Spring 2015

IRC channel data analysis using Apache Solr

Nikhil Reddy Boreddy

Purdue University

Follow this and additional works at: http://docs.lib.purdue.edu/open_access_theses

Part of the Communication Technology and New Media Commons

Recommended Citation

http://docs.lib.purdue.edu/open_access_theses/551

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.
This is to certify that the thesis/dissertation prepared

By Nikhil Reddy Boreddy

Entitled
IRC CHANNEL DATA ANALYSIS USING APACHE SOLR

For the degree of Master of Science

Is approved by the final examining committee:

Dr. Marcus Rogers
Chair

Dr. John Springer

Dr. Eric Matson

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University’s “Policy of Integrity in Research” and the use of copyright material.

Approved by Major Professor(s): Dr. Marcus Rogers

Approved by: Dr. Jeffery L Whitten 3/13/2015

Head of the Departmental Graduate Program Date
IRC CHANNEL DATA ANALYSIS USING APACHE SOLR

A Thesis

Submitted to the Faculty

of

Purdue University

by

Nikhil Reddy Boreddy

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

May 2015
Purdue University
West Lafayette, Indiana
To my parents Bhaskar and Fatima: for pushing me to get my Masters before I join the corporate world!

To my committee chair Dr. Marcus Rogers: for introducing me to the world of cyber forensics.
ACKNOWLEDGEMENTS

This research would not have been possible without the guidance and support of my committee members: Dr. Marcus Rogers, Dr. John Springer and Dr. Eric Matson
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>vii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>viii</td>
</tr>
<tr>
<td>GLOSSARY</td>
<td>ix</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>xi</td>
</tr>
<tr>
<td>CHAPTER 1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Scope</td>
<td>1</td>
</tr>
<tr>
<td>1.1.1 Solr</td>
<td>1</td>
</tr>
<tr>
<td>1.1.2 IRC Channel</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Significance</td>
<td>2</td>
</tr>
<tr>
<td>1.3 Statement of Purpose</td>
<td>3</td>
</tr>
<tr>
<td>1.4 Research Question</td>
<td>4</td>
</tr>
<tr>
<td>1.5 Assumptions</td>
<td>4</td>
</tr>
<tr>
<td>1.6 Limitations</td>
<td>4</td>
</tr>
<tr>
<td>1.7 Delimitations</td>
<td>5</td>
</tr>
<tr>
<td>1.8 Summary</td>
<td>5</td>
</tr>
<tr>
<td>CHAPTER 2. LITERATURE REVIEW</td>
<td>7</td>
</tr>
<tr>
<td>2.1 Introduction</td>
<td>7</td>
</tr>
<tr>
<td>2.2 IRC Protocols</td>
<td>8</td>
</tr>
<tr>
<td>2.3 IRC Channels</td>
<td>9</td>
</tr>
<tr>
<td>2.4 IRC Identity</td>
<td>10</td>
</tr>
<tr>
<td>2.5 IRC Threats</td>
<td>10</td>
</tr>
<tr>
<td>2.6 Challenges in Forensic Analysis of IRC Channels</td>
<td>12</td>
</tr>
<tr>
<td>2.7 Searching Content on IRC Channels</td>
<td>12</td>
</tr>
<tr>
<td>2.8 Monitoring IRC Channels</td>
<td>12</td>
</tr>
<tr>
<td>2.9 Data Storage</td>
<td>13</td>
</tr>
</tbody>
</table>
### 2.10 Data Analysis ........................................................................................................... 13
### 2.11 Apache Lucene ....................................................................................................... 15
### 2.12 Forensics using Big Data ......................................................................................... 16
### 2.13 Comparison between Apache Solr and Relational Databases ......................... 17
### 2.14 Features of Apache Solr ......................................................................................... 18
### 2.15 Current Implementations ......................................................................................... 18
### 2.16 Summary .................................................................................................................. 19

#### CHAPTER 3. METHODOLOGY ..................................................................................... 20
### 3.1 Internet Relay Chat (IRC) ......................................................................................... 20
### 3.2 SQL Server ................................................................................................................ 21
### 3.3 Apache Solr ............................................................................................................... 21
### 3.4 Hypothesis ................................................................................................................ 23
### 3.5 Technical Specifications .......................................................................................... 24
### 3.6 Challenges ................................................................................................................ 24
### 3.7 Summary .................................................................................................................... 25

#### CHAPTER 4. RESULTS .................................................................................................. 26
### 4.1 Implementation ......................................................................................................... 26
#### 4.1.1 IRC Bots ............................................................................................................... 26
#### 4.1.2 MS SQL Server .................................................................................................... 28
#### 4.1.3 Apache Solr Server .............................................................................................. 29
### 4.2 Execution ................................................................................................................... 29
#### 4.2.1 MS SQL ................................................................................................................. 30
#### 4.2.2 Apache Solr .......................................................................................................... 30
### 4.3 Findings ..................................................................................................................... 32
### 4.4 Summary .................................................................................................................... 34

#### CHAPTER 5. DISCUSSION ............................................................................................ 35
### 5.1 Implications .............................................................................................................. 35
### 5.2 Importance ............................................................................................................... 37
### 5.3 Limitations ................................................................................................................. 37
### 5.4 Future Work ............................................................................................................... 38
LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3. 1: Specifications of Virtual Machines</td>
<td>24</td>
</tr>
<tr>
<td>4. 1: Search Query Processing Times</td>
<td>32</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. 1:</td>
<td>Popularity of IRC channels (Download.com, 2014)</td>
<td>8</td>
</tr>
<tr>
<td>2. 2:</td>
<td>A snapshot of response from Undernet server upon successful connection</td>
<td>9</td>
</tr>
<tr>
<td>2. 3:</td>
<td>A search results from Undernet</td>
<td>10</td>
</tr>
<tr>
<td>2. 4:</td>
<td>&quot;The increase of data size has surpassed the capabilities of computation.&quot; (Chen &amp; Zhang, 2014)</td>
<td>14</td>
</tr>
<tr>
<td>2. 5:</td>
<td>Solr vs Relational Database (Apache Solr, 2009)</td>
<td>17</td>
</tr>
<tr>
<td>3. 1:</td>
<td>Research Methodology</td>
<td>23</td>
</tr>
<tr>
<td>4. 1:</td>
<td>IRC Bot</td>
<td>27</td>
</tr>
<tr>
<td>4. 2:</td>
<td>IRC Parent Bot</td>
<td>27</td>
</tr>
<tr>
<td>4. 3:</td>
<td>IRC Parent Bot – Sample Email</td>
<td>28</td>
</tr>
<tr>
<td>4. 4:</td>
<td>MS SQL Server Specifications</td>
<td>28</td>
</tr>
<tr>
<td>4. 5:</td>
<td>Apache Solr Search Interface</td>
<td>31</td>
</tr>
<tr>
<td>4. 6:</td>
<td>Search Query Processing Times</td>
<td>33</td>
</tr>
<tr>
<td>4. 7:</td>
<td>Three and Five Data Points</td>
<td>34</td>
</tr>
</tbody>
</table>
GLOSSARY

The definitions of the terms used in the research are as follows:

ASCII - ASCII was originally developed for communications and uses only seven bits per character, providing 128 combinations that include upper and lower case alphabetic letters, the numeric digits and special symbols such as the $ and % (PC Mag, n.d.).

Big Data - A popular term used to describe the exponential growth and availability of data, both structured and unstructured (SAS, n.d.).

Digital forensics investigator – A law enforcement official trained in computers and who can legally seize computer evidence following the legal procedures stated in law and obeying jurisdiction and state/federal laws (Noble, Pollitt, & Presley, 2000).

Indexing – Logically dividing data into section of fields and using a unique specification as identifier (Targett, n.d.).

IRC – Internet relay chat. Exchanging information in text over the internet with one or multiple users at the same time following RFC 2813 protocol for server and RFC 2812 protocol for client (Network Working Group, 2000).

JSON - JSON is a text format that facilitates structured data interchange between all programming languages (ECMA International, 2013).

Unicode - The Unicode Standard is the universal character encoding standard used for representation of text for computer processing (Unicode, INC., 2014).

XML - Extensible Markup Language (XML) is an abbreviated version of Standard Generalized Markup Language (SGML), for the exchange of structured documents over the Internet (United States Environmental Protection Agency).

.NET – Is a framework on Windows machines which allows safe execution of code with performance improvement over interpreted or scripted languages (Microsoft, n.d.).

RFC – Requests for Comments. It is a publication that contains the standards and protocols to be used over the internet (Internet Engineering Task Force, n.d.).
Heap – A portion of the volatile computer memory that stores variables and continues to hold them till their reference is deleted (VmWare, n.d.).
ABSTRACT

Boreddy, Nikhil Reddy. M.S., Purdue University, May 2015. IRC Channel Data Analysis using Apache Solr. Major Professor: Dr. Marcus Rogers.

Internet Relay Chat (IRC) was one of the first real-time communication protocols over the internet. It was not designed with any form of Authentication, Authorization and Accounting features. This made IRC channels a place to conduct transactions in complete anonymity. On the other hand with the advent of Big Data we are now able to process large quantities of data in a very short period of time. This research presents a method to use Apache Solr, a text indexing server built on top of Lucene to index and search large quantities of IRC data collected over months from public IRC channels. It even presents a highly scalable approach to monitor public IRC channels by creation of IRC Client Bots which are in turn controlled by a robust IRC Parent Bot. The data thus collected is analyzed by Apache Solr and MS SQL servers and the response times are compared. This research concluded that Apache Solr outperforms MS SQL by a very great margin and such an implementation can be used by digital forensic investigators to monitor and search public IRC channels.
CHAPTER 1. INTRODUCTION

This chapter provides an overview of the research. It includes scope, significance, the research question, assumptions, limitations and delimitations of the research.

1.1 Scope

This section gives the scope of the present research.

1.1.1 Solr

Solr is an open source enterprise search platform from Apache Lucene. According to Apache, Solr was built in Java and runs as a server (Apache Solr, 2011). It specializes in full text indexing and even supports Unicode. It supports various formats of input and output including the widely used XML, CSV and JSON. Communicating with a Solr server is extremely simple as it was optimized to communicate over HTTP (Apache Solr, 2011). Solr derives most of its features from Apache Lucene. Lucene indexes Unicode at rates exceeding 150GB/hour with 1 MB heap (Apache Lucene, n.d.). It is also known to size the index between 20-30% of the size of input text. From a digital forensic investigator’s point of view, such a machine can improve the rate of analysis as most of the present systems use SQL databases to index which are not as quick as Solr when it comes to large indices.
1.1.2 IRC Channel

Undernet is the most popular IRC service provider. They maintain over 19 servers and are instrumental in connecting over 35 nations with more than 1,000,000 people weekly using their server as a point of contact with other IRC users (Underenet, n.d.). Each server has logical partitions which can be considered a chat room. Each chat room can be called a “channel” in technical terms. When users join a channel they can broadcast a message to everyone on that channel. In the present research, data was collected from active IRC channels over months and was analyzed using Solr. This process of using Solr in digital forensic investigations can be expanded to other forms of data.

1.2 Significance

With rise in the use of computers in every field, there was a significant increase in crime that involved the use of computers. FBI’s Internet Crime Complaint Center received 262,813 complaints in 2013 as opposed to 16,838 in 2000 (Federal Bureau of Investigation, 2013). Over the years there was a decrease in the cost of digital storage media and that led to higher storage capacities in computes. Digital forensic investigators need tools that can analyze and process data at a much faster pace. Sometimes the crucial evidence can be a couple of KBs but the investigator has to analyze over 1 GB to get to that evidence. On the other hand corporations are using “Big Data” tools, which let them analyze data in gigabytes per second (Intel, 2013). The research is aimed at demonstrating the feasibility of the use of Big Data concepts and tools for digital forensic investigations. Digital forensic investigators who are mostly law
enforcement can benefit a lot from the integration of new technology into their investigation process which can reduce their analysis and indexing time of raw data from seized evidence.

1.3 Statement of Purpose

IRC channels have been around since 1988. Though they started off as harmless chat rooms they quickly transformed into a platform for illegal activities. There is no log of chats on IRC channels, so the only way to investigate them is to join a group on an IRC server and make a local copy of the conversations among the group members.

Solr (Apache Solr, 2011) is an enterprise search platform from Apache. Its major features include faceted search, full-text search and near real-time indexing. In most of its default implementations, Solr is optimized for high volume web traffic. It was built on top of Apache Lucene (Apache Lucene, n.d.), a fault tolerant text search engine and is in the open source domain. Big data and data warehousing concepts are driving the data revolution in recent times. Big data platforms can decrease the data processing time and increase the efficiency of our analytical systems. Solr, which implements big data concepts, can be useful in the field of digital cyber forensics. It can be adopted to index IRC Unicode data and advanced automated searches can be performed on the conversations collected from the IRC channels. This type of implementation of Solr is not limited to Unicode data. It can also be used to analyze hex (hard disk image) and because it implements big data concepts the searches will be faster when
compared to commercially available forensic software. FTK and Encase use logical or relational databases, which perform slower as the size of the data increases.

1.4 Research Question

How does the use of Solr, an open source enterprise search platform, improve the ability to analyze large quantities of Unicode text collected by monitoring active IRC channels?

1.5 Assumptions

This research involves collection of data from various IRC channels. It also involves the use of Apache Solr. The assumptions inherent to this research include:

- The IRC channels selected to be monitored follow the Internet Relay Chat – Server Protocol as defined in RFC 2813.
- The monitored IRC channels will always have at least one participant other than the client used in this research.
- The SQL server which is used to store data from IRC channels, captures the Unicode strings without manipulating or truncating them. This can be verified by checking the SQL log files (Microsoft, 2014)
- The “Message Received” and “Message Stored” timestamps are perceived as one and the same for IRC channel messages.

1.6 Limitations

The limitations inherent to this research include:
• The study will be limited to six publicly accessible IRC channels due to server constraints. Based on a pilot study, each IRC client is expected to get around 1500 messages per hour. Each message of length ‘n’ characters would need \((2n + 2)\) bytes of storage space on the SQL Server (Microsoft, 2014). Therefore each IRC client would require almost 1000 kilobytes of disk space every hour to store messages, timestamps and metadata.

• The study will be limited to the use of Apache Solr as an indexing engine for Unicode strings collected from IRC channels.

• The study will be limited to exact match of keywords in the IRC data.

• The study will be limited to the collection of performance metrics on Windows machines only.

1.7 Delimitations

The delimitations inherent to this research include:

• SQL Server and Apache Solr will not be implemented in a non-windows environment.

• The Unicode string search using Apache Solr does not include search for synonyms and fuzzy patterns.

1.8 Summary

This chapter introduced the research contained in this thesis, along with the research question. It also contains the scope of the research along with
limitations and delimitations. It outlines the contribution of the research to the field by explaining its significance.
CHAPTER 2. LITERATURE REVIEW

This chapter intends to give an overview of literature related to the study of Internet Relay Chat (IRC) and analysis of data collected over multiple IRC channels using Apache Lucene and Apache Solr.

2.1 Introduction

Internet Relay Chat (IRC) is one of first real-time communication channels over the internet. The origins of IRC date data back to 1988. During early 1990s IRC service was started in the United States under the name “Undernet”. As of today Undernet runs approximately 19 servers serving more than 1,000,000 visitors weekly (Undernet, n.d.). IRC is one of the very few modes of communication over the internet that does not require any kind of sign-ups or registration. The people using these channels could choose to remain totally anonymous. Anonymity being the prime reason IRC has become a hub for unlawful activities. IRC has become a popular platform for exchange of information. M-IRC is one of the popular IRC clients for the windows platform. The popularity has been quantified in the Figure 2.1.
2.2 IRC Protocols

There are two main components in the IRC protocol. One is the IRC server protocol and the other is IRC client protocol. According to Network Working Group, “The IRC protocol itself enables several possibilities of transferring data between clients, and just like with other transfer mechanisms like email, the receiver of the data has to be careful about how the data is handled” (Network Working Group, 2000, p. 1). The client uses RFC 2812 protocol. The IRC server uses RFC 2813 protocol.
2.3 IRC Channels

Each IRC server is logically divided into channels or chat-rooms. People who connect to the IRC server can join one or multiple channels. Usually the public channels are named to attract attention like “CreditCard”, “AllNighters”, “AllCafe” etc. When a connection is made to any IRC server a brief overview of the server is sent to the user as a welcome message as shown in Figure 2.2.

Welcome to the UnderNet IRC Network, testwing
Your host is Chicago.II.US.Undernet.Org, running version u2.10.12.14
This server was created Sun Jul 7 2013 at 14:30:29 UTC
-
Chicago.II.US.Undernet.Org u2.10.12.14 dieswkgx biklnopsturDR bklou
-
WHOX WALLCHOPS WALLVOICES USERIP CRYPTOCH NOTICE SILENCE-25 MODES-6 MAXCHANNELS-40
MAXBANS=50  NICKLEN=12 are supported by this server
MAXNICKLEN=15  TOPICLEN=160  AWAYLEN=160  KICKLEN=150  CHANNELLEN=200  MAXCHANNELLEN=200
CHANTYPES=H&  PREFIX=0A+  STATUSMSG=0A+  CHANNAMES=b,k,l,nnmpstrDdr
CASEMAPPING=rfc1459 NETWORK=UnderNet are supported by this server
-
There are 1358 users and 27226 invisible on 36 servers
15 operator(s) online
114 unknown connection(s)
9384 channels formed
I have 898 clients and 1 servers
-
-Chicago.II.US.Undernet.Org- Highest connection count: 14084 (14083 clients)

Figure 2.2: A snapshot of response from Undernet server upon successful connection

All public channels on the IRC server are searchable by sending a message to the server. The syntax of the message is /list *search term* (IRC Help, 2014). A similar search using “card” as the search term in Undernet yields results shown in Figure 2.3.
The results usually contain two major fields. First is the channel or chat-room name and second is the number of active users on that group in real-time.

2.4 IRC Identity

There is no authentication or authorization restricting access on public IRC channels. Before connecting to the server, the client should submit a nickname for example “TalkyPrince”. As Bechar says “On IRC, a person's physical existence and identity must be condensed textually into a single line which states his or her nickname, the electronic address, and a slogan or the person's real name” (Bechar, 1995, p. 1). It is very common to find unconventional names on the IRC channels.

2.5 IRC Threats

A Bot can be defined as a program that was coded to run without human supervision. It be has the capability to perform a series of tasks over and over again. On a different note, BotNet refers to a collection of bots that perform the same task but from different host computers. According to Govil, “BotNet refers to a collection of Bot—a type of malware which allows an attacker to gain complete control over the affected computer—running on an Internet Relay Chat
(IRC) network that has been created with a Trojan” (Govil & Govil, 2007, p. 215). IRC protocols are fairly simple to implement as they just have two levels. One machine is the server and all other machines are clients. IRC is the most common protocol used in communication between bots and a bot server (Rajab, Zarfoss, Monrose, & Terzis, 2006). According to Network Working Group, “While based on the client-server model, the IRC (Internet Relay Chat) protocol allows servers to connect to each other effectively forming a network.” (Network Working Group, 2000, p. 1).

In the case of organized Distributed Denial of Service attacks IRC protocols would come in use. IRC server upon activation could send out a message to all the connected clients. Then the clients could start attacking a common target causing outages at the target and/or network congestion. These attacks are fairly difficult to prevent as the IRC clients might reside at different IP address ranges and geographical locations with low similarities. In most of the cases the malware infected clients would not even know that the attacks are originating from their networks.

According to NIJ, “IRC servers can also be used by writers of malicious code to gain control over infected computer systems” (National Institute of Justice, 2007, p. 44). Once the attacker gains control, a backdoor is created to maintain access. Rootkits may also be used to hide from any Intrusion Detection Systems which might be in place. “Once control over an infected computer is established, commands can be given that direct the infected computer to send e-
mail, transfer files, or probe other computer systems” (National Institute of Justice, 2007, p. 44).

2.6 Challenges in Forensic Analysis of IRC Channels

IRC server unlike other traditional server does not store any data on itself. It merely works as a relay protocol (Caraballo & Lo, 2014). Its job is to send data it receives from one client to all other clients on that particular channel. However, most of the modern implementations of the IRC protocol facilitate a client to start a private conversation with another client. From a digital forensics standpoint messages exchanged over the IRC are not saved on either side of the communication. Most of the IRC clients clear out the conversation once the connection is severed. Such non persistent data is a cause for concern as it provides secret channels for unlawful activities.

2.7 Searching Content on IRC Channels

Most of the data on the World Wide Web is actively indexed by multiple search engines and crawlers. According to Haveliwala, “Although a small number of transcripts for special events are occasionally published on the Web by individuals, there is no continuous, large-scale archive of IRC channels available” (Haveliwala, 2002, p. 395). This is one of the reasons why terrorists use IRC channels to exchange information and in some cases even to recruit people (Michels, 2012)

2.8 Monitoring IRC Channels

There is no easy way to retrieve data from IRC server/client pair. The best alternate solution would be to program IRC clients to act as passive listeners.
According to CNET, a reputed media company which specializes in information technology news, Rensselaer Polytechnic Institute successfully got a research grant approved for a proposal which said “The proposed system could aid the intelligence community to discover hidden communities and communication patterns in chat rooms without human intervention” (McCullagh, 2004).

2.9 Data Storage

According to Undernet it connects people from over 35 countries (Undernet, n.d.). Monitoring every part of such a massive network would require a lot of computation power and storage space. Usually the channels with a large number of participants are chosen to be monitored. All messages in the chosen channels are logged to a local database. In most cases this ends up to be a SQL or MySQL server.

2.10 Data Analysis

The start of 21st century the size of data increased exponentially. This was true in all sectors which used computers. Our traditional system were no longer able to process such vast amounts of data and we needed a newer and better architecture. Figure 2.4 would present the trend of data growth according to Chen and Zhang.
The data flow increased even in the IRC channels which led to increase in the size of data monitored/stored to local relational databases (Trend Micro, 2013). Even in the corporate world more than 50% of the companies believe that the use of Big Data would increase their productivity and help them understand their vendors and customers in a better manner (Chen & Zhang, 2014).

Figure 2.4: “The increase of data size has surpassed the capabilities of computation.” (Chen & Zhang, 2014).
2.11 **Apache Lucene**

“Apache Lucene is a high-performance, full-featured text search engine library written entirely in Java.” (Apache Lucene, n.d.). Some of the scalability aspects according to Apache’s official site include:

- Over 150GB/hour on modern hardware
- Small RAM requirements -- only 1MB heap
- Incremental indexing as fast as batch indexing
- Index size roughly 20-30% the size of text indexed. (Apache Lucene, n.d.).

Data analysis or pattern searches in text is one of the challenging tasks and According to Apache Lucene’s website it provides a platform with the following features:

- Ranked searching -- best results returned first
- Many powerful query types: phrase queries, wildcard queries, proximity queries, range queries and more
- Fielded searching (e.g. title, author, contents)
- Sorting by any field
- Multiple-index searching with merged results
- Allows simultaneous update and searching
- Flexible faceting, highlighting, joins and result grouping
- Fast, memory-efficient and typo-tolerant suggestions
- Pluggable ranking models, including the Vector Space Model and Okapi BM25 (Apache Lucene, n.d.).
Apache Lucene uses Big Data concepts including Hadoop but does not implement all the features of Hadoop. This is because of the fact that Lucene is intended to work as an indexer and not as a database engine. According to Gao, “Lucene is an excellent full text search engine, its structure has a strong object-oriented features” (Rujia, Li, Li, & Dong, 2012, p. 106).

2.12 Forensics using Big Data

According to Lee:

General text retrieval tools used in the digital forensics at present perform retrieval at an average speed of approximately 20 MB/s with respect to one query. It means 14 hours or more are required to retrieve a query in data of 1 TB and time or cost cannot be expected exactly in case of Big Data. Especially, the most time-consuming job for search is indexing because word vectors of every document have to be created. (Lee, Lee, Rhee, & Shin, 2014, pp. 1039-1040).

There is a need to make use of Big Data tools in forensic analysis of data because traditional software including relational databases process data at a very slow pace when compared to the size of data which is available to a normal user today. Lucene and Solr are the tools which are commercially in use for text searches. According to Solr’s official website, “Solr is the popular, blazing-fast, open source enterprise search platform built on Apache Lucene” (Apache Solr, 2011, p. 1). It was pointed out at 12th Annual International Conference on Digital Government Research that, a tool which could search documents in the U.S. patent system was being developed on top of Lucene and Solr because of their
instant indexing and faster search metrics (Lau, Taduri, Law, Yu, & Kesan, 2011).

Lee and Un (2012), have mentioned in their case study of forensic indexed search that use of Hadoop improves the parallel processing power of the environment and more data can be analyzed in a lesser period of time.

2.13 Comparison between Apache Solr and Relational Databases

<table>
<thead>
<tr>
<th>Lucene</th>
<th>Solr</th>
<th>Relational DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text Search</td>
<td>Fast and sophisticated</td>
<td>Minimal and slow</td>
</tr>
<tr>
<td>Features</td>
<td>Few, targeted to text search</td>
<td>Many</td>
</tr>
<tr>
<td>Deployment Complexity</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>Administration Tools</td>
<td>Minimal open source projects</td>
<td>Many open source &amp; commercial</td>
</tr>
<tr>
<td>Monitoring Tools</td>
<td>Weak</td>
<td>Very Strong</td>
</tr>
<tr>
<td>Scaling Tools</td>
<td>Automated, medium scale</td>
<td>Large scale</td>
</tr>
<tr>
<td>Support Availability</td>
<td>Weak</td>
<td>Strong</td>
</tr>
<tr>
<td>Schema Flexibility</td>
<td>Must in general rebuild</td>
<td>Changes immediately visible</td>
</tr>
<tr>
<td>Indexing Speed</td>
<td>Slow</td>
<td>Faster and adjustable</td>
</tr>
<tr>
<td>Query Speed</td>
<td>Text search is fast &amp; predictable</td>
<td>Very dependent on design &amp; use case</td>
</tr>
<tr>
<td>Row Addition/Extraction Speed</td>
<td>Slow</td>
<td>Fast</td>
</tr>
<tr>
<td>Partial Record Modification</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Time to visibility after addition</td>
<td>Slow</td>
<td>Immediate</td>
</tr>
<tr>
<td>Access to internal data structures</td>
<td>High</td>
<td>None</td>
</tr>
<tr>
<td>Technical knowledge required</td>
<td>Java (minimal), web server deployment, IT</td>
<td>SQL, DB-specific factors, IT</td>
</tr>
<tr>
<td>Regular maintenance tasks</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.5: Solr vs Relational Database (Apache Solr, 2009).

According to the official Apache Solr website Solr features are:

- Advanced Full-Text Search Capabilities
- Optimized for High Volume Web Traffic
- Standards Based Open Interfaces - XML, JSON and HTTP
- Comprehensive HTML Administration Interfaces
- Server statistics exposed over JMX for monitoring
- Linearly scalable, auto index replication, auto failover and recovery
o Near Real-time indexing
o Flexible and Adaptable with XML configuration

Most of the above mentioned features are lacking in our present relational database systems as they are designed for the purpose of storing data and not conducting extensive queries on it.

2.14 Features of Apache Solr

According its official website, Apache Solr builds on top of Apache Lucene to extend the search features. It even supports faceted search and filtering. Caching, query processing and standard deviation is possible in Apache Solr by default (Apache Solr, 2012). Tools with such capability come in handy when analyzing Unicode data which is collected from IRC channels.

2.15 Current Implementations

A group of researchers from Massachusetts Institute of Technology, created an engine called “Butterfly”, which served as a crawler for IRC channels (Dyke, Lieberman, & Maes, 1998). Given the vast number of channels the sampling time or the time spent on each channel was limited to 30 seconds. The search was limited to the channel name and the 30 second sample gathered.

Michels from Purdue University created a tool to provide real time text analysis on IRC conversations. This tool conducted frequency analysis on words contained in IRC conversations and maintained a log (Michels, 2012).
In both the above implementations the size of IRC channel data stored locally was limited. In addition to that the messages were not time stamped. The present research addressed both the issues. The IRC client bots which were programmed for this research not only time stamped each and every message but also stored messages to the SQL server in thousands per hour.

According to the official website of Solr it has been implemented in various scenarios where search operation is primary. It even supports PDF, Word, HTML content (Apache Solr, 2012).

The speed of data analysis in digital forensic investigations can be greatly improved by the use of Apache Lucene and Apache Solr. With the Application Programming Interfaces provided by Apache, it is fairly simple to automate the process of data analysis over IRC channels. An investigator could manually monitor and analyze a fixed number of IRC channels but tools which use Apache Solr can monitor a large number of channels and provide real-time indexing to the investigators.

### 2.16 Summary

This chapter summarized the existing literature on Internet Relay Chat and how the Internet Relay Chat networks can be used to attack computers. Additionally the chapter also provided data analysis metrics of Apache Solr in comparison to SQL Server.
CHAPTER 3. METHODOLOGY

This research is aimed at studying the improvement achieved by the use of Big Data concepts and tools to analyze data collected over months from multiple active Internet Relay Chat (IRC) channels. The first step was the creation of IRC client bots. Each client bot joined a different IRC channel as a passive user. Upon successful connection it logged all the messages exchanged in the group to a local SQL database. Data from the SQL database is then exported as CSV flat files. These files were then indexed by Apache Solr. Once the index was complete, the researcher could query Apache Solr directly for Unicode keywords using a HTTP request. The default response was a list of matches in XML format.

3.1 Internet Relay Chat (IRC)

IRC client bots were coded in C# on ASP.NET 4.5 framework. They were implemented following the RFC 2812 Internet Relay Chat – Client Protocol. The bots were programmed to connect to Undernet.org on their Chicago server at chicago.il.undernet.org on port 6667. Undernet was chosen for this research because the IRC provider has one million users per week and maintains over 19 servers worldwide (Undernet, n.d.).

Each IRC server has separate chat-rooms called channels and each channel has a name. Any IRC client that follows RFC 2812 guidelines can
connect to the IRC servers on a channel of choice. Channels are usually named
to attract people to share information anonymously for example “CreditCard”,
"MasterCard", “All Nighters” etc. There is no authentication on public channels.
The IRC clients usually connect to the server by providing the server with a
nickname that will be used in the channel as a chat name or screen name. A
message from any user on the channel is broadcast to each and every member
of the channel except the sender. The IRC bots that were developed for this
research stored each and every message on the monitored channels to a local
SQL database. Based on a pilot study, the bots were expected to gather around
one million messages per channel per month. Around half the messages were
not in standard ASCII characters.

3.2 SQL Server

The present research used SQL Database Server 2012 – Datacenter
Edition. Database management was through Microsoft SQL Server 2012
Management Studio; Version 12.0.2254.0.

3.3 Apache Solr

Most of the data collected by the IRC client bots was not in ASCII
characters so Apache Solr increased the speed of data analysis as text indexing
on Solr is almost instantaneous. Bitnami is an organization that develops cloud
applications. One of its latest products is Bitnami Apache Solr. Bitnami Apache
Solr is a virtual machine with a minimal operating system and a complete
implementation of Apache Solr capable of running in a VMplayer or VMware
(Bitnami). This present research used the latest version of Bitnami Apache Solr-4.10.0-0.

The data from the SQL database was extracted to a flat Comma Separated Value (CSV) file. This CSV file was given as input to the Solr server. An application was written in C# on ASP.NET 4.5 framework, which sent a HTTP POST request to the Solr server with the pattern of text. The given pattern was searched against the index and the appropriate fields that contain this pattern were communicated back the application using HTTP POST. The same process was repeated over and over with random text input patterns including Unicode. The response time for each search was noted. The same patterns were given as input to the SQL server and the response times were noted. The raw response times from Apache Solr and SQL server were then compared.
Pattern recognition is one of the prime deciding factors when choosing a data analysis engine. Solr with its faceted search outranks the commonly used SQL for analyzing large quantities of data.

3.4 Hypothesis

This research will compare the search query response times between SQL Server and Apache Solr over stored IRC channel data.

H₀ = There is no statistically significant difference between SQL Server and Apache Solr search query processing times and thus the use of Solr provides no reduction in data analysis duration.
\( H_a = \) There is a statistically significant difference between SQL Server and Apache Solr search query processing times. Therefore Solr can be used for faster search results.

3.5 Technical Specifications

The following is the technical specifications of the computers used in the research.

*Table 3. 1: Specifications of Virtual Machines*

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Processor</th>
<th>RAM</th>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRC Client Bot 1</td>
<td>Intel Core i5 – 4250U</td>
<td>8 GB</td>
<td>Windows 8.1 Pro</td>
</tr>
<tr>
<td>IRC Client Bot 2</td>
<td>Intel Core i5 – 4250U</td>
<td>8 GB</td>
<td>Windows 8.1 Pro</td>
</tr>
<tr>
<td>Data Analysis</td>
<td>Intel Core i5 – 4250U</td>
<td>2 GB</td>
<td>Windows 8.1 Pro</td>
</tr>
</tbody>
</table>

3.6 Challenges

The use of Big Data tools in computer forensic investigations is at a very nascent stage. Most of the local law enforcement agencies do not have computer specialists on team which limits the use of such newer technologies. From a technical stand point Big Data is an evolving science and any major changes it undergoes can change the way we interact with it. Such a drastic remodeling can be a matter of concern for this research.
3.7 Summary

This chapter summarized the proposed methodology for the research. It also covered all associated software, hardware and framework used to conduct analysis.
CHAPTER 4. RESULTS

This research was aimed at comparing the time-taken in searching keywords using MS SQL and Apache Solr. The data on which the searches were conducted was Internet Relay Chat (IRC) messages collected for the research over a course of three months. Prime result of this research was, Apache Solr outperformed MS SQL by a very great margin.

4.1 Implementation

This section gives the implementation of IRC bots, MS SQL Server and Apache Solr Server.

4.1.1 IRC Bots

IRC bots which were used in this research to monitor IRC channels were implemented in C# running on .NET Framework 4.5.1. Each instance of the IRC Bot monitored one IRC channel. A library called SmartIRC4Net was used in the creation of bots (Bauer, 2014).
A parent bot was programmed to refresh IRC Bots every 60 minutes. This prevented IRC bots from crashing. This parent bot also helped to keep the memory print as low as possible.

**Figure 4. 1: IRC Bot**

**Figure 4. 2: IRC Parent Bot**
A light-weight Simple Mail Transfer Protocol (SMTP) client was also implemented in the parent bot which enabled it to send hourly emails with details on messages collected in the last 60 minutes to the researcher.

Server: chicago.il.us.undernet.org
Group: carrier
Total Rows Collected till 2/5/2015 6:16:02 PM - 10.
Total Rows Collected in last 1 hour - 0.

Server: chicago.il.us.undernet.org
Group: validcard
Total Rows Collected till 2/5/2015 6:16:02 PM - 5526.
Total Rows Collected in last 1 hour - 1363.

Server: chicago.il.us.undernet.org
Group: CreditCards
Total Rows Collected till 2/5/2015 6:16:02 PM - 476.
Total Rows Collected in last 1 hour - 154.

Figure 4. 3: IRC Parent Bot – Sample Email

4.1.2 MS SQL Server

The IRC Channel messages were finally transferred to a fresh install of a MS SQL Server for analysis. Built-in MS SQL full-text indexing was used on the table which held IRC channel data.

Figure 4. 4: MS SQL Server Specifications
4.1.3 Apache Solr Server

IRC Channel messages were transferred to Apache Solr by the use of a program which was developed in C# running on .NET Framework 4.5. The program used SolrNET API (Scheffer, 2015). Latest version of Apache Solr, 4.10.3-0, was used in this research.

4.2 Execution

In this research, search query processing time of MS SQL and Apache Solr was compared for the following search terms.

- card
- master
- mastercard
- visa
- visacard
- credit card
- debit card
- banks
- banks acct
- moneypak
- usamoney
- western union
- money gram
- money pack
o bitcoin
o money order
o check
o ebay
o craigslist
o perfect money

The above terms were chosen after examining 1000 randomly selected IRC channel messages.

4.2.1 MS SQL

The following command was run in SQL Management Studio, replacing “Search Term” with the above list and the reported execution times were recorded:

SET STATISTICS TIME ON;
SELECT COUNT(*) FROM [IRC].[dbo].[IRCData]
where
FREETEXT(Rant,'%Search Term%');
SET STATISTICS TIME OFF;

4.2.2 Apache Solr

Apache Solr had a search handler built into the Solr interface. Search commands were issued from the provided web interface.
Figure 4. 5: Apache Solr Search Interface
4.3 Findings

The Search Query Processing Time taken by MS SQL and Apache Solr in milliseconds is shown in Table 4.1.

Table 4. 1: Search Query Processing Times

<table>
<thead>
<tr>
<th>Search Terms</th>
<th>MS SQL</th>
<th>Apache Solr</th>
</tr>
</thead>
<tbody>
<tr>
<td>card</td>
<td>2701.67</td>
<td>1928.33</td>
</tr>
<tr>
<td>master</td>
<td>983.67</td>
<td>478.00</td>
</tr>
<tr>
<td>mastercard</td>
<td>725.33</td>
<td>446.33</td>
</tr>
<tr>
<td>visa</td>
<td>6725.00</td>
<td>464.33</td>
</tr>
<tr>
<td>visacard</td>
<td>193.33</td>
<td>224.67</td>
</tr>
<tr>
<td>credit card</td>
<td>13344.00</td>
<td>218.33</td>
</tr>
<tr>
<td>debit card</td>
<td>11866.70</td>
<td>574.00</td>
</tr>
<tr>
<td>banks</td>
<td>5626.33</td>
<td>297.33</td>
</tr>
<tr>
<td>banks acct</td>
<td>4670.67</td>
<td>26.33</td>
</tr>
<tr>
<td>moneypak</td>
<td>1383.33</td>
<td>28.33</td>
</tr>
<tr>
<td>usamoney</td>
<td>3420.67</td>
<td>363.00</td>
</tr>
<tr>
<td>western union</td>
<td>6116.67</td>
<td>212.33</td>
</tr>
<tr>
<td>money gram</td>
<td>10526.30</td>
<td>583.00</td>
</tr>
<tr>
<td>money pack</td>
<td>2018.00</td>
<td>63.33</td>
</tr>
<tr>
<td>bitcoin</td>
<td>2011.33</td>
<td>362.00</td>
</tr>
<tr>
<td>money order</td>
<td>7976.33</td>
<td>52.67</td>
</tr>
<tr>
<td>Check</td>
<td>4907.00</td>
<td>456.67</td>
</tr>
</tbody>
</table>
Table 4.1 Continued

<table>
<thead>
<tr>
<th></th>
<th>eBay</th>
<th>Craigslist</th>
<th>Perfect Money</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4232.67</td>
<td>1096.3</td>
<td>30.67</td>
</tr>
<tr>
<td></td>
<td>1967.67</td>
<td>130.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3371.33</td>
<td>30.67</td>
<td></td>
</tr>
</tbody>
</table>

A graphical comparison of search query processing times of MS SQL and Apache Solr was generated in Microsoft Excel 2013.

Figure 4.6: Search Query Processing Times

In Figure 4.6, the data points three and five look extremely close due to the scale assumed. The graph when plotted for data points three and five is as shown in Figure 4.7.
According to the Figure 4.6 and Table 4.1, it is safe to conclude that Apache Solr on an average is ten times faster than MS SQL at indexing text. Hence the assumed null hypothesis is rejected.

4.4 Summary

This chapter summarized the implementation, execution and results of the present research. It also included the statistical analysis of results.
CHAPTER 5. DISCUSSION

This chapter is intended to discuss the implications of the research and future work.

5.1 Implications

This research proved that the use of Apache Solr in text searches produces faster results when compared to relational databases. Such a search engine could reduce the indexing time and enable the investigator to perform more searches per minute. Though this research was about analyzing IRC channel data, the concept of Unicode analysis using Apache Solr could serve in other applications like log analysis, network traffic analysis etc.

Apache Solr does not need any special software to access the server. The default implementation comes with a browser based GUI tool which enables the investigators to access the tool remotely from their browser. This is not the case with MS SQL which needs the installation of MS SQL Management Studio for client access. Hence Apache Solr can be rolled out at a massive scale fairly quickly. Apache Solr is licensed under Apache License, Version 2.0, which enables free usage for any purpose (Apache Solr, 2011). On the other hand MS SQL server is not free.
The IRC Parent Bot which was programmed for this research has the capability to control over a 100 IRC Bots given the storage space. Such customizable bots can really help investigators with IRC channels. IRC Parent Bot was customized to send an automated e-mail every fixed interval of time reporting the system status to the researcher. Such real-time updates relieve the investigator from constant manual monitoring of the system.

The IRC Bots were programmed to reconnect automatically upon facing network connectivity issues. The investigator can leave the system running because every bot clears out any unused extra memory every 60 minutes preventing operating system crash.

Some IRC servers disconnect clients who are not posting messages in groups they joined after a certain inactivity period. The IRC Bots were programmed to respond with a “Pong” message when the server checks for inactivity. The bots were also programmed to use a different chat name every 60 minutes. Automation at such a level can reduce the time an investigator needs to monitor IRC channels. This also enables the investigator to monitor more and more IRC channels with a minimum amount of work.

IRC Parent Bot and IRC Bots were implemented on .NET 4.5 framework, which is fully supported by Microsoft Windows. After .NET 4.5 optimization each IRC Bot takes about 100 Kilobytes of disk space and consumes about 4 Megabytes of RAM during execution. Digital investigators can spawn hundreds of these bots without much overhead.
5.2 Importance

This research could boost the capabilities of Digital Forensics in the arena of IRC. Without authentication, authorization and accounting IRC network has become a platform for illegal activities and the only way to counter is by monitoring public IRC channels. This research has shown that memory efficient Bots can be created and used to monitor hundreds and thousands of IRC channels with minimum effort and resources. The applications created were user friendly and need minimum training. Such an automated program could empower the investor to monitor thousands of IRC channels at the same time.

Once the data collection is complete, data should be analyzed for relevant matches. Historically even reputed digital forensics software like Forensics Toolkit(FTK) by Access Data used relational databases (Access Data, 2015). Relational databases are slower in performance when it comes to text indexing. This research has proved that the use of Apache Solr for text indexing produces results at least 10 faster than using MS SQL. It also indicated that Apache Solr can be used as a service and such almost zero cost upgrade could really boost the speed of data analysis which is a crucial part of digital forensic investigations.

5.3 Limitations

This research had the following limitations during data collection and data analysis.

- Data collection for this research involved monitoring six IRC channels over the period of three months. Two out of the six initially chosen channels
were almost dormant with less than three active participants for a greater part of one month.

- Apache Solr was installed on a Windows 8.1 Pro platform and was not tested on other platforms of Windows, Linux and Mac OS.
- MS SQL was installed on a Windows 8.1 Pro platform and was not tested on other platforms of Windows, Linux and Mac OS.
- Search index on Apache Solr and MS SQL servers was limited to default index in both.
- During data analysis messages from six different channels were treated as messages from one channel and that aggregated dataset was queried by Apache Solr and MS SQL in separate environments.

5.4 Future Work

This research can be taken forward in a lot of ways. The scope of analysis can be expanded by the use of WordNet which was developed by Princeton University (Princeton University, 2014). WordNet would enable synonym matching and semantic analysis of the text.

The IRC Bots used in this research can be modified to write to MongoDB, which has better response time over MS SQL. MongoDB stores documents as Key-Value pairs thereby increasing the performance metrics (MongoDB Inc., 2015). Such new technologies would also optimize the use of disk space.

This research can be expanded to thousands of IRC channels as Apache Solr supports Distributed Search. Distributed Search is searching across a
network of multiple Apache Solr servers and merging the search results in near real-time (Apache Solr, 2014).

The entire process of data collection can be automated in a server which supports .NET Framework 4.5. For data analysis the investigator can be granted access to an instance of Apache Solr. Apache Solr can be accessed through a web browser so no special client software would be necessary and that would cut down the set-up time for investigators.

The IRC Bots can also be used to send messages to other IRC users on the same channel. This feature can be used in future implementations to start conversations in IRC channels to gather information.

5.5 Summary

This chapter summarized the implications of the present research and its contribution to the field of digital forensics. Along with limitations of the present research this chapter also included future works which can be undertaken to carry this research forward.
REFERENCES
REFERENCES


APPENDIX

IRC Bot Code

using Meebey.SmartIrc4net;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace IRCBotConsole
{
    class Program
    {
        public static IrcClient irc = new IrcClient();

        //this is for database connection
        static SqlCommand com;
        static SqlConnection con;

        public static void connect()
        {
            
        }
con = new SqlConnection(con.ConnectionString = "Data Source=MACBOOKAIR;Integrated Security=True");
con.Open();

public static void OnQueryMessage(object sender, IrcEventArgs e)
{
    switch (e.Data.MessageArray[0])
    {
        case "dump_channel":
            string requested_channel = e.Data.MessageArray[1];
            Channel channel = irc.GetChannel(requested_channel);

            irc.SendMessage(SendType.Message, e.Data.Nick, "<channel "" + requested_channel + "/">");

            irc.SendMessage(SendType.Message, e.Data.Nick, "Name: "" + channel.Name + "/");
            irc.SendMessage(SendType.Message, e.Data.Nick, "Mode: "" + channel.Mode + "/");
            irc.SendMessage(SendType.Message, e.Data.Nick, "Key: "" + channel.Key + "/");

            string nickname_list = "";
            nickname_list += "Users: ";
            IDictionaryEnumerator it = channel.Users.GetEnumerator();
            while (it.MoveNext())
            { /* continue */ }
        
    }
}
{  
    string key = (string)it.Key;
    ChannelUser channeluser = (ChannelUser)it.Value;
    nickname_list += "(";
    if (channeluser.IsOp)
    {
        nickname_list += "@";
    }
    if (channeluser.IsVoice)
    {
        nickname_list += "+";
    }
    nickname_list += ")" + key + "+" => " + channeluser.Nick + ", ";
}
irc.SendMessage(SendType.Message, e.Data.Nick, nickname_list);

irc.SendMessage(SendType.Message, e.Data.Nick, "</channel>");
break;
case "gc":
    GC.Collect();
    break;
case "join":
    irc.RfcJoin(e.Data.MessageArray[1]);
    break;
case "part":
    irc.RfcPart(e.Data.MessageArray[1]);
    break;
case "die":
    Exit();
break;
}
}
public static void OnError(object sender, ErrorEventArgs e)
{
    System.Console.WriteLine("Error: " + e.ErrorMessage);
    //Exit();
}

public static void OnRawMessage(object sender, IrcEventArgs e)
{
    System.Console.WriteLine("Received: " + e.Data.RawMessage);

    string data = e.Data.RawMessage;

    try
    {
        com = new SqlCommand();
        com.Connection = con;

        com.CommandText = "insert into [HospitalData].dbo.Data1(Timestamp,Rant) values(CURRENT_TIMESTAMP,@data);";  


        com.ExecuteNonQuery();
    }
    catch(Exception exp)
    {
    }
public static void Main(string[] args) {
    connect();

    Thread.CurrentThread.Name = "Main";

    irc.SendDelay = 200;

    irc.ActiveChannelSyncing = true;

    irc.OnQueryMessage += new IrcEventHandler(OnQueryMessage);
    irc.OnError += new ErrorEventHandler(OnError);
    irc.OnRawMessage += new IrcEventHandler(OnRawMessage);

    string[] serverlist;

    serverlist = new string[] { "chicago.il.us.undernet.org" };  
    int port = 6667;
    string channel = "#creditcard";
    try {
        irc.Connect(serverlist, port);
    } catch (ConnectionException e)
try
{
    irc.Login("credit23", "credit23");
    // join the channel
    irc.RfcJoin(channel);
    Console.WriteLine("I am here after join channel");
    for (int i = 0; i < 3; i++)
    {
        irc.SendMessage(SendType.Message, channel, "test message (" + i.ToString() + ")");
    }
    new Thread(new ThreadStart(ReadCommands)).Start();
    irc.Listen();
    irc.Disconnect();
}
catch (ConnectionException)
{
    //Exit();
}
catch (Exception e)
```csharp

{ 
    System.Console.WriteLine("Error occurred! Message: " + e.Message);
    System.Console.WriteLine("Exception: " + e.StackTrace);
    //Exit();
}

public static void ReadCommands()
{
    while (true)
    {
        irc.WriteLine(System.Console.ReadLine());
    }
}

public static void Exit()
{
    System.Console.WriteLine("Exiting...");
    System.Environment.Exit(0);
}
}

IRC Parent BOT Code

using System;
using System.Collections.Generic;
```
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace AutoPilot
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

            progressBar1.Visible = true;

            progressBar1.Minimum = 0;
            progressBar1.Maximum = 3600;
            progressBar1.Step = 1;

            timer1.Interval = 1000;
            timer1.Start();
        }

        private void button1_Click(object sender, EventArgs e)
{  
    new Thread(new ThreadStart(botsControl)).Start();
}

private void botsControl()
{
    while (true)
    {
        try
        {
            Process[] myProcesses;

            string[] processes = textBox1.Text.Split(new char[] { '|' });

            foreach (string process in processes)
            {
                myProcesses = Process.GetProcessesByName(process);

                Thread.Sleep(Convert.ToInt32(textBox2.Text));

                foreach (Process myProcess in myProcesses)
                {
                    myProcess.Kill();
                    myProcess.WaitForExit();
                }
            }

            Thread.Sleep(2000);

            Console.WriteLine("At Process Start");
        }
    }
}
ProcessStartInfo startInfo = new ProcessStartInfo();
startInfo.WindowStyle = ProcessWindowStyle.Normal;
startInfo.FileName = textBox3.Text + process;
startInfo.Arguments = "mine";

Process application = new Process();
adapplication.StartInfo = startInfo;
adapplication.Start();

Console.WriteLine("At Process Started");
}
}catch(Exception exp)
{
    continue;
}
}
}

private void timer1_Tick(object sender, EventArgs e)
{
    if (progressBar1.Value == 3600)
        progressBar1.Value = 0;

    progressBar1.PerformStep();

    System.Diagnostics.Debug.WriteLine("in ticker");
}
}
namespace AutoPilot
{
    partial class Form1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #endregion
    }
}
private void InitializeComponent()
{
    this.components = new System.ComponentModel.Container();
    this.label1 = new System.Windows.Forms.Label();
    this.textBox1 = new System.Windows.Forms.TextBox();
    this.label2 = new System.Windows.Forms.Label();
    this.textBox2 = new System.Windows.Forms.TextBox();
    this.button1 = new System.Windows.Forms.Button();
    this.label3 = new System.Windows.Forms.Label();
    this.textBox3 = new System.Windows.Forms.TextBox();
    this.progressBar1 = new System.Windows.Forms.ProgressBar();
    this.timer1 = new System.Windows.Forms.Timer(this.components);
    this.SuspendLayout();
    //
    // label1
    //
    this.label1.AutoSize = true;
    this.label1.Location = new System.Drawing.Point(12, 18);
    this.label1.Name = "label1";
    this.label1.Size = new System.Drawing.Size(145, 13);
    this.label1.TabIndex = 0;
    this.label1.Text = "Enter the name of application";
    //
    // textBox1
    //
    this.textBox1.Location = new System.Drawing.Point(203, 18);
    this.textBox1.Name = "textBox1";
    this.textBox1.Size = new System.Drawing.Size(363, 20);
    this.textBox1.TabIndex = 1;
this.textBox1.Text = "SimpleProcess";

//

// label2
//
this.label2.AutoSize = true;
this.label2.Location = new System.Drawing.Point(12, 83);
this.label2.Name = "label2";
this.label2.Size = new System.Drawing.Size(87, 13);
this.label2.TabIndex = 2;
this.label2.Text = "Enter the interval";

//

// textBox2
//
this.textBox2.Location = new System.Drawing.Point(203, 83);
this.textBox2.Name = "textBox2";
this.textBox2.Size = new System.Drawing.Size(363, 20);
this.textBox2.TabIndex = 3;
this.textBox2.Text = "3000";

//

// button1
//
this.button1.Location = new System.Drawing.Point(171, 167);
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(226, 23);
this.button1.TabIndex = 4;
this.button1.Text = "Submit";
this.button1.UseVisualStyleBackColor = true;
this.button1.Click += new System.EventHandler(this.button1_Click);
// label3
this.label3.AutoSize = true;
this.label3.Location = new System.Drawing.Point(12, 57);
this.label3.Name = "label3";
this.label3.Size = new System.Drawing.Size(148, 13);
this.label3.TabIndex = 5;
this.label3.Text = "Path to the executables folder";

// textBox3
this.textBox3.Location = new System.Drawing.Point(203, 54);
this.textBox3.Name = "textBox3";
this.textBox3.Size = new System.Drawing.Size(363, 20);
this.textBox3.TabIndex = 6;
this.textBox3.Text = "C:\Users\Nikhil\Desktop\";

// progressBar1
this.progressBar1.Location = new System.Drawing.Point(15, 121);
this.progressBar1.Name = "progressBar1";
this.progressBar1.Size = new System.Drawing.Size(551, 23);
this.progressBar1.TabIndex = 7;

// timer1
this.timer1.Tick += new System.EventHandler(this.timer1_Tick);

// Form1
this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
this.ClientSize = new System.Drawing.Size(599, 202);
this.Controls.Add(this.progressBar1);
this.Controls.Add(this.textBox3);
this.Controls.Add(this.label3);
this.Controls.Add(this.button1);
this.Controls.Add(this.textBox2);
this.Controls.Add(this.label2);
this.Controls.Add(this.textBox1);
this.Controls.Add(this.label1);
this.Name = "Form1";
this.Text = "Optimus Prime";
this.ResumeLayout(false);
this.PerformLayout();

}
namespace IRCBot {
    public partial class Form1 : Form {

        IRCClient client = new IRCClient();
        DateTime startTime;

        public Form1()
        {
            InitializeComponent();
        }
    }
}
startTime = DateTime.Now;

timer1.Interval = 1000;
timer1.Start();

timer2.Interval = 1*60*60*1000;
timer2.Start();

Application.ApplicationExit += new EventHandler(IrcBotExit);
}

private void button1_Click(object sender, EventArgs e)
{
    SqlConnection con = new SqlConnection();
    con.ConnectionString = textBox1.Text;
    try
    {
        con.Open();
        MessageBox.Show("Connection Successful");
    }
    catch(Exception exp)
    {
        MessageBox.Show("The connection was not successful. Error "+exp.Message);
    }
    finally
    {
        con.Close();
    }
private void button2_Click(object sender, EventArgs e)
{
    SendEmail("Start","Ignore");
    //new Thread(new ThreadStart(collectRant)).Start();
}

private void collectRant()
{
    client.SqlConnectToServer(textBox1.Text, textBox7.Text);

    client.irc.SendDelay = 200;
    client.irc.ActiveChannelSyncing = true;

    client.irc.OnQueryMessage += new IrcEventHandler(client.OnQueryMessage);
    client.ircOnError += new ErrorEventHandler(client.OnError);
    client.irc.OnRawMessage += new IrcEventHandler(client.OnRawMessage);

    string[] serverlist;
    serverlist = new string[] { "chicago.il.us.undernet.org"};
    int port = Convert.ToInt32(textBox3.Text.Trim());
    string channel = "#" + textBox7.Text;
    try
    {
        client.irc.Connect(serverlist, port);
    }
    catch (ConnectionException exp)
{  
  Console.WriteLine("couldn't connect! Reason: " + exp.Message);  
  SendEmail("Error", exp.ToString());  
  //client.Exit();
}

try
{
    client.irc.Login(textBox4.Text, textBox5.Text);  
    client.irc.RfcJoin(channel);  
    Console.WriteLine("I am here after join channel");
    for (int i = 0; i < 3; i++)
    {
        client.irc.SendMessage(SendType.Message, channel, "test message (" + i.ToString() + ")");
    }
    client.irc.Listen();
    client.irc.Disconnect();
}
catch (ConnectionException)
{
    SendEmail("Error", "ConnectionException");  
    //client.Exit();
}
catch (Exception exp)
{
    SendEmail("Error", exp.ToString());
}
Console.WriteLine("Error occurred! Message: " + exp.Message);
Console.WriteLine("Exception: " + exp.StackTrace);
//client.Exit();
}
}

private void timer1_Tick(object sender, EventArgs e)
{
textBox6.Text = client.realTimeRant;
label9.Text = "Total Run Time : " + (DateTime.Now - startTime).ToString();
Console.WriteLine("This is rant from ticker");
}

private void timer2_Tick(object sender, EventArgs e)
{
    //if((DateTime.Now - startTime).TotalHours % 4 == 0)
    {
        SendEmail("Reporting", "Ignore");
    }
}

private void SendEmail(string emailType, string message)
{
    MailMessage email = new MailMessage();
    email.From = new MailAddress("nikhilboreddy@hotmail.com");
    email.To.Add("nikhilboreddy@gmail.com");

    if (string.Compare("Start", emailType) == 0)
{
    email.Subject = "AutoBot Reporting";
    email.Body = "The IRC Bot execution started for all" + textBox2.Text + "<br />
" + "<br />Thanks,<br />AutoBot Engineering Team";
}
else if(string.Compare("Error",emailType) == 0)
{
    email.Subject = "AutoBot Reporting Error";
    email.Body = "There was an error in IRC execution in " + textBox7.Text + " on " + textBox2.Text + ". Error " + message + "<br />" + "<br />Thanks,<br />AutoBot Engineering Team";
}
else if(string.Compare("Reporting",emailType)==0)
{
    email.Subject = "AutoBot Reporting";

    try
    {
        SqlConnection con = new SqlConnection();
        con.ConnectionString = textBox1.Text;
        SqlCommand com = new SqlCommand();
        con.Open();
        com.CommandText = "select count(*) from AssignmentHospitalData.dbo.carding;";
        SqlDataReader reader = com.ExecuteReader();
    }
reader.Read();
string rowsCollected = reader[0].ToString();
reader.Close();

email.Body = "Server : " + textBox2.Text + "<br />" +
"Group : " + "carding" + "<br />" +
"Total Rows Collected till " + DateTime.Now.ToString() + " - " +
rowsCollected + ".<br />

com = new SqlCommand();
com.Connection = con;
com.CommandText = "SELECT COUNT(*) FROM [AssignmentHospitalData].[dbo].[carding] +
"where Timestamp > "'" + DateTime.Now.AddHours(-1).ToString() + "";
com.CommandTimeout = 0;
reader = com.ExecuteReader();
reader.Read();
rowsCollected = reader[0].ToString();
reader.Close();

email.Body += "Total Rows Collected in last 1 hour - " +
rowsCollected + ".<br />

com = new SqlCommand();
com.Connection = con;
com.CommandText = "select count(*) from AssignmentHospitalData.dbo.mastercard;"
com.CommandTimeout = 0;
reader = com.ExecuteReader();
reader.Read();
rowsCollected = reader[0].ToString();
reader.Close();

email.Body += "<br />Server : " + textBox2.Text + "<br />" + "Group : " + "mastercard" + "<br />" + "Total Rows Collected till " + DateTime.Now.ToString() + " - " + rowsCollected + ".<br />";

com = new SqlCommand();
com.Connection = con;
com.CommandText = "SELECT COUNT(*) FROM [AssignmentHospitalData].[dbo].[mastercard] " + "where Timestamp > " + DateTime.Now.AddHours(-1).ToString() + "";
com.CommandTimeout = 0;
reader = com.ExecuteReader();
reader.Read();
rowsCollected = reader[0].ToString();
reader.Close();

email.Body += "Total Rows Collected in last 1 hour - " + rowsCollected + ".<br />";

com = new SqlCommand();
com.Connection = con;
com.CommandText = "select count(*) from AssignmentHospitalData.dbo.creditcard;";
com.CommandTimeout = 0;
reader = com.ExecuteReader();
reader.Read();
rowsCollected = reader[0].ToString();
reader.Close();

email.Body += "<br />Server : " + textBox2.Text + "<br />" +
"Group : " + "creditcard" + "<br />" +
"Total Rows Collected till " + DateTime.Now.ToString() + " - " +
rowsCollected + ".<br />

com = new SqlCommand();
com.Connection = con;

com.CommandText = "SELECT COUNT(*) FROM [AssignmentHospitalData].[dbo].[creditcard] " +
"where Timestamp > "+ DateTime.Now.AddHours(-1).ToString() + ";
com.CommandTimeout = 0;
reader = com.ExecuteReader();
reader.Read();
rowsCollected = reader[0].ToString();
reader.Close();

email.Body += "Total Rows Collected in last 1 hour - " +
rowsCollected + ".<br />" +
"<br />Thanks,<br />AutoBot Engineering Team";

con.Close();
}
catch(Exception exp)
{

email.Subject = "AutoBot Reporting Error";
email.Body = "There was an error in AutoBot reporting. Error " +
exp.ToString() + "<br />" +
"<br />Thanks,<br />AutoBot Engineering Team";
}
else if (string.Compare("AppExit", emailType) == 0) {
    email.Subject = "AutoBot Reporting App Close";
    email.Body = "There was an error in AutoBot. Someone closed the application. "+"<br />" +
    "<br />Thanks,<br />AutoBot Engineering Team";
}
email.IsBodyHtml = true;

SmtpClient smtp = new SmtpClient("smtp.live.com", 587);
smtp.Credentials = new NetworkCredential("nikhilboreddy@hotmail.com", **************);
smtp.EnableSsl = true;
smtp.Send(email);

private void IRCBotExit(object sender, EventArgs e) {
    SendEmail("AppExit", "Ignore");
}

namespace IRCBot
{
    partial class Form1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        // ...
    }
}
private System.ComponentModel.IContainer components = null;

/// <summary>
/// Clean up any resources being used.
/// </summary>
/// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
protected override void Dispose(bool disposing)
{
    if (disposing && (components != null))
    {
        components.Dispose();
    }
    base.Dispose(disposing);

#region Windows Form Designer generated code

/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    this.components = new System.ComponentModel.Container();
    this.label1 = new System.Windows.Forms.Label();
    this.textBox1 = new System.Windows.Forms.TextBox();
    this.button1 = new System.Windows.Forms.Button();
    this.label2 = new System.Windows.Forms.Label();
}
this.textBox2 = new System.Windows.Forms.TextBox();
this.label3 = new System.Windows.Forms.Label();
this.textBox3 = new System.Windows.Forms.TextBox();
this.label4 = new System.Windows.Forms.Label();
this.label5 = new System.Windows.Forms.Label();
this.textBox4 = new System.Windows.Forms.TextBox();
this.textBox5 = new System.Windows.Forms.TextBox();
this.button2 = new System.Windows.Forms.Button();
this.textBox6 = new System.Windows.Forms.TextBox();
this.label6 = new System.Windows.Forms.Label();
this.timer1 = new System.Windows.Forms.Timer(this.components);
this.label7 = new System.Windows.Forms.Label();
this.textBox7 = new System.Windows.Forms.TextBox();
this.label8 = new System.Windows.Forms.Label();
this.timer2 = new System.Windows.Forms.Timer(this.components);
this.label9 = new System.Windows.Forms.Label();
this.SuspendLayout();
//
// label1
//
this.label1.AutoSize = true;
this.label1.Location = new System.Drawing.Point(22, 22);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(210, 13);
this.label1.TabIndex = 0;
this.label1.Text = "Connection String - Database (SQL Server)";
//
// textBox1
//
this.textBox1.Location = new System.Drawing.Point(248, 22);
this.textBox1.Name = "textBox1";
this.textBox1.Size = new System.Drawing.Size(393, 20);
this.textBox1.TabIndex = 1;
this.textBox1.Text = "Data Source=MACBOOKAIR;Integrated Security=True";

// // button1
//
this.button1.Location = new System.Drawing.Point(647, 22);
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(134, 23);
this.button1.TabIndex = 2;
this.button1.Text = "Check Connection";
this.button1.UseVisualStyleBackColor = true;
this.button1.Click += new System.EventHandler(this.button1_Click);

// // label2
//
this.label2.AutoSize = true;
this.label2.Location = new System.Drawing.Point(22, 54);
this.label2.Name = "label2";
this.label2.Size = new System.Drawing.Size(107, 13);
this.label2.TabIndex = 3;
this.label2.Text = "IRC Channel - Server";

// // textBox2
//
this.textBox2.Location = new System.Drawing.Point(248, 54);
this.textBox2.Name = "textBox2";
this.textBox2.Size = new System.Drawing.Size(393, 20);
this.textBox2.TabIndex = 4;
this.textBox2.Text = "chicago.il.us.undernet.org";

//
// label3
//
this.label3.AutoSize = true;
this.label3.Location = new System.Drawing.Point(22, 85);
this.label3.Name = "label3";
this.label3.Size = new System.Drawing.Size(95, 13);
this.label3.TabIndex = 5;
this.label3.Text = "IRC Channel - Port";

//
// textBox3
//
this.textBox3.Location = new System.Drawing.Point(248, 85);
this.textBox3.Name = "textBox3";
this.textBox3.Size = new System.Drawing.Size(393, 20);
this.textBox3.TabIndex = 6;
this.textBox3.Text = "6667";

//
// label4
//
this.label4.AutoSize = true;
this.label4.Location = new System.Drawing.Point(22, 140);
this.label4.Name = "label4";
this.label4.Size = new System.Drawing.Size(129, 13);
this.label4.TabIndex = 7;
this.label4.Text = "IRC Channel - Nick Name";

//
// label5
//
this.label5.AutoSize = true;
this.label5.Location = new System.Drawing.Point(22, 112);
this.label5.Name = "label5";
this.label5.Size = new System.Drawing.Size(133, 13);
this.label5.TabIndex = 8;
this.label5.Text = "IRC Channel - Login Name";

//
// textBox4
//
this.textBox4.Location = new System.Drawing.Point(248, 112);
this.textBox4.Name = "textBox4";
this.textBox4.Size = new System.Drawing.Size(393, 20);
this.textBox4.TabIndex = 9;
this.textBox4.Text = "ChatKing6677";

//
// textBox5
//
this.textBox5.Location = new System.Drawing.Point(248, 140);
this.textBox5.Name = "textBox5";
this.textBox5.Size = new System.Drawing.Size(393, 20);
this.textBox5.TabIndex = 10;
this.textBox5.Text = "ChatBuddy";

//
// button2
//
this.button2.Location = new System.Drawing.Point(25, 202);
this.button2.Name = "button2";
this.button2.Size = new System.Drawing.Size(616, 23);
this.button2.TabIndex = 11;
this.button2.Text = "Start Monitoring";
this.button2.UseVisualStyleBackColor = true;
this.button2.Click += new System.EventHandler(this.button2_Click);

this.textBox6.Location = new System.Drawing.Point(25, 270);
this.textBox6.Multiline = true;
this.textBox6.Name = "textBox6";
this.textBox6.Size = new System.Drawing.Size(756, 60);
this.textBox6.TabIndex = 12;

this.label6.AutoSize = true;
this.label6.Location = new System.Drawing.Point(22, 243);
this.label6.Name = "label6";
this.label6.Size = new System.Drawing.Size(81, 13);
this.label6.TabIndex = 13;
this.label6.Text = "Real Time Rant";

this.timer1.Tick += new System.EventHandler(this.timer1_Tick);
//
// label7
//
this.label7.AutoSize = true;
this.label7.Location = new System.Drawing.Point(22, 169);
this.label7.Name = "label7";
this.label7.Size = new System.Drawing.Size(136, 13);
this.label7.TabIndex = 14;
this.label7.Text = "IRC Channel - Group Name";
//
// textBox7
//
this.textBox7.Location = new System.Drawing.Point(248, 169);
this.textBox7.Name = "textBox7";
this.textBox7.Size = new System.Drawing.Size(393, 20);
this.textBox7.TabIndex = 15;
this.textBox7.Text = "carding";
//
// label8
//
this.label8.AutoSize = true;
this.label8.Location = new System.Drawing.Point(644, 140);
this.label8.Name = "label8";
this.label8.Size = new System.Drawing.Size(169, 13);
this.label8.TabIndex = 16;
this.label8.Text = "Total Number of Rows Collected : ";
//
// timer2
//
this.timer2.Tick += new System.EventHandler(this.timer2_Tick);

//
// label9
//
this.label9.AutoSize = true;
this.label9.Location = new System.Drawing.Point(647, 88);
this.label9.Name = "label9";
this.label9.Size = new System.Drawing.Size(89, 13);
this.label9.TabIndex = 17;
this.label9.Text = "Total Run Time : ";

//
// Form1
//
this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
this.ClientSize = new System.Drawing.Size(886, 342);
this.Controls.Add(this.label9);
this.Controls.Add(this.label8);
this.Controls.Add(this.textBox7);
this.Controls.Add(this.label7);
this.Controls.Add(this.label6);
this.Controls.Add(this.textBox6);
this.Controls.Add(this.button2);
this.Controls.Add(this.textBox5);
this.Controls.Add(this.textBox4);
this.Controls.Add(this.label5);
this.Controls.Add(this.label4);
this.Controls.Add(this.textBox3);
this.Controls.Add(this.label3);
this.Controls.Add(this.textBox2);
this.Controls.Add(this.label2);
this.Controls.Add(this.button1);
this.Controls.Add(this.textBox1);
this.Controls.Add(this.label1);
this.Name = "Form1";
this.Text = "IRC Bot";
this.ResumeLayout(false);
this.PerformLayout();
}

#endregion

private System.Windows.Forms.Label label1;
private System.Windows.Forms.TextBox textBox1;
private System.Windows.Forms.Button button1;
private System.Windows.Forms.Label label2;
private System.Windows.Forms.TextBox textBox2;
private System.Windows.Forms.Label label3;
private System.Windows.Forms.TextBox textBox3;
private System.Windows.Forms.Label label4;
private System.Windows.Forms.Label label5;
private System.Windows.Forms.TextBox textBox4;
private System.Windows.Forms.TextBox textBox5;
private System.Windows.Forms.Button button2;
private System.Windows.Forms.TextBox textBox6;
private System.Windows.Forms.Label label6;
private System.Windows.Forms.Timer timer1;
private System.Windows.Forms.Label label7;
private System.Windows.Forms.TextBox textBox7;
private System.Windows.Forms.Label label8;
private System.Windows.Forms.Timer timer2;
private System.Windows.Forms.Label label9;

namespace SolrDataHandlingConsole
{
    class SolrObject
    {
        [SolrUniqueKey("id")]
        public string id { get; set; }
    }
}
[SolrField("rant")]
public string rant { get; set; }

[SolrField("timestamp")]
public string timestamp { get; set; }
}
class Program
{
    static void Main(string[] args)
    {
        SolrConnection solrConnection = new
            SolrConnection("http://192.168.40.135/solr");


        SolrNet.ISolrOperations<SolrObject> entry =

        SqlConnection sqlConnection = new SqlConnection("Data
            Source=MACBOOKAIR;Initial Catalog=IRC;Integrated Security=True");
        sqlConnection.Open();

        SqlCommand sqlCommand = new SqlCommand();
        sqlCommand.Connection = sqlConnection;
        sqlCommand.CommandText = "select * from ThesisIRC.dbo.Data14;";

        SqlDataReader sqlReader = sqlCommand.ExecuteReader();

        while (sqlReader.Read())
        {
        }
entry.Add(new SolrObject()
{
    id = sqlReader[0].ToString(),
    rant = sqlReader[2].ToString(),
    timestamp = sqlReader[1].ToString()
});

Console.WriteLine("Written ID: " + sqlReader[0].ToString());

SolrNet.Commands.CommitCommand Commit = new
SolrNet.Commands.CommitCommand();
    Commit.WaitFlush = null;
    Commit.WaitSearcher = true;

    string response = Commit.Execute(solrConnection);


sqlConnection.Close();

Console.ReadLine();
}