the semantic gap between the textual contents and the graphical objects in order to support users querying for locations based on textual queries.

1.1.2 Redundant Data

Users want to retrieve results that are up-to-date. They want access to recently added content and they do not want to click on a result only to find that that item no longer exists. Thus, the freshness of the data in the search engine's collection is of great importance. In order to maintain freshness, crawlers have to revisit locations to collect information periodically. The more often they visit each region, the more up to date the information about the contents of that region. However, the majority of the contents the crawler discovers during subsequent crawls are stale, i.e., they were previously collected and the content is neither new nor updated. This collection of stale data imposes an unnecessary burden on the server and the crawler. The redundant data collected by virtual world crawler has much severe impact in comparison to the WWW crawlers as the former mostly has to collect the multimedia and graphical content. Our work focuses on exploring a method to avoid redundant data collection.

1.1.3 Bandwidth Saving Issues

Web site publishers want their pages to be easily collected by search engine crawlers. However, the Web server has to accommodate the bandwidth consumed imposed by the crawlers without compromising the performance provided to visitors to the Web site. To address this,

many large web sites maintain separate servers or RSS feeds to support crawlers, separating their traffic from human users. More novel approaches propose using web logs to keep track of changes in the sites' web pages. The crawlers can quickly discover the fresh pages or the updated pages from those logs. Virtual world servers also have to handle a huge amount of traffic when crawlers harvest their data. By avoiding redundant data collection, and developing a collaborative architecture, our work focuses on techniques to reduce bandwidth consumption by virtual world crawlers.

1.2 Goals

Our goal in this work is to explore a more efficient and exhaustive method of collecting content from virtual worlds. In this thesis, we investigate the potential bandwidth savings that a collaborative crawling approach could achieve. In addition, this collaborative approach could be used to direct the crawler to a list of unvisited regions or a region in the virtual world or areas that have a high rate of change. To do so, we have developed a focused crawler allowing us to collect data from the Second Life and/or OpenSimuator virtual worlds. Once we gather data, we explore how frequently content changes in different regions to build a model of the rate of change in virtual worlds. We use that model to propose an architecture that could allow the crawler better collect new or unvisited objects in a virtual world. In addition, we are also interested in how bandwidth traffic can be improved for the crawling process if we can well manage the data redundancy after each download. Our empirical experiments on the data collected from Second Life servers shows the effectiveness of our approach in terms of supporting distributed search in virtual worlds.

1.3 Approach

In this study we crawled regions hosted on a Second Life server to collect the data. We developed a crawler that mimics normal user behavior in Second Life. The crawler could be directed to crawl a list of selected regions.

We crawled several regions of Second Life over regular interval of times and collected snapshots of the data in these regions at each visit. Because they make up the vast majority of the content and are also the most expensive to collect, we focused our analysis on static objects in Second Life rather than other forms of dynamic content such as notecards, landmarks and avatars. After collecting the data from the collected snapshots, we measured the redundancy and estimated the portion of average bandwidth consumed by collecting redundant data.

1.4 Organization of the Thesis

The thesis is organized as follows: In Chapter 2 we discuss background information about virtual worlds, virtual worlds search technology, virtual worlds research, and introduce crawlers. In Chapter 3, we discuss our approach in detail and propose our novel system architecture. Chapter 4 presents our experiments and finally we conclude our thesis in Chapter 5.

2. BACKGROUND

This section describes related work on crawling data in virtual world and efforts to support crawlers in exploring objects in 3D environment.

2.1 Virtual World Search Technology

Among current virtual worlds, Second Life[1] and OpenSimulator (OpenSim)[2] are the two most active worlds that have most subscribers. There are two existing search services, one relies on the internal content database and the other relies on a combination of avatar crawlers and database access. The official Second Life search relies on the internal content database and does not extend to searching in virtual worlds hosted in other servers.

Second Life provides a search interface for internal search that supports search for several types of entities based on different filters: Everything, Classifieds, Events, Destination Guide, Groups, Land & Rentals, People, and Places. These filters are supposed to provide relevant search results; however, the search interface does not support Boolean search, search based on partial keywords, or search for objects by type, location shape, color, texture and other physical attributes. Since Second Life is, in effect, a single large virtual world, their search allows search for objects only within that one site.

The OpenSimluator's search also relies on internal database search, however since OpenSimulator is open source and stores data about the virtual worlds in an accessible relational database, the virtual world provider can chose which data to be public so that the search content is less restricted in comparison to to Second Life. The OpenSimulator exposes the internal

[1] http://secondlife.com
[2] http://opensimulator.org/wiki/Main_Page

content to external API through DataSnapshot[3] module and OpenSimSearch (OSS)[4]. The OpenSimulator worlds that are connected in a grid can be searched through the grid database. However this is not true if one has to search individual non-connected standalone simulator or the grid with a single query. Moreover, since the Second Life grid and OpenSimulator grids are not connected together, searching in both grids with a single search query is not common at present. Similar to modern search engines, a typical search service in each world of OpenSimulator is hosted on a separate server.

There have been several works to develop search engine that can retrieve data from several non-connected virtual worlds. One of such attempts is done by Metaverse Inc[5] whose search service is based on crawlers in Second Life, web access to the content of Second Life and DataSnapShot module of the OpenSimulator. Though the project is still in the stage of development, its search service provides keywords search, phrase search, Boolean search and image search to some extent.

2.2 Virtual Worlds Research

Relatively little research has explored open questions in virtual worlds; of the several research projects, few focus on crawling and mining of the data. A more popular area of research focuses on the behavioral aspects of the users and others focus on interoperability among the several virtual worlds. In the following subsection we discuss representative research projects in each of these areas.

---

[3] http://opensimulator.org/wiki/OpenSim.Region.DataSnapshot
[4] http://opensimulator.org/wiki/OpenSimSearch
[5] http://metaverseink.com/

2.2.1 Interoperation

Currently, interoperation among several virtual world environments remains a major challenge. The lack of interoperation on several current and possibly the future virtual worlds is one of the main constraints in the widespread growth of the virtual worlds. Several researchers have been working on how to mitigate the constraints. Bell et al. [1] introduced the VWRAP (Virtual World Region Agent Protocol) addressing the problem of interoperability for a family of current and future virtual worlds. This protocol is an application layer protocol aimed at supporting interaction between virtual environments over which different virtual world service providers can deploy their services. The VWRAP work group defined various components such as authentication between client and agent services, a protocol for placing users and their movements within a region, communication between virtual world users, content format, and deployment patterns. The protocol intends to leverage existing standards such as HTTP in order to take advantage of existing code libraries. Other work by Lopes [11] proposed a Hypergrid[6] architecture and protocol for decentralizing multiuser virtual environments in which multiuser applications can exchange user agents and assets. They implemented a Hypergrid architecture in OpenSimulator which has been adopted by a number of independent virtual world providers. Their architecture enables interoperation between different virtual worlds transparently such that an avatar can be teleported from one world to another. During teleportation to other world, the user agent has to send the user's private information to the other server so they described a separate login protocol for securely transferring the user agents. Although these and other interoperability research project have not been adopted by current virtual world servers, they

[6] http://opensimulator.org/wiki/Hypergrid

shed a light on the future in which multiple virtual worlds interoperate, providing the user with seamless experience much in the way the users can link from one web site to another and browse the web seamlessly.

2.2.2 Behavioral Aspects

Some research on virtual worlds focuses on the avatars rather than the environment. Zhang et al. [16, 17] studied avatar action differences between the genders as well as between younger and older avatars. For their study, they developed an integrated framework that combines bot-and spider–based approaches to collect avatar behavioral and profile data quickly and efficiently from three different regions. Their bots are avatars with scripts written in LibOpenMetaverse[7] and are used to collect behavioral data of the avatars. The spidering is done to collect profile information such as birth date, profile picture and self short introduction. The spider operates by sending the name of the avatar that was collected by bot. As the profile page does not include gender, the researchers manually entered the gender of the avatars. They characterize the avatar behavior as physical behavior such as flying, running, walking, turning right or left and standing. Their results revealed that avatars behave similarly to people in the real world. Indeed, male avatars are more physically active than their female counterparts and young avatars are more active than older avatars.

Yee et al. focused their research on exploring how users interact within virtual worlds. They created a framework to collect avatar's spatial coordinates, change of outfits, or chatting activity using Linden Scripting Language (LSL) [15]. Their framework could be used by other researchers to capture avatar-related data from Second Life every one minute or less over a

[7] http://openmetaverse.org/projects/libopenmetaverse

period of a week. In other work by La et al. [10], they collected users' spatial data to study user mobility in Second Life. They used a virtual sensor-based approach and compared that to information collected using a crawler. Their virtual sensor-based approach faced several hurdles because of restrictions on the number of messages that could be retrieved from the sensor. To overcome this, they used a crawler that mimicked normal user behavior. Although their crawler could monitor spatial information about avatars in a public region, they initially faced unwanted attraction of avatars towards their crawler. They repeated their experiment by randomly moving around the land and broadcasting chat messages so as to give the impression of a normal user to other avatars in the region. Since they were only concerned with the spatial distribution of avatars, they merely moved their crawler within confined distance and collected data at interval 10 second intervals for 24 hours. From their experiment, they found that the avatars generally concentrate around points of interest and travel relatively short distances.

2.2.3 Server Architecture

Varvello et al. in [13] developed a peer-to-peer (P2P) client for delivering virtual worlds' information and compared it to the centralized client-server architecture currently used by Second Life. They compared their approach to Second Life's client server architecture by measuring the "Quality of Experience" as the fraction of inconsistent avatar information collected and the time needed to collect consistent information (inconsistency duration) in an area of interest. They compared the performance of their P2P architecture with Second Life's client server architecture for receiving and disseminating the avatar's state updates. Their experiment indicated that the P2P architecture outperformed the client server architecture in terms of "Quality of Experience" perceived by multiple Second Life users.

In their more recent work bye Varvello et al. [12], the authors collected object from several thousand regions, collected server statistics of public regions and tracked avatar distribution in the entire virtual world over one week and found that 30% regions were never visited during six days, fewer than 1% of regions had large peak populations, and the majority of the regions were static. They concluded that support for stale regions wastes precious resources at a time of increasing for bandwidth-heavy features such as more and more active avatars and higher quality streaming information [8, 9].

In other work similar to ours, Varvello et al. [14] analyzed Second Life's scalability, popularity, staleness of the objects and the quality of the user experience based on the number of objects in various regions at different points of time. However, staleness may not be accurately assessed merely by counting the number of objects. In our work, we also compared the objects themselves over time to take into account the number of edited objects at different points of time during our crawling process. By also comparing the identities of the objects, not just their raw counts, we were able to track created, deleted, and modified objects.

2.3 Crawlers

Crawlers are programs that navigate through virtual space following links or visiting a pre-set list of locations, collecting information and storing the collected data at a central location for indexing and processing of user queries. At present, leading virtual worlds such as Second Life are hosted on single domain that has access to the database of all objects, so their search service does not rely on crawlers to collect information.

However, virtual worlds are growing quickly and several enterprises are hosting 3D-worlds in their own simulators that can be navigated through any client program that can navigate Second

Life. Also IBM and Linden lab have been collaborating to work on Open Grid Protocol to allow interoperability between virtual worlds. In 2008, they announced that they could successfully teleported avatars from second life to OpenSimulator grids. This success opened the door for interoperability of virtual worlds. Moreover OpenSimulator allows developers to customize the code so several services in virtual worlds do not have to rely on boundary of Second Life protocol. So these trends potentially lead future virtual worlds to interoperate and there will be a need to support crawlers for cross-site search services.

2.3.1 WWW Crawlers

Web crawlers, sometimes called robots or spiders, aim to collect exhaustive, fresh content from the World Wide Web (WWW) while minimizing bandwidth utilization. Matthew Gray of MIT[8] introduced the first crawler, Wanderer, in 1993. Wanderer was written in Perl and its task was to discover new websites. WebCrawler[9], written by Brian Pinkerton in 1994 was the first search engine to provide full text search for web pages. This was the first step to creating a search engine that could index contents from multiple web sites, process keyword-based queries, and direct users to pages hosted anywhere on the web. His crawler started with a few seed URLs, mimicked a user requesting those web pages, and downloaded a copy of each page. Each page was then parsed to extract the links to add to the list of URLs to visit and the process continued until all links were processed. WebCrawler's first index had pages from 6000 sites[18]. To keep the search index size small, many early search engines only indexed the first 1000 characters of each web page rather than the entire contents. They also ignored images and

[8] http://www.mit.edu/~mkgray/net/background.html
[9] http://www.webcrawler.com/

other embedded multimedia content. In comparison, due to the explosive growth of the web, Google's collection now contains over 30 trillion pages and it is constantly growing.

One issue faced by Web crawlers is the need to revisit sites to keep the index up to date. Since the web has grown so large, crawlers must strike a balance between visiting newly discovered urls and revisiting existing urls to keep their collections fresh. Revising web sites over and over again to update the search engine's collection is very expensive in terms of bandwidth. Also, because many web pages do not change between visits, much of this recollection is unnecessary. Cho et al. [21] found that more than 20% of pages change in a day and also about 70% of the pages did not change over 4 months period.

Due to the enormous number of web pages, revisiting all the pages frequently is impossible, making it a major task to collect fresh information. The content of web pages changes so frequently that the crawler may miss the updates from a particular page between two crawls or it may never get chance to crawl a new page because it has to recrawl the previously crawled pages after some interval. Cho et al. [19] proposed an approach to priority-based URL collection by using matrices such as Similarity of page with query, Backlink Count, PageRank, Forward Link Count and Location Matric. Their experiment revealed that the PageRank metric discovered important pages more quickly than other metrics. Najork et al. [20] extended Cho et al.'s approach [19] by crawling somewhat more pages and demonstrated that the important pages could be discovered earlier by crawling with a breadth first search order supplemented with PageRank as a metric for important pages.

The PageRank approach and the Breadth First Search approaches can both negatively impact a user's experience. PageRank penalizes new pages whereas important pages located deeper in hierarchy may never be collected by Breadth First Search.

Several researchers have explored new techniques have the web server support crawlers by providing metadata about web pages. Bradman et al. [22] proposed a scheme in which the web server provides a list of URLs and their meta-data containing the information about modified pages and dynamic pages and the crawler can download that file to plan the crawling. Gupta et al. [23] proposed an algorithm to send updates on popular pages from the web server to search engines. Buzzi [2] proposed the creation of a text file to implement a web service that publishes information such as creation and update dates, file size, request frequency for each object of a website. Google introduced Sitemaps[10] in 2005 to support the web crawler. A Sitemap is an XML file of URLS of a website. In Google Sitemap, webmasters can create text or XML file of Sitemap by using third party software. Google has reported that introduction of Sitemap provides faster discovery of important pages and improved coverage of the website.

Chandramouli et al. [3, 24] designed a collaborative architecture in which web servers combine information from web logs and the file system to keep track of page creation, deletion, and modification. This information was available to Web crawlers via a web service. They evaluated bandwidth savings over eight weeks and compared the results with Traditional Crawling (TC) techniques to their different techniques such as File System (FS), Web Logs (WL), Merge Hybrid (MH), File system first Hybrid (FH) and SiteMap (SM). Their result showed nearly 100% improvements in bandwidth savings by all the collaborative approaches except SiteMap which was only over 74%. Their result clearly indicated that the collaborative architecture using Web logs and File system tremendously reduced bandwidth consumption by the crawler. In their more recent work [25], they also found that that exploiting the web log

[10] https://www.google.com/webmasters/sitemaps/login

information could be used to improve the accuracy of the search results for users by ranking heavily visited web pages more highly.

In this thesis we focus on calculating the bandwidth required when regions are repeatedly crawled completely, including the collection of stale objects, and contrast that with the bandwidth needed by a collaborative architecture. Due to its effectiveness in decreasing crawler bandwidth, our approach is modeled on [3], a web service that publishes information about the virtual world's updates to the crawler.

2.3.2 Virtual Worlds Crawlers

The release of OpenSimulator in 2007 enabled the user to access the other simulators or the worlds with same account. Thus the expansion of virtual worlds of homogeneous or possibly the heterogeneous protocol demands search facility over the web. In order to provide search, the virtual worlds should be crawled similar to the WWW. The virtual world crawlers are the programs that navigate through the 3-D world by mimicking the normal user behavior. Eno et al. [4, 5, 7] demonstrated that virtual worlds could be effectively crawled with an autonomous agent that behaves like a normal human. In our thesis, we used the crawler developed by [4] to collect objects from our regions of study. In other work [6], Eno et al. examined landmarks and the picks to analyze the link between the regions. Their results showed that the regions in the virtual world are linked similarly to pages in the flat web. Although they primarily studied regions within Second Life, they detected evidence of existence of a denser link structure on virtual world sites hosted in OpenSimulator.

Indexing of the collected content is one of the most important steps in providing an effective search service. Users can only query for information that is properly indexed. Virtual

world crawlers must collect a wide variety of information in a wide variety of formats. The indexer must also extract content from these objects so that they can be indexed semantically and made searchable. For example, objects currently are represented by an ID and a collection of primitive graphic elements so users cannot search for objects semantically. Some graphic objects have explicit labels, but most do not, making it difficult for users to find “tables” or “businesses”. Similarly, the contents of notecards and other dynamically-generated text is not typically collected and/or searchable.

3. OUR APPROACH

Our goal is to present the design of a collaborative virtual world crawler and measure the benefits of this design in terms of reduced bandwidth consumption. Since the reduced bandwidth is primarily due to the avoidance of collection of redundant information, we need to measure the amount of redundant data collected by a traditional virtual world crawler for comparison. We collected our experimental data using a traditional virtual world crawler that repeatedly revisits regions and downloads all of the objects it can access. To do this, we developed a crawler that visits sample regions and we analyze the data collected, week by week, to identify any changes within that region. We then estimate the amount of data redundancy in this collection and propose a new architecture (Figure 2) that would avoid the collection of redundant data, reducing bandwidth and the time necessary to update a search service's database. In the following subsections we describe our proposed system architecture. We also define the necessary metrics to measure the bandwidths.

3.1 System Architecture

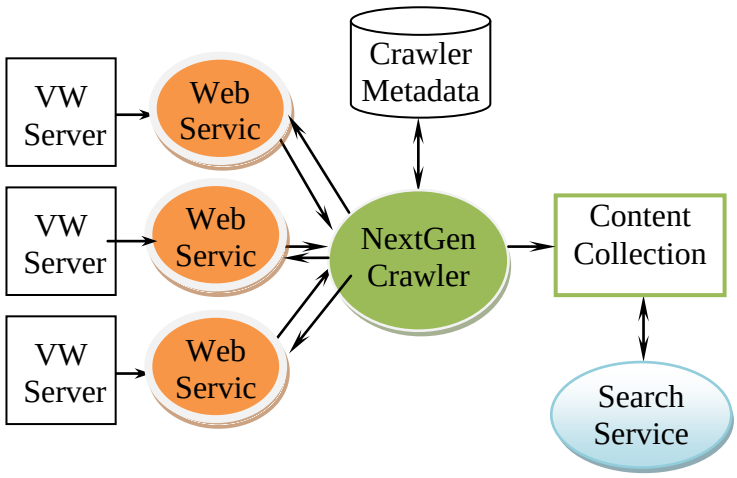

Figure 2: System Architecture

The proposed system consists of several components that have designated functions to support the crawler and collecting and indexing the data from the virtual worlds. As shown in Figure 2, the proposed system essentially consists of the same components as a web search engine, with the addition of a web service that interacts with the crawler. These components are:

Web Service: The web service is an XML-based messaging service between the crawler and the virtual world servers. The virtual world server publishes web service to inform crawler about the content metadata such as unique identifier of object, created date, last modified or deleted date. This is the main difference between our proposed architecture and a traditional virtual world crawler. By providing a web service as part of each virtual world server, the crawler and server can work collaboratively to provide more efficient data collection.

Crawler: The crawler is a program that collects user-generated content from virtual worlds by navigating through the virtual worlds' regions.

Metadata: The content collection system maintains metadata to inform the crawler agent about last crawled date and the objects collected most recently.

Content Collection: The content collection is the actual data collected by the crawler agent. The crawled data is stored in storage layer constructed in MySql database system.

Search Service: The search service provides an interface between the search engine and the user. It is the task of search service to expand and reformulate the user query to retrieve relevant results.

In the following subsections we discuss the individual components of the system architecture in detail.

3.1.1 Virtual World Server

Our proposed architecture would work as add-ons to the existing virtual world servers such as Second Life and OpenSimulator. A virtual world server is a multiuser, 3D application server that runs the simulated world. A virtual world server typically consists of spatial data, multimedia contents as well as textual contents. Depending on the architecture the virtual world, a server may host single region or a grid of multiple regions. A user has to login and navigate in a virtual world and create objects or interact with other users in the virtual worlds. A typical virtual world server does not provide support to the crawler, so, in order to crawl the virtual world, the crawler agent has to emulate the user behavior. Our approach proposed the extension of the architecture of the popular virtual worlds such as Second Life and OpenSimulator by publishing web services to support crawlers.

3.1.2 Web Service

The key contribution of this work is the addition of a crawler-assist web service on each virtual world. The virtual world servers know the details about all of the objects and avatars on their system. They can collect that information up by examining the contents of their database without the need for time-consuming and difficult to implement crawling. This information is what they use as the basis for their own site-search capabilities. Thus, the web service needs to have three primary functions: extracting and formatting the information for sharing, keeping track of modification/addition/deletion date; and responding to requests from external crawlers for appropriate subsets of the data.

The web services will publish the list of objects in the region along with the other metadata such as unique identifier of object, created date, last modified or deleted date. Current virtual worlds such as Second Life do not provide external access to their data or metadata. The web service will need to provide suitable access protocols to support the crawlers. We envision two types of requests. In one type of request, the web service would respond to a request for all of the information about a particular object, identified by a unique id. This would allow the crawler to refresh its index. The second type of request would require the web service to create a subset of objects based on metadata criteria. This would allow the web service to provide all objects of a certain type (e.g., notecards, avatars, graphics objects, etc., or all of the above) within a specific time range (e.g., since the last crawler visit) that meet a certain state (e.g., modified, deleted, or created. The web service should also respond, depending on the query, by providing either the metadata about the objects meeting the search criteria and/or providing the objects themselves.

To support this advanced querying, the web service will need to have a database indexed by object id and date that can quickly identify objects that meet the query criteria. As objects are modified, added, or deleted from the virtual world, the database supporting the web service would need to be updated as well.

```
<xs:element name="getObject">
   <xs:complexType>
      <xs:sequence>
          <xs:element minOccurs="0" name="createddate" nillable="true" type="xs:date"/>
          <xs:element minOccurs="0" name="regionid" nillable="true" type="xs:string"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
<xs:element name="getObjectResponse">
   <xs:complexType>
      <xs:sequence>
           <xs:element minOccurs="0" name="return" nillable="true" type="xs:string"/>
       </xs:sequence>
   </xs:complexType>
</xs:element>
```

Figure 3: Typical WSDL type definition of web service

```
<wsdl:message name="getObjectRequest">
      <wsdl:part name="parameters" element="ns:getObject"/>
</wsdl:message>
<wsdl:message name="getObjectResponse">
      <wsdl:part name="parameters" element="ns:getObjectResponse"/>
 </wsdl:message>
```

Figure 4: Typical message in WSDL

The proposed web service architecture provides functionalities to query list of objects in the region. Also the server exposes the functionalities to query the objects that were modified

and created during some time period. The WSDL[11] web service prototype for the virtual world server is shown in Figure 3 and Figure 4. Figure 3 shows the prototype of function to request the object by providing created date and region id whereas Figure 4 shows the messaging between server and client.

We envision a scenario in which the first time a crawler visits a particular virtual world, it requests metadata and data for all objects in the virtual world. Policy decisions will need to be considered regarding the appropriate sharing of avatar information. The best first approach is to store and share information about permanent objects only, excluding avatars, so that user privacy is preserved. On subsequent visits, the crawler would then request information about all objects created, modified, or deleted since the previous visit. For deleted objects, only the ID would need to be provided since this would be sufficient to identify and delete the objects from the crawler's index. For objects modified and/or created since the previous visit, the semantically meaningful object features should be provided so that they can be processed and added to the crawlers index for future searchability.

3.1.3 NextGen Crawler

The crawler in this new architecture does not really do any crawling; it merely requests information from the web service depending on the task that it is performing and uses the results it gets from the web server to update the content collection. Thus, we call it a "NextGen" crawler. It has two main components. The first component is in charge of all coordination and communication. Based on criteria specified in its metadata, it formulates requests for subsets of information from specific virtual worlds and receives data and metadata in response. The second

[11] http://www.w3.org/TR/wsdl

component is in charge of processing the data that is received, extracting index information where appropriate, and using that to update the content collection used by the search system.

3.1.4 Crawler Metadata

The search engine dispatches the crawler at regular intervals to crawl pre-existing as well as new regions. In doing so the search engine has to maintain the metadata to inform the crawler about the last crawled date and the objects collected in the most recent crawl. The crawler agent exploits this information to determine which virtual world to "crawl" next and what data to request from each world. The crawler can also save metadata about the rate of change per region so that future crawls can target fast-changing regions/worlds more frequently. Essentially, the crawler metadata controls the crawling process.

3.1.5 Content Collection

Collecting and indexing the visual as well as the textual content is an important step for building virtual world's search engine. Although virtual worlds have limited textual contents, the text that does exist provides useful hints about the region and the objects. The objects in virtual worlds have static text attributes that a client can obtain by sending request to the server; however the dynamic text contents that mostly appear as notecards or chats can only be obtained interacting in the virtual worlds.

Of a particular interest, our proposed system exploits the similar architecture of content collection system proposed by Eno [4]. The collection system is a three layered architecture consisting of client library that connects to virtual worlds, an agent based crawler and relational

database storage. The client library provided by OpenMetaverse[12] is implemented in C#. The storage layer was implemented in MySql and resembles to the collection schema of OpenSimulator. The agent based crawler mimics the normal human user and collects information about regions, objects, landmarks, avatars, parcels, avatars and groups, and store those information in the database provided by storage system.

One of the important variations in our collection system is that we exploit crawler metadata and web service to discover newly created, edited or deleted objects and focus the crawling system to collect static as well as dynamic contents in Second Life and OpenSimulator virtual worlds. As the content is discovered, it is added to both a metadata database and an inverted index structure for query retrieval.

3.1.6 Search Service

Once the contents are collected from the virtual worlds, those are converted into inverted index structure so that the search engine can retrieve and rank the results based on user queries. The search service acts as interface between the search engine and the user.

Our content collection system can collect visual as well as textual content of which large proportion is of visual content. As discussed earlier, static textual information comes from the object attributes and dynamic text from the chat or some other scripted objects such as notecards. So the textual information obtained contributes useful component of inverted index of our search engine. Besides these, the visual content could be indexed based on the textual attributes, spatial information and the rendering data. However detailed discussion of indexing the visual content is beyond the scope of this thesis.

[12] http://openmetaverse.org/projects/libopenmetaverse

From the user point of view, the user enters the query from the search interface of his/her choice and it is the job of the search service to receive a query from the search interface and retrieve the regions or the objects of interest of the user from the indexed content collection. The search service module is indeed the module that implements the method to expand the query and reformulate it to retrieve most relevant result.

3.2 Traditional Virtual World Crawler

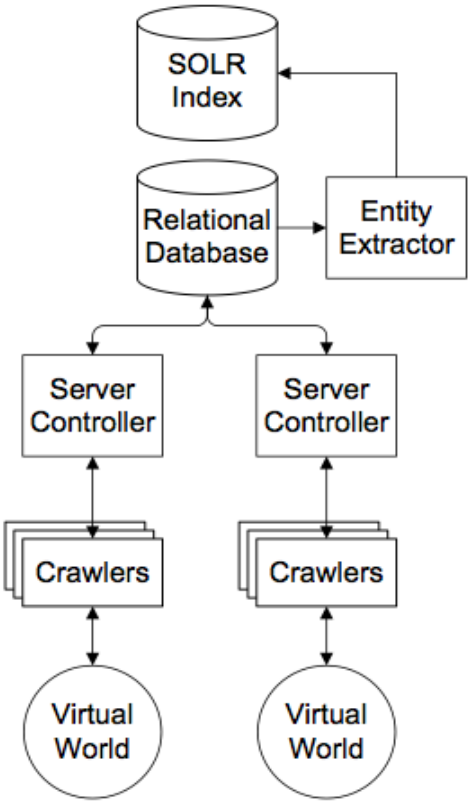

Figure 5: Traditional Virtual World Crawler Architecture [4]

For comparison with a baseline system, we compare the performance of our proposed architecture with an existing traditional crawler that has to navigate a virtual world to collect the content [4]. We also use this traditional crawler to collect data for the experiments described in

Chapter 4. The crawler is designed to collect user-generated content from Second Life and similar virtual worlds. As shown in Figure 5 the crawler agent consists of several components in order to set the navigation path, to control the movement and avoid collision. Specifically, the crawler server manager starts the crawling tasks by assigning specific regions to crawl for individual agents. Then, it keeps track of completed and queued regions as well as the status of the crawler in each region. The coordination layer consists of a server management program that coordinates individual crawler agents and includes components for duplicate detection and queue management for the entire virtual world. Once the data is collected from each region, it is saved by the storage layer that includes both database storage and searchable index storage [4].

The crawler program has been implemented with different input parameters allowing us better control of the collection process to facilitate research. It can be set up to autonomously navigate regions, parcels of land as well as to communicate with other avatars, user-created objects on the way it goes in order to collect different type of content including object positions, note cards, messages, links, etc. When a crawler instance is assigned to explore data from a specific region, it will attempt to teleport to that region. There are two phases in this crawling process [5]:

- When the crawler agent teleports successfully to a region, it will begin storing the object positions as they are automatically sent from the server. Each object is also added to a queue for object property requests, which are staggered to reduce server load and improve successful response rates.
- Next, the crawler agent begins moving around the region to get in close proximity to each object, a requirement for both object discovery and script triggering. Objects with touch actions are queued for later interaction.

Though the basic architecture of the crawler is the same in this thesis work, we modified the crawler behavior to enable the crawler to crawl in the list of regions that are designated for experimental data collection. So, instead of teleporting the crawler to some random region, we intentionally teleported the crawler to specified regions in each epoch of data collection.

4. EXPERIMENTS

Our crawler has the task of interacting with the virtual world environment to collect content. It has been built using the OpenMV library available from Open Metaverse. This C# library provides the Second Life client emulation functionality necessary to connect to a Second Life server, move around, and interact with other objects or avatars. Multiple instances of the crawler connect to the virtual world at one time, coordinating their activities through the Server Controller. We have designed a test scenario that would determine the effectiveness of our crawler and help us characterize different classes of objects that are found in Second Life regions.

4.1 Data Collection

Our first task was to select regions to crawl. Initially, a list of regions is obtained by configuring our crawler to teleport to random Second Life regions from a seed region. From 300 regions selected from Second Life servers, we calculated the normalized number of objects crawled and the normalized time rate of collecting objects from each region as shown in Figure 6. The smallest regions took only a few minutes to be crawled while the largest regions required hours to visit. Most of the regions among the 300 took from 6 to 12 minutes to crawl. In order to experiment with average sized regions, we excluded 100 regions including smallest and largest regions, and randomly selected half of the remaining 200 regions to obtain our final set of 100 regions.

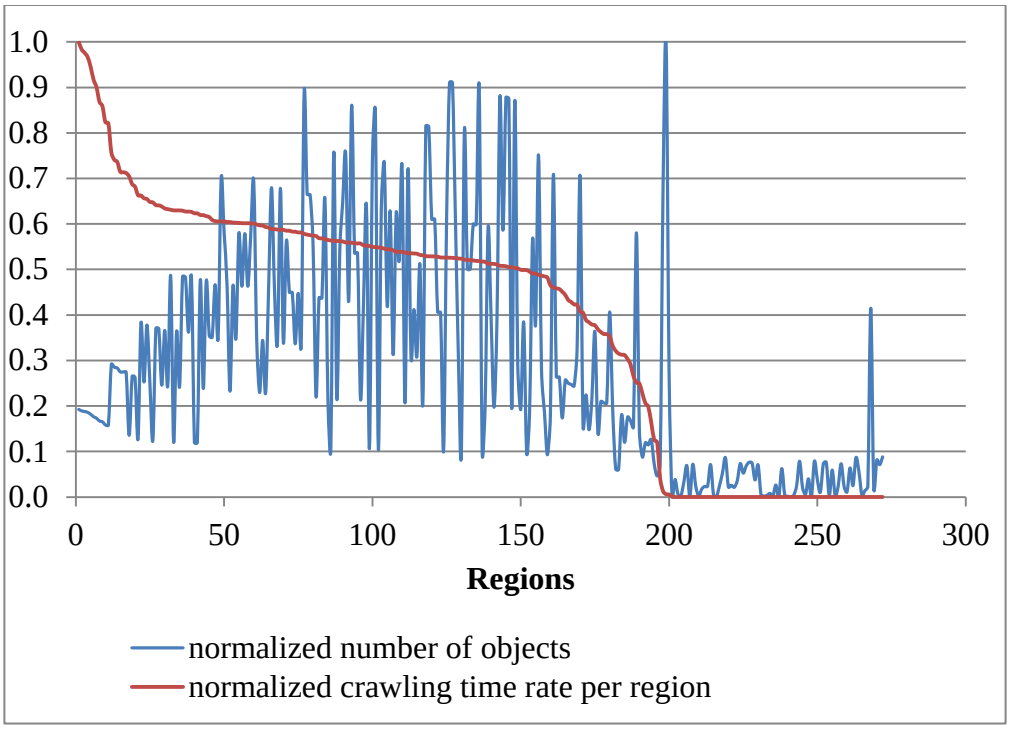

Figure 6: Selection of Second Life Regions for Experiments

In order to see how effectively the crawler captures objects in the virtual world, we launched the crawler with the same input parameters four times in four consecutive weeks. We started the crawler in a fixed time of the week (Thursday at 9:00AM) and saved data crawled for each week. Among 100 experimental regions, there were 3 regions from which the crawler was unable to collect objects; therefore we report our results over 97 regions. Table 1 shows a summary of data crawled over four weeks. For each snapshot, we report the total number of objects crawled. We then compare object records from consecutive crawls and calculate the number of objects added, deleted, modified and unchanged during each snapshot. Notice that the number of objects added and the number of objects deleted are relatively consistent from snapshot to snapshot, and are roughly 33% as large as the total number of objects crawled. The number of objects with modified attributes is also consistent from snapshot to snapshot, but these

values are less than 2% of the total number of objects. Finally, the number of unchanged objects is roughly 66% of the number of objects crawled.

Table 1: Data Collection from 4 Weeks

	#Objects crawled	#Objects added	#Objects deleted	#Objects modified	#Objects unchanged
Week 1 snapshot	490,850	0	0	0	0
Week 2 snapshot	486,294	157,492	162,048	5,627	323,175
Week 3 snapshot	481,313	154,860	159,841	8,074	318,379
Week 4 snapshot	449,729	136,653	168,237	8,293	304,783

4.2 Evaluation

In order to demonstrate the merits of our proposed architecture, we need to measure the amount of redundant data that the traditional crawler collects when it revisits a site. Based on that, we then calculate the potential bandwidth savings that our proposed architecture could achieve by avoiding this unnecessary data collection. In the following subsections, we define the metrics we use to measure data redundancy and bandwidth consumption savings.

4.2.1 Data Redundancy Metric

The main issue for any information crawler is how to avoid collecting the data that has been already downloaded. This creates unnecessary Internet traffic and wastes search engine, and virtual world server, resources during page collection and indexing.

In the virtual worlds, given the increased time necessary to physically navigate the 3D environment, it is particularly important to avoid collecting redundant data and prioritize

collecting the newly added objects or modified objects. We calculate the data redundancy collected by traditional crawling techniques by counting the common objects between two crawler collections, divided by the total number of unique objects collectedly either crawl.

$$DR = \frac{UC}{TC} * 100$$

where UC is the number of unchanged objects which are crawled in a region; and TC is total number of objects crawled from that region. The average data redundancy is calculated using the average UC and TC values over all regions that are crawled.

4.2.2 Bandwidth Consumption Savings Metric

In this section, we present a measure that estimates the potential savings in bandwidth consumed by the crawler using the proposed architecture. The bandwidth consumption savings metric is defined as follows:

$$BS = \frac{TC - UC}{TC} * 100$$

where UC and TC are defined as in the previous subsection. This bandwidth consumption savings value represents the percentage savings that a crawler can achieve by avoiding the collection of redundant data.

4.2.3 Data Redundancy Results

Our goal with this experiment was to get a sense of how much redundant information is collected during each region crawl. Figure 7 represents the data redundancy reported over four weeks. The DR_W2 curve shows the data redundancy between weeks 1 and 2. Similarly, the DR_W3 and DR_W4 curves show the redundancy between weeks 2 and 3, and between weeks 3

and 4. In all three cases, the redundancy values for each region have been sorted in decreasing order. Figure 8 combines the data redundancy over four weeks plotting the average data redundancy for each region crawled over the 4-week experiment. Notice that the median value for data redundancy is roughly 66%, which is consistent with the tabular data in Table 1.

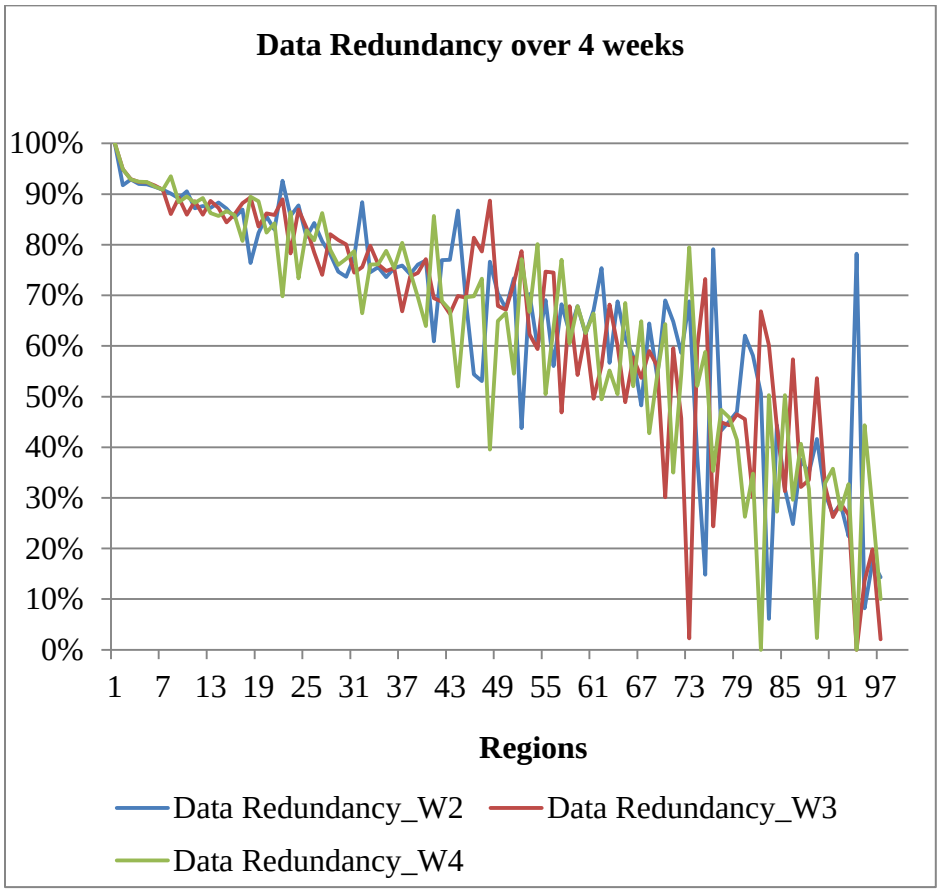

Figure 7: Data Redundancy Over 4 Weeks

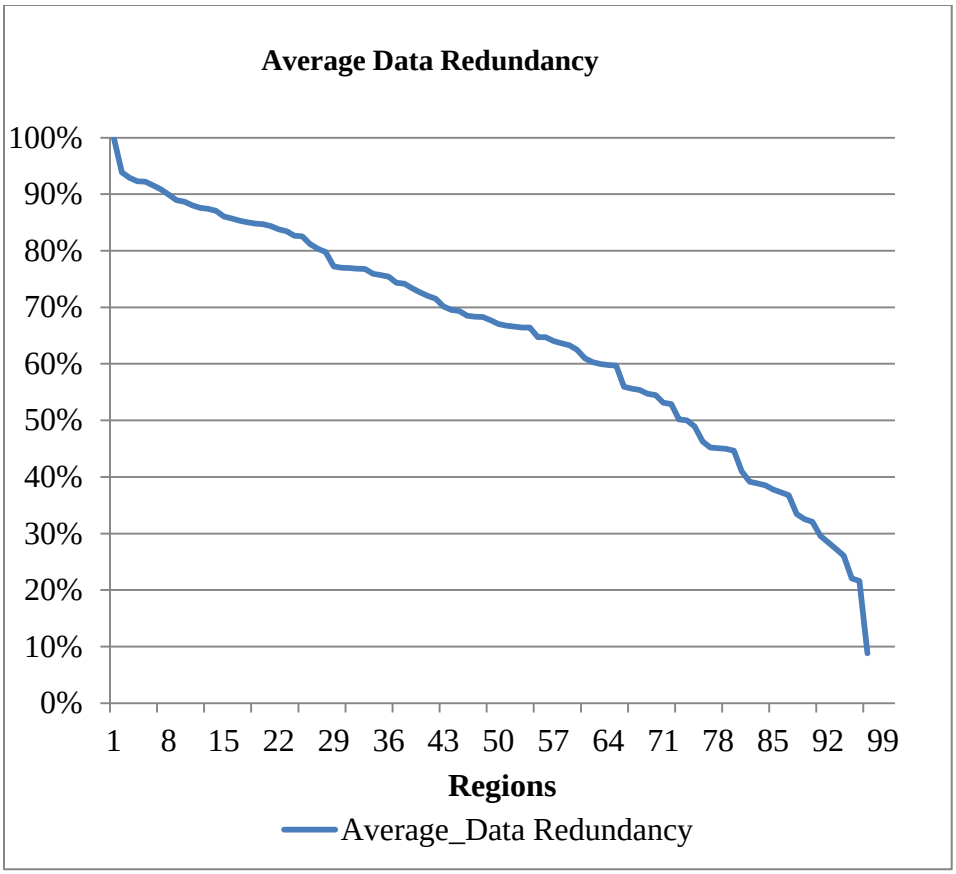

Figure 8: Average Data Redundancy

For each crawl, we also calculated the number of new objects that are created, the number of objects deleted, and the number of objects that are modified. In Table 2 we illustrate to how the total number of objects changed is distributed in each week. TC_W2 is the total number of objects changed between week 1 and 2 for selected regions. Similarly, TC_W3 and TC_W4 are the total number of objects changed between week 2 and 3, and between week 3 and 4. We presents the main percentile values at 25th, 50th and 75th positions in order to see how the number of objects are varied over four weeks.

Table 2. Percentile of number of objects changed over 4 weeks

Percentile	TC_W2	TC_W3	TC_W4
25th	4,138	4,119	4,072
50th	1,815	2,012	2,172
75th	1,324	1,213	1,287

4.2.4 Bandwidth Savings Results

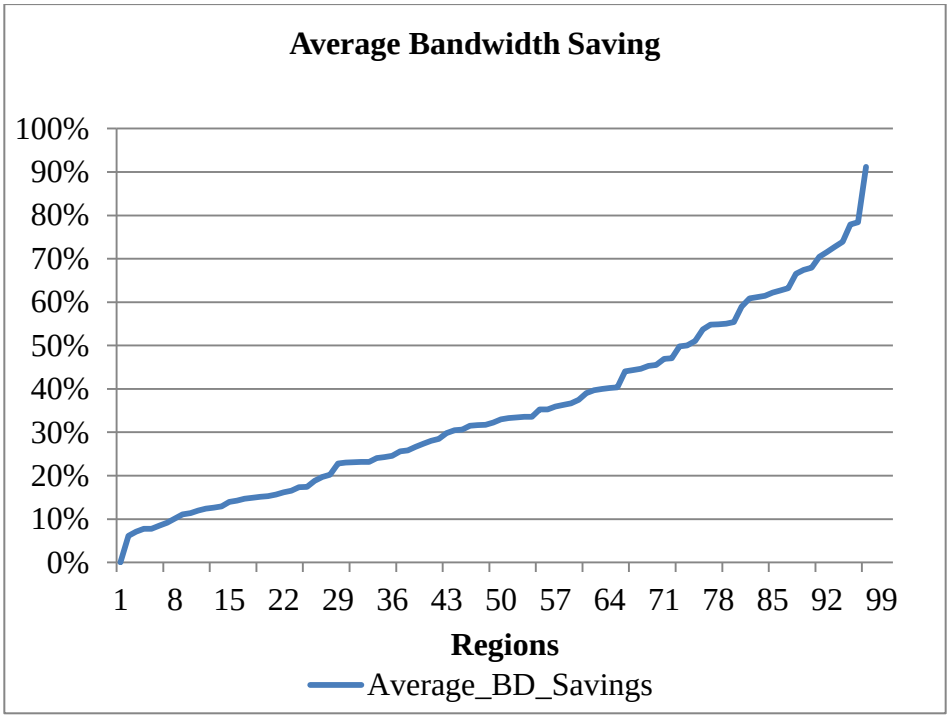

Figure 9: Average Bandwidth Saving

A main advantage of our approach is its ability to analyze the data redundancy and decrease the amount of bandwidth used by crawlers. Figure 9 represents the average bandwidth consumption saving for the whole data collection over four weeks.

4.4 Discussion

After collecting and investigating the data collected from our four weekly crawls of 100 regions of Second Life, we analyzed in detail the number of objects added, deleted and modified in each region. Using this information, we calculated the bandwidth savings that would be provided by our proposed architecture. Table 3 shows a summary of the 25th, 50th, and 75th percentile values of bandwidth saving over four weeks. These values are very consistent from week to week. The bandwidth savings for the 25th percentile region averaged 48.50%, which indicates that roughly ½ of the objects in these regions remain unchanged from week to week. The 50th percentile region had an average bandwidth saving of 31.70%, which is a significant drop from the 25th percentile region. Finally, the 75th percentile region only had a bandwidth saving of 16.37%, which is only 1/3 of the 25th percentile value.

Table 3. Percentile of bandwidth savings over 4 weeks

Percentile	BD Saving Week 2	BD Saving Week 3	BD Saving Week4	Average BD Saving
25th	45.86%	46.31%	49.70%	48.50%
50th	29.61%	30.14%	31.45%	31.70%
75th	15.16%	14.89%	16.50%	16.37%

According to our investigation, the crawler collected large amounts of redundant data over subsequent weeks because the majority of the content in virtual worlds are static objects such as chairs, trees, towers, buildings, etc. Thus, our proposed architecture has the potential to significantly decrease bandwidth and server and search system load by avoiding these unnecessary collections.

An added benefit to our approach is that the traditional crawler is unable to get certain types of data, particularly dynamic content such as sand boxes, avatar outfits, billboards and some robots that can appear at one place at a certain time and may disappear or be relocated at other times. In addition, because of the difficulty of navigating, the traditional crawler might miss objects that it fails to approach closely enough or that not in the range of visibility.

5. CONCLUSIONS

5.1 Summary

The goal of this research study is to implement and validate a collaborative crawler that collects data from virtual worlds. We have demonstrated that the crawler performance can achieve substantial bandwidth consumption savings since, with the help of a web service supported by the virtual world server, the crawler could avoid collecting redundant data. Our experiments analyzed the data redundancy due to the overlapping contents that are returned by the crawler to estimate the amount of bandwidth savings possible using this approach.

5.2 Contributions

The main contributions of this thesis are two fold.

Our first contribution is that we measured the amount of data redundancy in virtual worlds crawling and estimated the amount of bandwidth wasted collecting this redundant information. This can also be viewed as the amount of bandwidth that could be saved by having a more efficient crawling mechanism that avoided information recollection. We have shown that there is always a considerable amount of data redundancy in crawling virtual worlds since the crawler may have collected unmodified objects several times as it revisits a given virtual world. This will eventually consume unnecessary bandwidth for extracting the unchanged content again. In measuring data redundancy, we found that the median value of data redundancy is about 66% which means, half of the regions we crawled in Second Life had approximately 66% redundant data.

Our second contribution is that we proposed a co-operative architecture for a next generation crawler that exploits a web service published by the virtual world servers to get information about created, modified or deleted objects. Our approach was empirically tested based on the data we collected from Second Life servers that contained different kinds of objects in virtual worlds. The experimental results showed that, by avoiding redundant data in the crawling process, we can reduce the bandwidth consumed by crawling by up to 48% for 25% of the regions and about 32% for 50% of the regions.

The result from our experiment revealed that the co-operative approach of crawling the virtual world would save precious bandwidth, allowing the crawler to visit more sites in a given time period, increasing the comprehensiveness of the content collection and/or revisit sites more often, increasing the freshness of the content collection. The cost of this approach is some extra work by the virtual world server to maintain and provide access to the meta-information.

5.3 Future Work

Now we have demonstrated the potential advantages of our proposed architecture, our future work includes implementing a prototype Web Service and NextGen Crawler and using it to create a search service. This prototype can then be used as the basis of research into how to best control the crawling process to minimize bandwidth and maximize the content collection's breadth, depth, and freshness. With information in the crawler metadata about how quickly different virtual worlds and regions change, we could develop and evaluate different strategies for controlling the crawler so that regions could be visited in priority-order where the priority is a combination of how often the content in the region changes and how often users search for the

content in the region. Thus, fast-changing, high-demand regions would be crawled most frequently to be sure that the collection contains up-to-date content.

REFERENCES

[1] Bell, J., Dinova, M. and Levine, D. 2010. VWRAP for virtual worlds interoperability [Standards]. *Internet Computing, IEEE,* 14,1 (Jan.-Feb. 2010), 73-77. DOI=http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5370823&isnumber=5370810.

[2] Buzzi, M. 2003. Cooperative Crawling. In *Proceedings of first Latin American web Congress*(LA-WEB '03). IEEE Computer Society, Washington, DC, USA, 209-211, Nov.2003). DOI=http://doi.ieeecomputersociety.org/10.1109/LAWEB.2003.1250300.

[3] Chandramouli, A. 2007. *A co-operative web services paradigm for supporting crawlers*. Ph.D. dissertation, Univ. of Kansa, Lawrence, KS, USA.

[4] Eno, J. 2010. *An Intelligent Crawler For A Virtual World*. Ph.D. dissertation, Univ. of Arkansas, Fayetteville, AR, USA.

[5] Eno, J., Gauch, S. and Thompson, C. 2009. Intelligent Crawling in Virtual Worlds. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology. Volume 03* (WI-IAT '09), Vol. 3. IEEE Computer Society, Washington, DC, USA, 555-558. DOI=http://dx.doi.org/10.1109/WI-IAT.2009.348

[6] Eno, J., Gauch, S. and Thompson, C. 2010. Linking Behavior in a Virtual World Environment. In *Proceedings of the 15th International Conference on Web 3D Technology* (Web3D '10). ACM, New York, NY, USA, 157-164. DOI=http://doi.acm.org/10.1145/1836049.1836073.

[7] Eno, J., Gauch, S. and Thompson, C. 2009. Searching for the Metaverse. In *Proceedings of the 16th ACM Symposium on Virtual Reality Software and Technology* (VRST '09), Steven N. Spencer (Ed.). ACM, New York, NY, USA, 223-226. DOI=http://doi.acm.org/10.1145/1643928.1643976.

[8] Fernandes, S., Kamienski, C., Sadok, D., Moreira, J. and Antonello, R. 2007. Traffic Analysis Beyond This World: the Case of Second Life. In *Proceedings of Network and Operating Systems Support for Digital Audio and Video* (NOSSDAV, Illinois, USA, June 2007).

[9] Kinicki, J. and Claypool M. 2008. Traffic analysis of avatars in Second Life. In *Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video* (NOSSDAV '08). ACM, New York, NY, USA, 69-74. DOI=http://doi.acm.org/10.1145/1496046.1496063.

[10] La, C. A. and Michiardi, P. 2008. Characterizing User Mobility in Second Life. In *Proceedings of the first workshop on Online social networks* (WOSN '08). ACM, New York, NY, USA, 79-84. DOI=http://doi.acm.org/10.1145/1397735.1397753.

[11] Lopes, C. 2011. Hypergrid: Architecture and Protocol for Virtual World Interoperability. *IEEE Internet Computing* 15, 5 (Sept. 2011), 22-29. DOI=http://dx.doi.org/10.1109/MIC.2011.77.

[12] Varvello, M., Ferrari, S., Biersack, E. and Diot, C. 2011. Exploring second life. *IEEE/ACM Trans. Netw.* 19, 1 (Feb. 2011), 80-91. DOI=http://dx.doi.org/10.1109/TNET.2010.2060351.

[13] Varvello, M., Ferrari, S., Biersack, E. and Diot, C. 2009. Distributed avatar management for Second Life. In *Proceedings of the 8th Annual Workshop on Network and Systems Support for Games* (NetGames '09). IEEE Press, Piscataway, NJ, USA, , Article 5 , 6

[14] Varvello, M., Picconi, F., Diot, C. and Biersack, E. 2008. Is there life in Second Life?. In *Proceedings of the 2008 ACM CoNEXT Conference* (CoNEXT '08). ACM, New York, NY, USA, , Article 1 , 12 pages. DOI=http://doi.acm.org/10.1145/1544012.1544013.

[15] Yee, N. and Bailenson, J. N. 2008. A method for longitudinal behavioral data collection in second life. *Presence: Teleoper. Virtual Environ.* 17, 6 (December 2008), 594-596. DOI=http://dx.doi.org/10.1162/pres.17.6.594.

[16] Zang, Y., Yu, X., Dang, Y. and Chen, H. 2010. An Integrated Framework for Avatar Data Collection from the Virtual World. *IEEE Intelligent Systems* 25, 6 (Nov. 2010), 17-23. DOI=http://dx.doi.org/10.1109/MIS.2010.138.

[17] Zang, Y., Yu, X., Dang, Y. and Chen, H. 2010. An Integrated Framework for Avatar Data Collection from the Virtual World: A Case Study in Second Life. *IEEE Intelligent Systems 99*, 1. *DOI*=http://dx.doi.org/ 10.1109/MIS.2010.106.

[18] Pinkerton, B. 2000. *WebCrawler: Finding What People Want*. Ph.D. dissertation, Univ. of Washington.

[19] Cho, J., Molina, G. H. and Page, L. 1998. Efficient crawling through URL ordering. *Computer Networks and ISDN Systems* 30 (April 1998), 161-172.

[20] Najork, M. and Wiener, J. L. 2001. Breadth-first crawling yields high-quality pages. In *Proceedings of the 10th international conference on World Wide Web (WWW '01)*. ACM, New York, NY, USA, 114-118.

[21] Cho, J., Molina and G. H. 2000. The Evolution of the Web and Implications for an Incremental Crawler. In *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB '00),* 200-209.

[22] Brandman, O., Cho, J., Molina, G. H. and Shivakumar, N. 2000. Crawler-Friendly Web Servers. *SIGMETRICS Performance Evaluation Review.* 28, 2 (September 2000), 9-14.

[23] Gupta, V. and Campbell, R. 2001. Internet Search Engine Freshness by Web Server Help. In *Proceedings of the 2001 Symposium on Applications and the Internet (SAINT 2001) (SAINT '01)*. IEEE Computer Society, Washington, DC, USA, 113-.

[24] Chandramouli, A. and Gauch, S. 2007. A co-operative web services paradigm for supporting crawlers. In *Proceeding of Large Scale Semantic Access to Content (Text, Image, Video, and Sound) (RIAO '07)*. LE CENTRE DE HAUTES ETUDES INTERNATIONALES D'INFORMATIQUE DOCUMENTAIRE, Paris, France, France, 475-489.

[25] Chandramouli, A., Gauch, S. and Eno, J. 2010. A popularity-based URL ordering algorithm for crawlers.In *Proceeding of 3rd Conference on Human System Interactions (HSI)*. 556-562.