5-18-2007

Keyword Indexing and Searching for Large Forensics Targets using Distributed Computing

Sanjeeb Mishra
University of New Orleans

Keyword Indexing and Searching for Large Forensics Targets using Distributed Computing

A Thesis

Submitted to the Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirements for the degree of

Master of Science
in
Computer Science

By

Mr. Sanjeeb Mishra

BE, Bangalore University, Bangalore, India, 2000

May, 2007

ACKNOWLEDGEMENTS

The mediocre teacher tells, the good teacher explains, the superior teacher demonstrates, but the great teacher always inspires and Prof. Dr. Golden Richard III is one of the very few who always believed in his students; who tugged, pushed and lead them to the next plateau inspiring them achieve the unattainable.

People will come and go, but UNO will never see a Teacher in the form of his stature who has created a mark for himself in the student community inspiring them think big like leaders and making them better human beings in the society giving something back to their learned fraternity.

In a time where people win wars through tools he tries to win the minds of students providing them with unending freedom and bestows faith stimulating them in all possible ways.

Prof. Dr. Golden is one of the very few among our other beloved CS Dept. Professors like Prof. Dr. Adlai Depano, Prof. Dr. Chiu, Prof. Dr. Korthright, Prof. Dr. Shengru Tu & Prof. Dr. Vassil, on whom we can proudly tell:
"One looks back with appreciation to our brilliant teachers, but with gratitude to those who touched our human feelings. The curriculum is so much necessary raw material, but warmth is the vital element for the growing plant and for the soul of the child."

I have always tried to learn something from him and my Thesis on Grid Computing is his vision to make me do something big not done before by anybody.

It is a dream come true to get associated with a person of such humility who always stands by his pupils in their time of need and would like to extend my heartiest regards to him and his family.

I am grateful to Madam Janice Thomas, President, OISS, UNO; Mr. Jason Keller, Vice President, OISS, UNO & Mr. Willams, Placement Officer, for their valuable support throughout my study at University of New Orleans encouraging and supporting me when I missed deadlines in paying fees and allowing me for valuable internship experience whenever there was an opportunity. I am always indebted to Dr. Eason, Graduate School, UNO and Madam Amanda, Graduate School, UNO, for their support in allowing me for all Thesis and Graduation Proceedings whenever I missed deadlines considering my hard work and commitment towards my fulfillment of all requirements. These people are the livewires of UNO and the love, affection and care they have shown to all students throughout is exemplary which will put UNO in the forefront list where students' voices never go unheard.

As alumnus, our responsibilities won't end here but starts here to remind us to give something back to our UNO community in our lifetime which ever position wherever we are in our odyssey of life; it is a pride and honor to be associated with such lovely and

caring people of UNO and saluting the true spirit of New Orleans I admit that I have had the best time in life here at UNO. Long live University of New Orleans, New Orleans!

I would like to thank all my friends and critiques whose valuable inputs and criticism helped me hone my skills to my best and to all my juniors who always found my door to clear their doubts and put up challenges to solve their strenuous programming problems.

I am grateful to my parents and my sister whose blessings helped me come here to realize my goals and I owe them everything for their toil of many years to send me here.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

When computer forensics investigation is carried out on single workstations and the forensics image in hand is of terabytes of length, indexing of keywords takes a heavy toll. With the manifold increase of forensics data, there is an urgent need to develop a distributed digital forensics toolkit associated with sophisticated indexing and searching capabilities which utilizes multiple idle computers to index forensics images. This toolkit will deliver fast paced search performance running indexing of large forensics data and will produce smaller indexing time and faster search speeds as compared to non indexed searches on standalone workstations. The distributed resources of available computers thus make an ideal platform for virtual supercomputing performing indexing and searching on available machines.

Chapter 1. INTRODUCTION

1.1 Background and Motivation

Digital Forensics is the field of analyzing and evaluating digital data as evidence. Computer forensics is the application of computer investigation and analysis of procedures in the process of determining evidence for computer crime, misuse or theft of trade secrets, theft or destruction of intellectual property, and fraud. Computer Forensics involves an array of methodologies for discovering data that resides in a suspect's computer system or recovering encrypted, deleted, and damaged file contents which help during the discovery and investigation process [1].

The responsibilities of the people who are involved in forensics activities are enormous as they can make innocent people guilty of crimes if minor or major mistakes creep into their investigative process.

The forensics science has matured and developed significantly during the last decade and is now regarded as a science of its own standing with a solid foundation built on the acceptance of numerous certified forensic techniques and methodologies devised by scientific and law communities. The research and overall interest in computer forensics has increased manifold during the last few years, but it is still in its infancy stage.

Digital forensics practice is currently performed in many stages:

- Securing of evidence. This involves the process of producing exact replicas of seized medium, i.e., imaging. The copies must be imaged exactly and cryptographic hashing techniques are used to prove that the copy contains the exact information as the original one had. Several steps are taken carefully to identify and retrieve evidence which may exist on a suspect’s computer system. These steps protect the computer system from possible damage, corruption or virus introduction during the investigation process. This process uncovers all files such as normal files, deleted files, hidden files (password protected and encrypted files). It recovers or tries to recover all deleted files. It reveals hidden files and swap files. It accesses or tries to access the contents of all encrypted files. It also analyzes all relevant data found in special areas of the disk and prints out an overall analysis of the computer system.

- Analysis of evidence. This involves the analysis of evidence items in the large data set when it is unknown which pieces of information may have value as evidence.

A broad description of the forensics process includes the following:

- Identification – This recognizes an incident and determines its type.

- Preservation – This isolates, secures and preserves the state of physical and digital evidence.

- Collection – This records physical scene and duplicates digital evidence using established and proved methods.

- Examination – This involves a through search of evidence relating to the suspect's crime.

- Analysis – This determines facts, joins all necessary fragments of information, and draw conclusions based on evidences seized.

- Presentation – This summarizes and provides explanations of conclusions.

- Return of Evidence – This ensures that physical and other property is returned to legal owner.

- Evaluation. This involves assessing what implications the listed evidence items have in the investigation process and what the evidence tells us about the use of the computer and the actions of the user. It requires deep understanding of practices, definitions, effective use of proper forensics instruments, writing effective reports, and testifying on findings in an appropriate way based on observations and opinions derived from the course of forensics investigation. These tasks must be executed with professional excellence as per the rules and regulations of law systems.

Many methods and scientifically evaluated software packages exist for image copying and securing of digital evidence

Forensics investigation is a difficult task to handle with a single workstation or a standalone computer facility where processing power is limited by scarcity of resources, inefficient CPU power, and low level performance. With the increase of forensics data by thousands of gigabytes, it is intolerable to do digital forensics with a single workstation as it is more time-consuming. When the first hard disk drives came to market, a 5.25-inch hard disk held about five to ten Megabytes of data. Today, it is not uncommon for simple home computers to have something like two hundred Gigabytes of storage. In order to keep up with the increasing amount of forensics data, advanced systems have to be designed for expediting forensics analysis. The process of forensics investigation is mainly dependent on correct and flawless analysis of given data and, to assist investigators, this analysis should be conducted in a reasonable amount of time irrespective of the exponential growth in size of disk images.

We are going to focus on analysis of forensics images using effective indexing and searching techniques to find traces of crime evidence in a suspect's computer system. Many commercial forensics toolkits such as FTK, Sleuth Kit, and Autopsy are available in the market now. There already exist some commercial software applications for keyword indexing and searching of forensics images, e.g., dtSearch. Such applications, however, have a lot of limitations:

- They work on single workstation systems and are very slow to deal with large gigabytes of forensics targets. Their processing is limited by inefficient CPU power. They demonstrate below par performance when they try to read and search for keywords in disk images. Single workstation systems consume more time for indexing and, searching of keywords unfolds at a very slow speed. The investigator has to wait long hours to start his usual search operations and the process is intolerable when the forensics image is thousands of gigabytes of length.

- Great tools mainly useful for UNIX-like systems, "Grep" and "Strings" commands, are used for searching files for occurrence of input patterns. It becomes very tedious to search for appropriate data using "grep" and "strings" commands since one has to scroll through a large pool of redundant data to pin-point valuable information. As a standalone tool, "grep" can be cumbersome when dealing with computer forensics evidence, as there might be no actual file system, but massive chunks of data imaged from a suspects hard disk drive. The investigator has to correlate the search results to actual files and folders. This process is time-consuming and error-prone even if utmost care is taken to handle it properly.

- Lack of indexing before keyword searching is a drawback in some forensics applications when the tool searches for occurrence of a keyword within the forensics image taking considerable amount of time. Whenever keywords are searched, the whole disk image is read again for available data and, if there is a match for the specified search keyword, it is printed.

- Sleuth Kit and Autopsy toolkits fail when it comes to sophisticated text analysis for indexing and searching. dtSearch toolkit also fails when it tries to index forensics images without a file system.

- There are currently no distributed tools for indexing and searching of keywords in digital forensics. The need is to conduct analysis and investigation operations in a fast paced environment. Sleuth Kit and Autopsy do not support distributed processing. Distributed Processing consumes less time for indexing and keyword searches happen at a breathtaking speed.

- In many forensics applications which are based on keyword indexing and searching of disk images, when we search for keywords in the index file, we miss some hits as some are case sensitive and the software does not allow case sensitive searches; some are typed wrongly and cannot be searched as there is no provision in the software to make them correct; some are derived from their root words and although root words exist, these derived words cannot be searched in the index; some words are synonyms to other words and cannot be searched effectively; regular expression search is also sometimes not feasible as these searches do not support regular expressions; boolean search is also not possible with two or many keywords; Unicode foreign language search is also not possible when we deal with disk images seized in foreign countries as they are dealt in ASCII text but not in Unicode formatting.

1.2 Existing Systems and their Pitfalls

dtSearch

When a suspect's system is seized, its hard disk is imaged and all analysis and investigation are carried out on the image file for finding evidence. The dtSearch forensics toolkit operates on disk images which have a file system stored in them. When it comes to disk images without file system, it fails as it does not support them. dtSearch heads the list of forensics toolkits which employs a faster indexing and searching mechanism.

Sleuth Kit

Sleuth Kit, a command line based utility, is a formidable computer forensics tool. Although fast and efficient for smaller investigations, it is cumbersome to use when handling large sets of data.

Autopsy

Autopsy, a graphical user interface based utility, mainly used for large scale case management, adds further functionality as compared to Sleuth Kit. Autopsy correlates the search results to actual files and folders automatically which saves time and eliminates errors while handling the forensics image. Autopsy does not support sophisticated text analysis during indexing and searching.

Grep/String Commands

Mainly useful for UNIX-like systems, "Grep" and "String" commands are great tools, which are used for searching occurrence of some input patterns in files. It becomes a tedious process to search for appropriate data using "grep" or "string" commands since one has to scroll through a large pool of redundant data inside huge disk image files to pin-point valuable keywords.

Pyflag

Pyflag toolkit is another example of a keyword indexing and searching toolkit which incorporates effective index and search algorithms for it operations. Pyflag fails when the data size is of terabytes of length because it is designed for standalone systems which can handle only a few hundred gigabytes of data.

Searchtools

Searchtools toolkit is another example of keyword indexing and searching toolkit that incorporates efficient index and search algorithms for its operations. It is

designed for standalone systems which can handle only a few gigabytes of data and, therefore, fails like Pyflag when the data size is of larger length.

FTK

FTK has the best functionalities for indexing and searching of keywords, but fails when the dataset is of terabytes of length. Indexing of all possible keywords using a standalone system takes a lot of time even before the actual search starts.

Microsoft Index Server

In view of searching website contents, Microsoft came up with a version of Microsoft Index Server which is used to search website contents. For the Index Server to work, all the website files are stored into the server first. Once all the website files are stored, indexing and searching operates as usual where indexing of all website keywords are carried out first and they are searched thereafter for possible hits. Microsoft Index Server, however, works on website contents and is limited to website files only.

Google Indexing and Searching

The way Google has revolutionized the way we do search is unprecedented. The Google site indexes all websites' contents. While searching for keywords, it gives hits to those websites which can be referenced from the indexes stored in the index file: if you get a hit on a keyword, it displays all the sites which reference that word. Google is a perfect example of keyword indexing and searching, but like Microsoft Index Server it works exclusively on website contents and is limited to internet or web.

1.3 Objective

When computer forensics investigation is carried out on single workstations and the forensics image is thousands of gigabytes of length, indexing of keywords takes considerable time. To start searching for keywords, one has to wait until indexing of the entire image is completely done. This is tiresome and time-consuming on the part of the user and could be eliminated if multiple computers are utilized to index the forensics image. Indexing forensics images without the use of the distributed resources of available computers is a strenuous task for forensics investigation when one's scope is limited by scarcity of resources, insufficient CPU power, and various performance issues. With the manifold increase of forensics data, it is absolutely necessary to develop a distributed digital forensics toolkit to perform indexing and searching on large forensics targets.

The most desirable method for keyword searching is to provide a single keyword and to have the tool to search for occurrence of that keyword within the entire forensics image. This is time-consuming because the search engine needs to read the whole forensics image and then search for the keyword within it. A better way should be to index the data in the first phase and then use the index file to read the offsets of keywords within it. This allows that the next time keywords are searched, the whole disk image is not read for available data but the index is read and if there is a match for the specified keyword, it is printed. We are going to create a tool just like "grep" or "strings" commands, which is specifically used for searching forensics images for occurrence of some input patterns.

In the indexing phase, in order to limit the complexity and size of the index, restriction on the type of keywords which are indexed is required. The indexing can be set to simply index the keywords which consist of sequences of 4 or more printable characters. Although it may seem more convenient to index the entire image for all the possible keywords that may occur within it, it is extremely inefficient as most investigators are unlikely to search for arbitrary sequences of printable characters which may appear within the image. Usually investigators have a list of words and a hit on these words signifies an area of interest.

In the searching phase, the index file which is a list of offsets is read first and the indexer prints the offset of each hit with few lines or single line contexts. While searching, the source file is provided for the indexer to print some context around each hit. While searching for keywords, care should be taken to utilize stemming, fuzzy logic, boolean, and thesaurus searches as well as regular expression searches with provision for single word and multiple words searches. We will employ various search mechanisms to generate more search hits. When the search words are typed wrongly, we can use fuzzy logic to correct them to a large extent. Using stemming, we can find the root words of the search word. Using dictionary or thesaurus search, we can find all words similar in meaning to the search keyword. Case insensitive search can also be carried out if we do not need case sensitive search. Regular expression search can be provided to implement regular expression searches and appropriate patterns in the index file can be found out. Boolean search using operators "or" and "and" can be possible among search keywords.

Single and multiple word searches can also be employed so that single words as well as multiple words can be searched sequentially.

The main objective is to make the index time faster and not only the search time but the regular expression search time also is to be made faster with the introduction of effective algorithms during indexing and searching. Search time has to be less than 1 sec and, the regular expression search time, although it depends upon the dataset size, should be made minimum only to a few seconds probably less than a minute even for terabytes of dataset.

The distributed digital forensics toolkit should be comparable to dtSearch, the industry rated software toolkit for indexing and searching and should eliminate all bad features found in contemporary forensics toolkits such as Autopsy and Sleuth kit. Autopsy and Sleuth kit work on non-distributive environments and make index and search processes time-consuming. To make indexing considerably faster, there is no better way than the distributed framework which indexes and searches large datasets.

Our forensics toolkit should index Forensics Images (Terabytes of Length), not simple .html/.htm/.txt or other text file formats.

Even in scenarios when the index file grows bigger, using index file customization index can be made comparably shorter. This makes data retrieval faster.

1.4 My Different Approach

The Distributed Digital Forensics Application will consist of two parts: the Distributed Framework and the Forensics Application.

In view of department security and intranet based development work, Microsoft .NET Remoting will be chosen to do the bulk of distributed programming work, which will use idle CPU computers and other Beowulf cluster nodes for faster performance. Choice of .NET Remoting over other available distributed technologies is that .NET Remoting is reliable, flexible, easy to use, and suitable for enterprise network computing. .NET Remoting is a generic method of inter-process and remote communication available in Microsoft .NET and is the backbone of our distributed framework.

The Forensics Application will be implemented using Microsoft C# language which is upper compatible to C, C++, and Java programming languages. The whole forensics application will be written in Visual C# using Visual Studio.NET 2005. C# Windows Forms will be used to create the windows forms and modules will be developed using C# 2.0 on VS.NET 2005. C# Generics will be used which is a part of VS.NET 2005 implementation and was not a part of earlier VS.NET 2003 implementation. .NET Framework 2.0, which is default with VS.NET 2005, will be the platform on which our application will be built. Choice of Microsoft C# language over C/C++/Java programming languages is due to its advanced object oriented capabilities, greater user interface design using Windows Forms and Web Forms, enhanced capabilities for Generics, ease of network programming, rapid application development, and faster web services development time.

1.5 Overview

This Master's Thesis is organized into eight chapters as follows:

Chapter 1 includes background, motivation, thesis objective, and thesis outline.

Chapter 2 describes the design and implementation of the distributed framework.

Chapter 3 explains the design and implementation of the forensics application.

Chapter 4 mentions the results and the performance figures for different data sizes.

Chapter 5 shows the conclusions made out from Master’s Thesis work.

Chapter 6 discusses the avenues for future work and further discussions.

Chapter 7 cites the list of references used.

Chapter 2. DISTRIBUTED FRAMEWORK TO DEVELOP

2.1 Overview

The need of distributed processing for keyword indexing and searching is enormous. Many forensics applications do not have distributed processing. Distributed Processing does not consume much time for indexing and keyword searches happen at a breathtaking speed.

Distributed Computing is defined as a way to break down a single problem into multiple pieces that can be worked on independently by multiple machines. Distributed computing, often described as grid computing, parallelism or clustering, is not suitable for all types of tasks because application environments vary depending upon the requirement. Grid computing deals with heterogeneous systems, parallelism deals with an application being divided and performed at its participating peers, while clustering works with homogeneous systems. An operation that takes a short amount of time may be lengthened by the overhead needed to send messages to other participating computers in the network and extra work is needed to divide the problem into multiple parts and reassemble the result. This extra work is in fact an overhead on part of the running application. It also raises the complexity of the system to different levels, creates complex issues which are not easily encountered in a single process system, and should be kept to a minimum.

So the need of the hour is to come up with a Distributed Framework which taps the CPU power of all available computers, but does not need much interaction across the network with other computers. The computers will run independently to perform their assigned task and messages will be exchanged between them while starting or completing a task.

2.2 Distributed Framework Goals

The Distributed Framework has to follow certain programming guidelines to ensure it that it will be easy to handle and maintain once it is built. The features to be considered are as follows:

Ease of Programming: The Distributed Framework will be developed using a high level programming language that will handle many critical language, networking, and future technology issues. The framework has to be designed using the latest technologies and should be upper compatible to existing languages. The framework should support many advanced and evolving technologies such as security, threading, web services, and remote networking. Although C/C++/Java programming languages have many existing features, to design and implement a whole Distributed Framework with all the latest state of the art technologies, we have to choose a language which not only solves the problem of complex distributed programming, but should also give us features for its maintenance as well as high productivity in the future. Hence, a suitable choice of a programming

language with its associated networking technologies is the need of the hour to develop our desired Distributed Framework solution.

Platform Independence: Different OS systems can be connected to the distributed framework. For Linux machines to talk to Windows systems, Linux systems can be added having VMware installed on them and guest OS should be Windows so that any number windows machines as well as any number of Linux machines can be added to the network.

Robustness: Tool should raise exceptions and can easily be handled.

Extensibility: Must allow us to add new features and remove existing features with ease.

Interactivity: Server and Clients have to be interactive.

Testing: The indexing and searching application will be tested on a cluster of machines at NSSAL Lab, Department of Computer Science, University of New Orleans, which consists of 10 Windows XP Pro 3.0 GH Multithreading Intel Xeon High Speed Processors and a Beowulf Cluster of 72 nodes for high performance computing. They are used primarily for building and testing high performance distributed systems.

2.3 Use of Microsoft C# .NET for Application Development

- Rapid Application Development as compared to other languages such as C/C++/Java
- Object Oriented Programming in C#
- All phases of Product Development are made easy using Microsoft .NET Tools
- Suitable for Enterprise Application Development
- User Interface Design using C# Windows Forms is quite easy
- Use of C# Generics while Indexing
- Networking is made easy using .NET Remoting
- Future Goals in Mind (XML Web Services)

2.4 Use of Microsoft .NET Remoting for Distributed Framework Development

.NET Remoting will be best suited for this sort of distributed application because we are creating a distributed supercomputer that taps CPU power of all available computers. This distributed application supports the Remoting architecture as it does not need much interaction across the network with other client computers. Clients will run independently to perform their assigned work and messages will be exchanged between the available computers while starting or completing a task. Although this suits .NET Remoting, the tedious task lies in coding the communication protocols and the message format between clients or workers since a lot of issues have to be dealt while coding for them.

- Does not need much interaction across the network, messages are just exchanged, and bulk of the work is done at the client or worker computers.

- Remoting is a generic method of inter-process and remote communication in .NET.
- Allows applications in different processes and computers to communicate seamlessly and is designed to use the objects in another application the same way one uses the objects.
- It can be used with various protocols and in various cross-platform environments.
- Easy to configure, is reliable, and maintains a high level of abstraction with networking code.
- .NET Framework CLR takes care of connections, objects creation, and destruction.
- .NET Remoting deals with how objects communicate out-of-process, how to serialize data that must be sent across the network, how to handle concurrent access, bidirectional communication, callbacks and events, and object lifetime.
- It handles state management and object lifetime to ensure that objects time out when the client is not using them and, thus, implements security by safeguarding the system from hacking and other malicious attacks.
- In Remoting, we have the choice of different activation types, transport protocols, serialization formats, and object-lifetime policies. These choices can be carried out with few lines of code changes in the configuration file, but the code for using the remote object remains unchanged.
- Remoting supports many types of objects such single client or multiple clients.

2.5 Basics of Microsoft .NET Remoting

2.5.1 What is .NET Remoting?

Remoting is flexible, reliable, easy to configure, and a high-level abstraction that wraps networking code. With Remoting, the common language runtime (CLR) in .NET Framework takes care of releasing unneeded objects, creating and closing connections, and managing requests with a pool of threads. Remoting works better for brokered communication with a coordination server than for decentralized peer-to-peer applications.

.NET Remoting deals with how objects communicate out-of-process, how to serialize data that must be sent across the network, how to handle concurrent access, bidirectional communication, callbacks and events, and object lifetime issues.

Remoting is a generic method of inter-process and remote communication in .NET. It allows applications in different processes and different computers to communicate seamlessly and is designed to use the objects in another application the same way we use the objects.

In Remoting, we have the choice of different activation types, transport protocols, serialization formats, and object-lifetime policies. One can change these options with few lines of code in the configuration file but the code for using the remote object remains unchanged.

2.5.2 Inter-Process Communication

To bridge the gap between more than one application domains, .NET Remoting is used.

Remoting is often described as the way programs communicate with each other in .NET. There are literally dozens of different ways for applications to communicate on any platform. Some of the options for inter-process communication include: serialization of information to a data store that any applications can access, sending a message to a Microsoft Message queue, calling an ASP.NET web service with a SOAP message, creating a connection by using .NET's networking support which provides classes that wrap Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) channels or raw sockets, and using services based on COM/DCOM.

2.5.3 Advantages of .NET Remoting:

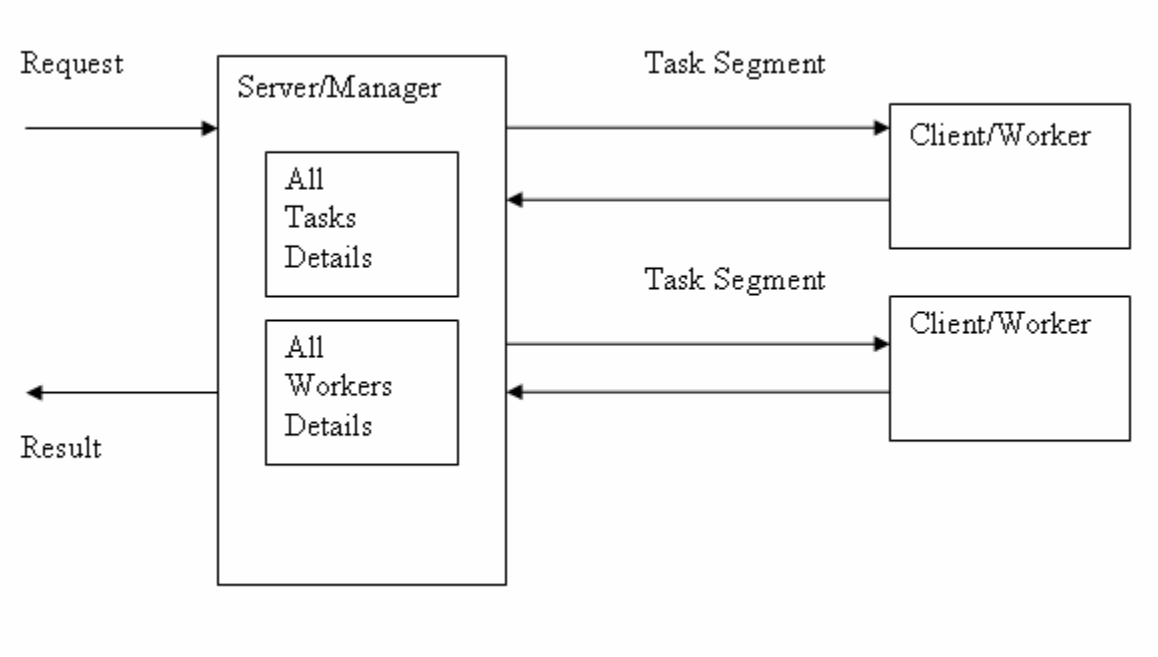
Remoting is suited for any type of distributed application. It can be used with various protocols and in various cross-platform applications. It handles state management and object lifetime to ensure that objects time out when the client is not using them. It is extensible. It allows one to use Secure Sockets Layer (SSL) security to encrypt messages when used with Internet Information Server (IIS).

Remoting supports many types of objects including Client-Activated, SingleCall, and Singleton objects. Singleton objects are the most complex types of objects because they deal with scenarios where multiple clients may use them at once.

2.5.4 Disadvantages of .NET Remoting

Remoting sends objects in all or nothing chunks. If one needs to stream large files across the network, Remoting may not be the most suitable approach.

2.6 Distributed Framework Architecture and Design



2.1: Distributed Framework Architecture

Here we are creating a distributed-computing framework which breaks down an individual task into multiple pieces and reassembles the results afterwards. In order to reduce the time taken to complete the original task, parallelism methodology is utilized by dividing each task into multiple pieces. Speed and overall efficiency have been achieved at the expense of lot of complex distributed programming.

The distributed system is designed to find a solution for our indexing and searching task: the indexing of large gigabytes of digital forensics data and searching these indexed contents for possible keywords. For maximum speed and efficiency, the system uses multiple workers and assembles their results on a work manager or a server. Here the workers are described as clients and the work manager acts as a server which co-ordinates all clients' interactions.

The whole application is written in Visual C# using Visual Studio.NET 2005. .NET Remoting (VS.NET 2005) is used for distributed application development. C# Windows Forms has been used to create all user friendly Windows Forms and all code modules have been developed using C# 2.0 on VS.NET 2005. VS.NET 2005 was officially released in 2005. .NET Framework 2.0 comes with VS.NET 2005 by default.

2.6.1 Individual Units of Distributed Framework

Server/Manager

- Distributor-Assembler (Distributes Task, Assembles them)
- Requester-Assembler (Requests Task, Assembles them)

Client/Worker

- Requester (Requests Task)
- Worker (Works on Task)

Task/Work

Customizable to any Distributed Application

Display/Notification

Results Display/Output

2.6.2 Tasks for Individual Units

Server/Manager

App.Config: Configuration Settings for Task Server

Server Startup: Main Form for Task Server and loads when the Task Server starts

Server Methods: Task Submission for Indexing

Task Submission for Searching

Task Complete for Indexing using one way Remoting

Task Complete for Searching using one way Remoting

Adding Worker

Removing Worker

Server Task:Worker Record which records workers history/details

Join Results for Indexing

Join Results for Searching

Client/Worker

App.Config: Configuration Settings for Task Worker

Worker Startup: Main Form for Task Worker and loads when Task Worker starts

MainForm Methods: Index Options for Indexing Task and Index Button Press

Search Options for Search Task and Search Button Press

Worker Methods: Login for Client

Logout for Client

Receive Results for Indexing One way Remoting

Receive Results for Searching One way Remoting

Index Task Submission to Task Manager

Search Task Submission to Task Manager

Task/Work

Task (Class/Interface) TaskRequest, TaskSegment and TaskResults Classes and Interfaces for both Indexing and Searching

Data Structures: Implementation of Complex Data Structures for Indexing

Task (Index/Search): Index and Search Method Implementation

Display/Notification

DisplayForm:Display Form to be invoked by Task Server while Startup

DisplayInterface:Interface

2.6.3 Steps of Execution in the Distributed Framework

- Server/Manager

App.Config: Configuration Settings for Task Server

- Display/Notification

Display Form:Display Form to be invoked by Task Server while Startup

Display Interface:Interface

- Server/Manager

Server Startup: Main Form for Task Server (Loads when the Task Server starts)

- Worker/Client

App.Config: Configuration Settings for Task Worker

Worker Startup: Main Form for Task Worker (Loads when Task Worker starts)

MainForm Methods: Index Options for Indexing Task (Index Button is pressed)

MainForm Methods: Search Options for Search Task (Search Button is pressed)

- Worker/Client

Worker Methods: Login for Client

Index Task Submission to Task Manager

Search Task Submission to Task Manager

- Server/Manager

Server Methods: Adding Worker

Server Task:Worker Record (Records workers history/details)

Server Methods: Task Submission for Indexing

Task Submission for Searching

- Task/Work

Task(Class/Interface):TaskRequest, TaskSegment and TaskResults Classes and Interfaces for both Indexing and Searching

Task (Index/Search): Index and Search Method Implementation

Data Structures: Implementation of Complex Data Structures for Indexing

- Server/Manager

  Server Methods: Task Complete for Indexing using one way Remoting

Server Methods: Task Complete for Searching using one way Remoting

- Server/Manager

Server Task: Join Results for Indexing

Server Task: Join Results for Searching

- Worker/Client

Worker Methods: Receive Results for Indexing One way Remoting

Worker Methods: Receive Results for Searching One way Remoting

- Worker/Client

Worker Methods: Logout for Client

- Server/Manager

Server Methods: Removing Worker

2.7 Design and Implementation

The major units of Distributed Framework are as follows:

- Task Component (Task)
- Task Manager (Server)
- Task Worker (Client)

2.7.1 Task Component

The objects which are used to transmit messages between the server and available clients and the interfaces they are exposed to are described here.

- Task Component Interfaces

  This describes all interfaces used in the application. Individual interfaces exist for the server, the client, and the requester/coordinator computers.

  The Server interface ITaskServer is used for defining methods for registering and un-registering client computers, for receiving task request, and task complete notification.

  The Client interface ITaskWorker is used for defining a method for receiving the task assignment.

  The Requester interface ITaskRequesterIndex is used for defining a method for receiving the final index task results.

  The Requester interface ITaskRequesterSearch is used for defining a method for receiving the final search task results.

  For a client to request and perform task simultaneously, the client and the requester/coordinator interfaces will be implemented by the same application.

- Task Message Objects

  The objects that are created from the time index request is submitted to the time index request is completed are as follows:

  - TaskRequestIndex

  This identifies initial task parameters for indexing.

  - TaskSegmentIndex

    This identifies task parameters for a portion of the indexing task.

  - TaskResultsIndex

    This contains the aggregated index results from all task segments which are delivered to the client that made the initial request.

The objects that are created from the time search request is submitted to the time search request is completed are as follows:

- TaskRequestSearch

This identifies initial task parameters for searching.

- TaskSegmentSearch

This identifies task parameters for a portion of the search task.

- TaskResultsSearch

This contains the aggregated search results from all task segments which are delivered to the client that made the initial request.

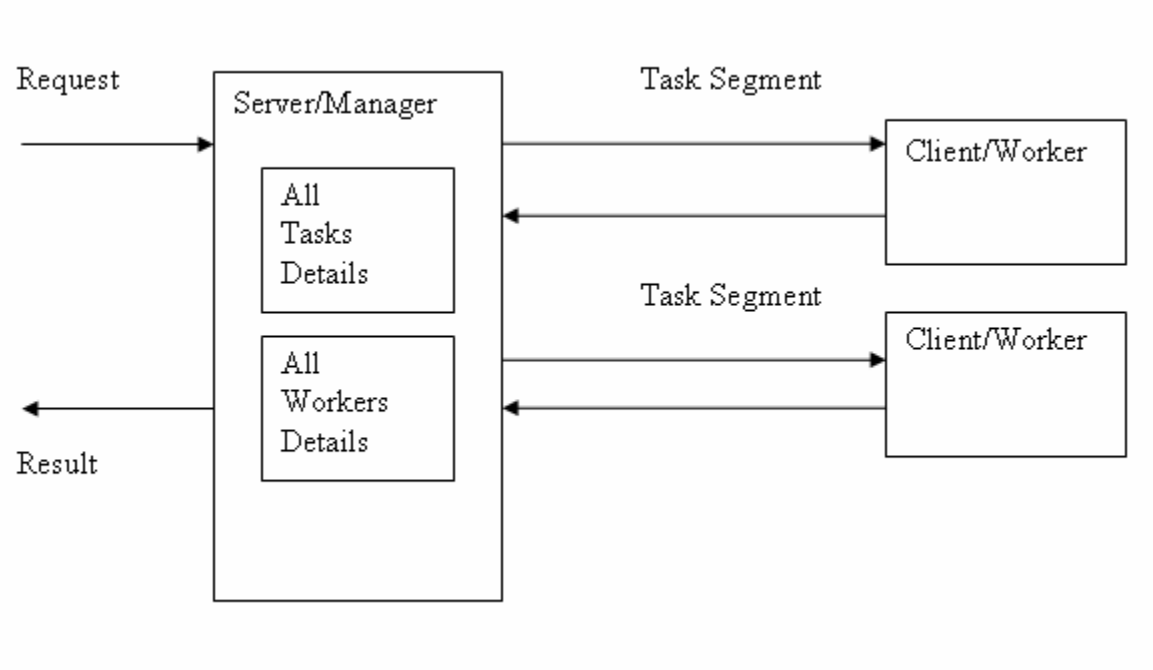
- Task Logic

Task Logic describes the index and search components which are used for carrying out the indexing and searching operations.

2.7.1.1 Task Component Interfaces

Three interfaces are required: a client/worker, a server/manager, and a task requester for each index and search phase. The client/worker and requester interfaces will be implemented by the same application so that all available clients can perform and request work.

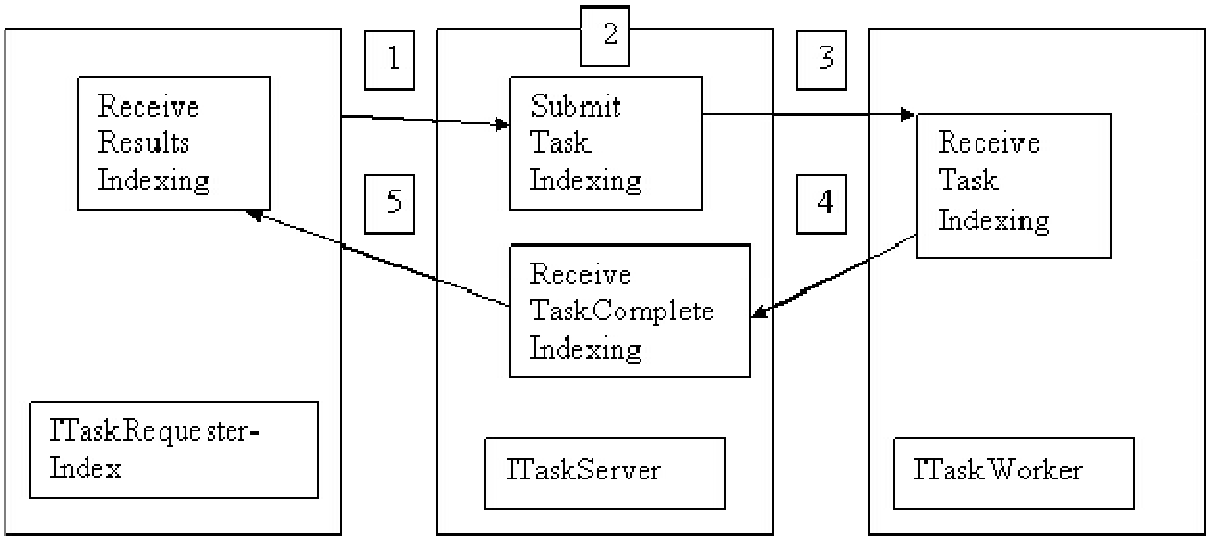
Message passing in the Server/Manager-Client/Worker Distributed System is performed as follows:



2:2: Message Passing between Worker and Manager

The objects for indexing are TaskRequestIndex, TaskSegmentIndex and TaskResultsIndex. Steps that occur from the time an index request is submitted to the completion of the index request are as follows:

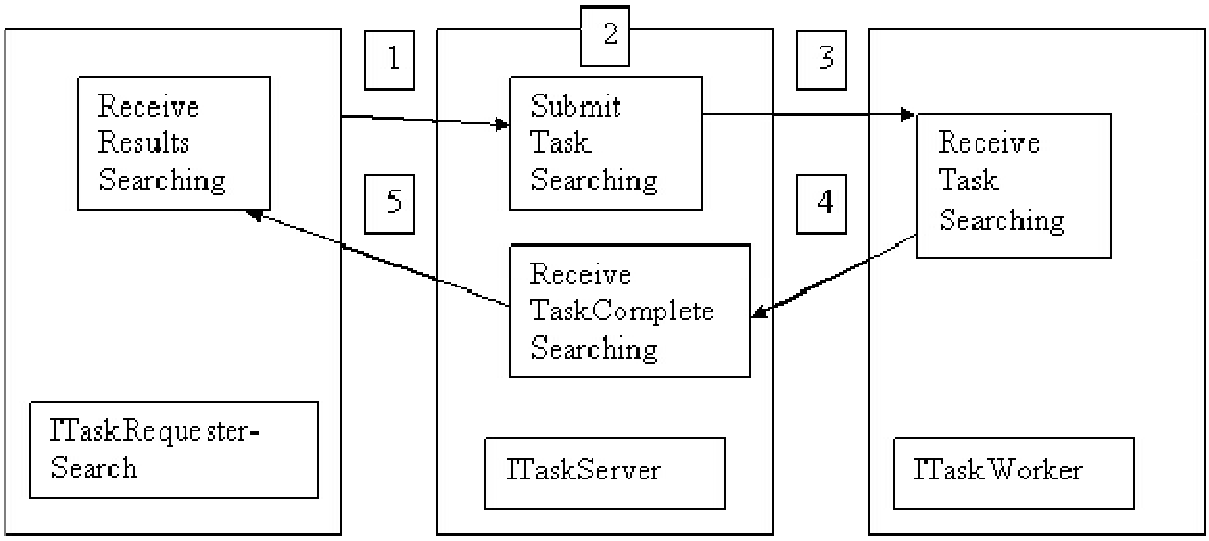
- Server/Manager receives TaskRequestIndex object.
- Server/Manager stores Task object internally in a collection.
- Server/Manager divides the work into segments and sends available workers/clients TaskSegmentIndex object with a part of the work.
- When workers/clients finish, they send back the TaskSegmentIndex with the result information added.
- When all task segments are received, server/manager compiles the information into TaskResultsIndex object and sends it to the client.



2.3: Steps for Index Task

The objects for searching are TaskRequestSearch, TaskSegmentSearch and TaskResultsSearch. The steps that occur from the time a search request is submitted to the completion of the search request are as follows:

- Server/Manager receives TaskRequestSearch object.
- Server/Manager stores Task object internally in a collection.
- Server/Manager divides the work into segments and sends available workers/clients TaskSegmentSearch object with a part of the work.
- When workers/clients finish, they send back the TaskSegmentSearch with the result information added.
- When all task segments are received, server/manager compiles the information into TaskResultsSearch object and sends it to the client.



2.4: Steps for Search Task

Methods which allow users to be registered and unregistered with the server are:

Guid AddWorker(ITaskWorker callback); //Add Worker

void RemoveWorker(Guid workerID); //Remove Worker

Methods used to register a task are as follows:

Guid SubmitTaskIndex(TaskRequestIndex taskRequestIndex; //For Indexing

Guid SubmitTaskSearch(TaskRequestSearch taskRequestSearch); //For Searching

Methods used by the server to submit a task to a client are:

void ReceiveTaskIndex (TaskSegmentIndex taskIndex); //For Indexing

void ReceiveTaskSearch(TaskSegmentSearch taskSearch); //For Searching

Methods which are invoked to send task complete notification are:

void ReceiveTaskCompleteIndex (TaskSegmentIndex taskSegmentIndex, Guid workerID); //For Indexing

voidReceiveTaskCompleteSearch(TaskSegmentSearch taskSegmentSearch, Guid workerID); //For Searching

ITaskServer interface defines the methods for registering and un-registering clients, for receiving task request, and task-completed notification.

ITaskWorker interface defines a single method for receiving task assignment.

ITaskRequesterIndex defines a single method for receiving final index task results.

ITaskRequesterSearch defines a single method for receiving final search task results.

2.7.1.2 Task Message Objects

Objects for Indexing:

- TaskRequestIndex identifies initial task parameters for indexing

- TaskSegmentIndex identifies task parameters for a portion of the index task and the index task result for that segment once it is complete.

- TaskResultsIndex contains the aggregated index results from all task segments which are delivered to the client that made the initial request.

Objects for Searching:

- TaskRequestSearch identifies initial task parameters for searching

- TaskSegmentSearch identifies task parameters for a portion of the search task and the search task result for that segment once it is complete.

- TaskResultsSearch contains aggregated search results from all task segments which are delivered to the client that made the initial request.

TaskRequestIndex, TaskSegmentIndex, TaskResultsIndex, TaskRequestSearch, TaskSegmentSearch and TaskResultsSearch classes are task specific. The server uses a Task object to store information about requested and in-progress tasks. The TaskRequestIndex, TaskSegmentIndex, TaskResultsIndex, TaskRequestSearch, TaskSegmentSearch and TaskResultsSearch classes are all necessary parts of the interface between the remotable components. These classes are called Message objects since they are used mainly for message passing between components.

The TaskRequestIndex and TaskRequestSearch classes indicate ITaskRequesterIndex and ITaskRequesterSearch interfaces that they should be notified when the indexing and searching task is completed.

The TaskSegmentIndex and TaskSegmentSearch class resembles TaskRequestIndex and TaskRequestSearch classes respectively with few new additional features. TaskSegmentIndex and TaskSegmentSearch classes store a TaskID and a SequenceNumber. The SequenceNumber is used to reassemble segments to ensure that results are ordered properly. The TaskSegmentIndex and TaskSegmentSearch classes identify GUID of workers who assigned the task. String array is used to hold results when TaskSegmentIndex and TaskSegmentSearch classes send results back to the server.

The TaskResultsIndex and TaskResultsSearch classes store the index and search result respectively inside a string array.

2.7.1.3 Task Logic

Task Logic for Indexing and Searching has been described in the next section for design and implementation of the forensics application.

2.7.2 Work Manager

- Work Manager

  Work Manager acts as a server and takes care of all client interactions, divides the specified task into parts, and assembles the result. Server Startup Form is displayed when the server starts. Manager stores the collection of Workers and Tasks in its memory. It creates hash table data structures to store these collections.

- Tracking Workers

  Server provides add and remove methods that allows clients to register themselves in the Workers collection and allows clients to remove them from the same collection.

  - Add Worker

    Server provides a method that allows clients to register themselves in the Workers collection

  - Remove Worker

    Server provides a method that allows clients to remove themselves from the collection.

- Tasks

  This gives complete information about a new task and contains methods for joining task results from individual clients/worker computers through their sequence numbers.

When the server receives a task request, a new Task is created. This task request stores the original task, along with additional information including ID information which the task generates, the collection that contains workers information that are processing the segments of this task, and a hash table with an entry for each task segment result specified by sequence numbers.

- Task for Joining Index Results

  This describes how index results from various workers are joined and the final index result is calculated.

- Task for Joining Search Results

  This describes how search results from various workers are joined and the final search result is calculated.

- Dispatching Tasks

  Manager/Server receives a task request; breaks it into multiple segments, and assigns it to available workers. This phase is used for dispatching tasks to available client/worker computers.

  - Submit Task Index

    Manager/Server receives an indexing task request, breaks it into multiple segments, and assigns it to available workers.

  - Submit Task Search

    Manager/Server receives a search task request, breaks it into multiple segments, and assigns it to available workers.

- Completing Tasks

  Manager/Server receives completed task segment objects, adds them to the corresponding task (Tasks collection), and marks the worker as available after the worker finishes its assigned task.

  - Receive Task Complete Index Remoting

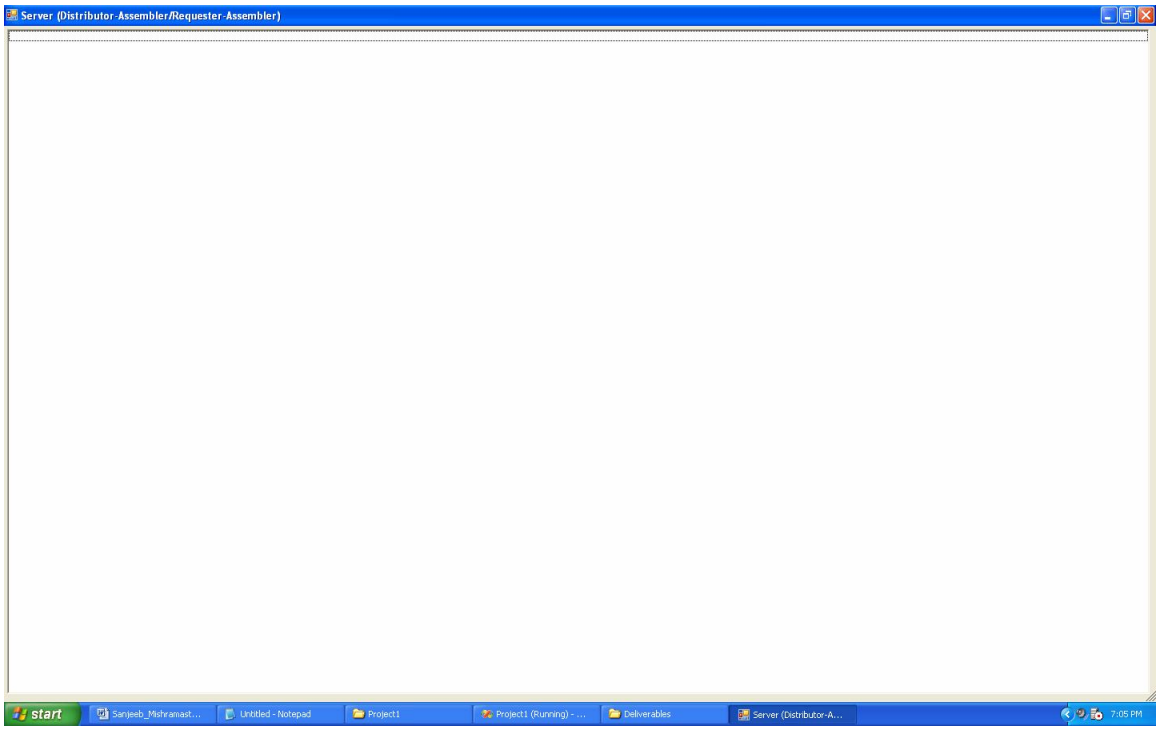
    Manager/Server receives completed index task segment objects, adds them to the corresponding indexing task (Tasks collection), and marks the worker as available after the worker finishes its assigned indexing task.

After this, index results from various workers are joined and the final index result is calculated as described in “Task for Joining Index Results”.

- Receive Task Complete Search Remoting

  Manager/Server receives completed search task segment objects, adds them to the corresponding searching task (Tasks collection), and marks the worker as available after the worker finishes its assigned search task. After this, search results from various workers are joined and the final search result is calculated as described in “Task for Joining Search Results”.

2.7.2.1 Work Manager



2.5: Server Startup Form

Work Manager is the Task Manager which acts as a server and takes care of all client interactions, divides the distributed application into parts and assembles the result. Server Startup Form (Fig 2.3) is displayed when the server starts. Work manager/server stores the collections of Workers and Tasks in its memory. It creates hash table data structures to store these collections as shown below:

```
private Hashtable Workers = new Hashtable();
private Hashtable Tasks = new Hashtable();
```

Tasks collection holds a collection of Task objects which represent the ongoing tasks. Objects in the Tasks collection are indexed by TaskID. The Workers collection keeps track of information about all registered clients/workers using WorkerRecord objects. WorkRecord objects are indexed by the WorkerID.

Using one setting in the configuration file, an integer that sets the maximum number of workers in the system can be assigned to a task which ensures that other available workers will be free to serve new incoming requests. This in turn prevents a task from being divided into so many pieces that the communication time becomes an extra overhead.

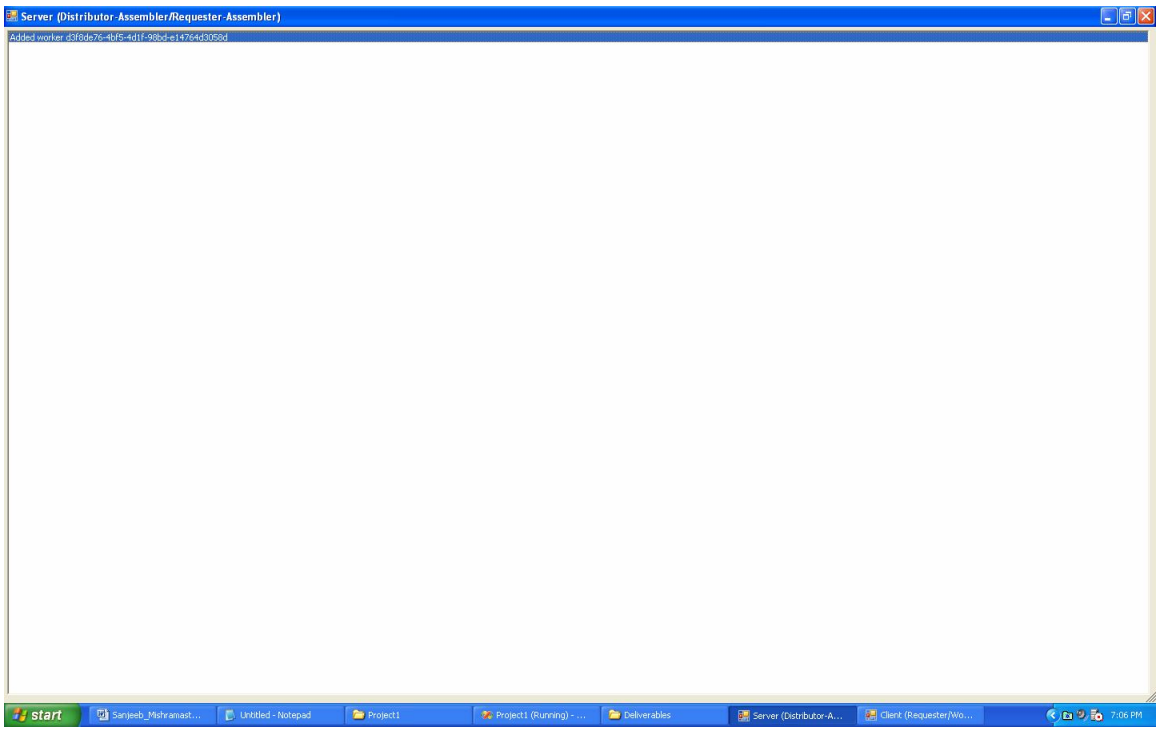
To display all clients' interactions as they happen, a window with a message that indicates what actions the server has performed, is loaded when the manager starts.

2.7.2.2 Tracking Workers

Server provides an AddWorker() method that allows clients to register themselves in the Workers collection which is a collection of all available workers. It also provides RemoveWorker() method that allows clients to remove themselves from the collection.

2.7.2.2.1 Add Worker

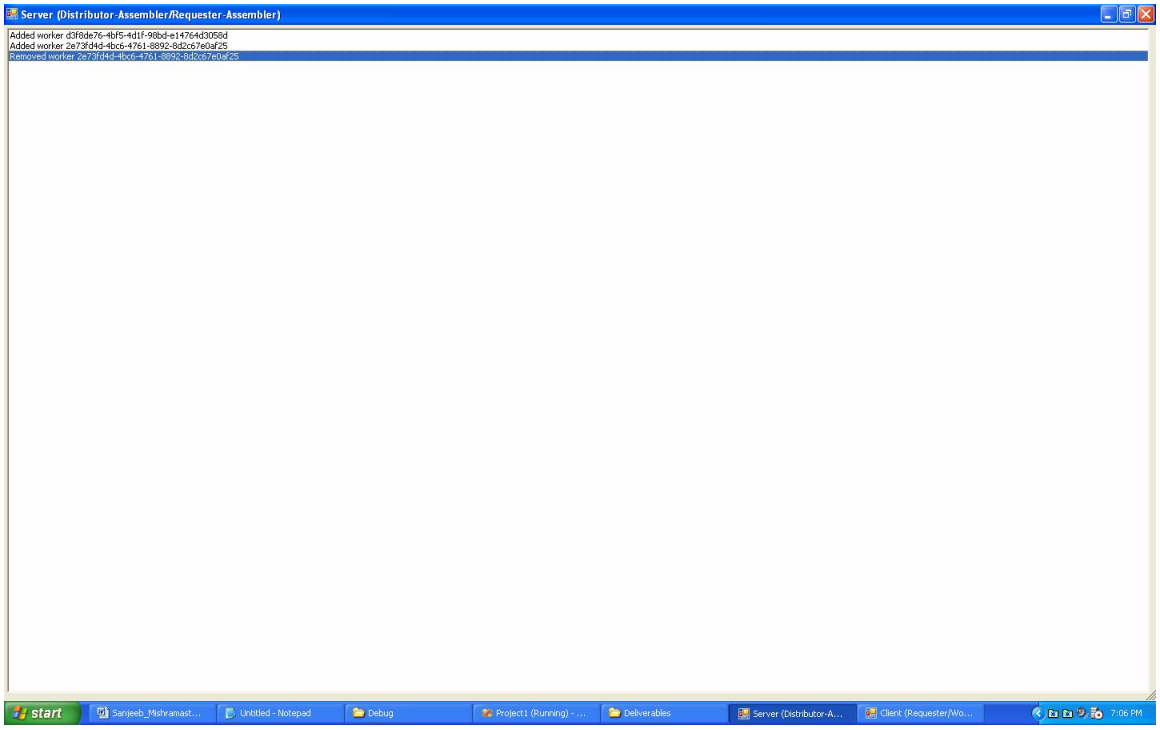
The AddWorker() method, provided by the Server, allows clients to register themselves in the Workers collection. The below figure shows how a worker is added to the available list of workers collection.



2.6: Add Worker

2.7.2.2.2 Remove Worker

The RemoveWorker() method allows clients to remove workers from the Workers collection. The below figure shows how a worker is removed from the available list of workers collection.



2.7: Remove Worker

RemoveWorker() method makes sure that the worker has finished all its tasks before exiting and a worker cannot exit before finishing its pending task. One, however, can make the worker exit the application deterministically by closing the client/worker application.

Workers are stored as WorkerRecord objects and to identify workers uniquely on a network, each worker has a globally unique identifier (GUID) which is generated automatically when the WorkerRecord class is instantiated. This is primarily used for not assigning them any preexisting names as they are only known by their GUID values.

A worker can be assigned at most one task at a time and not more than that because application exception will halt the worker to proceed with many tasks since it can handle only one task at a time.

2.7.2.3 Tasks

When server receives TaskRequestIndex, a new index Task object is created. The TaskRequestIndex stores the original index Task, along with additional information including GUID information which Task class generates automatically, the collection that contains workers information that are processing the segments of this task, and a hash table with an entry for each

TaskSegmentIndex result indexed by sequence numbers. In the same way, a new search Task object is created when the server receives a TaskRequestSearch.

Task class contains GetJoinedResultsIndex() and GetJoinedResultsSearch() methods. It combines all results into a large array, which is returned to the client. Each entry in the hash table is an array of strings that represents the solution for part of the original requested range. The entries in the hash table are indexed by their sequence numbers and they can be reassembled in order by starting with sequence number 0 which is the first sequence number and so on, regardless of the order in which the results were received.

2.7.2.3.1 Task for Joining Index Results

This describes how index results from various workers are joined and the final index result is calculated.

2.7.2.3.2 Task for Joining Search Results

This describes how search results from various workers are joined and the final search result is calculated.

2.7.2.4 Dispatching Tasks

While indexing, Manager/Server execution takes place in the SubmitTaskIndex() method. The SubmitTaskIndex() method receives an index task request, breaks it into multiple segments, and assigns it to available workers. When Manager/Server execution takes place in the SubmitTaskSearch() method during searching, the SubmitTaskSearch() method receives a search task request, breaks it into multiple segments, and assigns it to available workers.

It uses maximum workers allowed by MaxWorkers. A new Task object is created as follows:

```
Task Task = new Task(taskRequest);
```

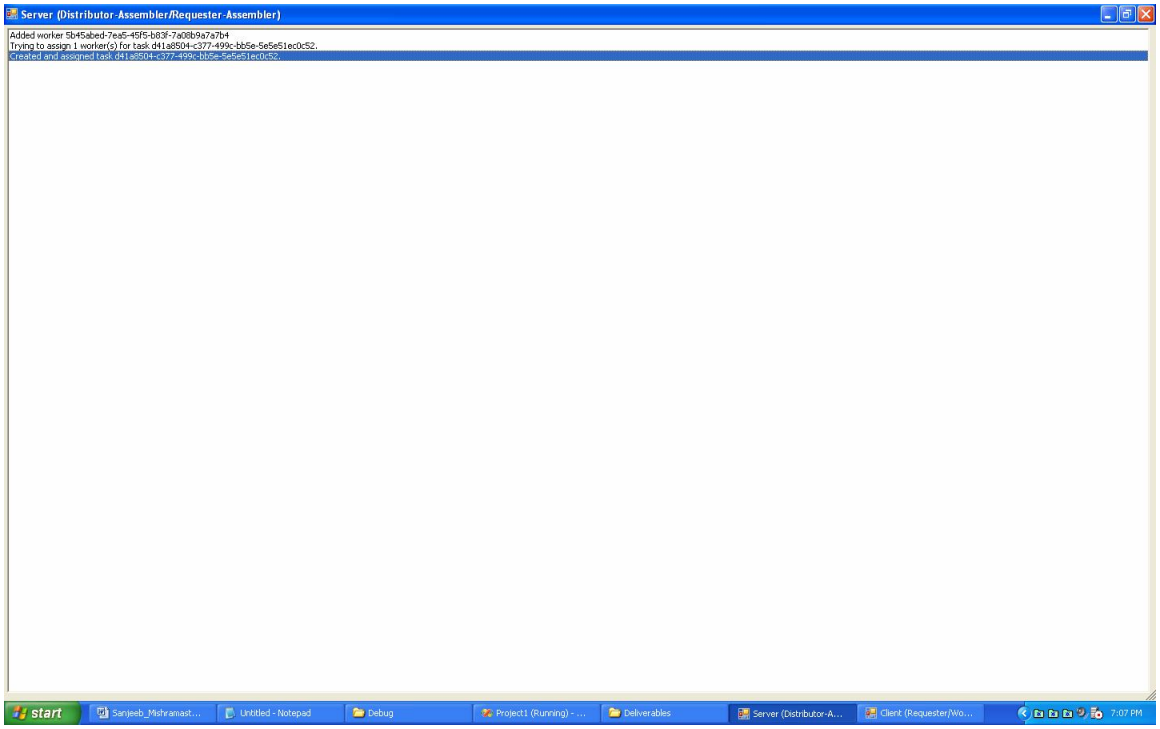
Search for available workers is carried out. All available workers are used. All workers including the worker making the task request are taken into consideration and are placed inside the workers collection. All participating workers are marked as assigned for the same task.

To ensure that there is at least one worker, proper error checking is performed otherwise an exception will be thrown.

The task of dividing the assigned work into segments is straightforward. Once the segment is constructed, it is sent asynchronously to the worker by calling the

worker's ReceiveTaskIndex() method for indexing and ReceiveTaskSearch() method for searching. At last, the Task object is stored in the Tasks collection.

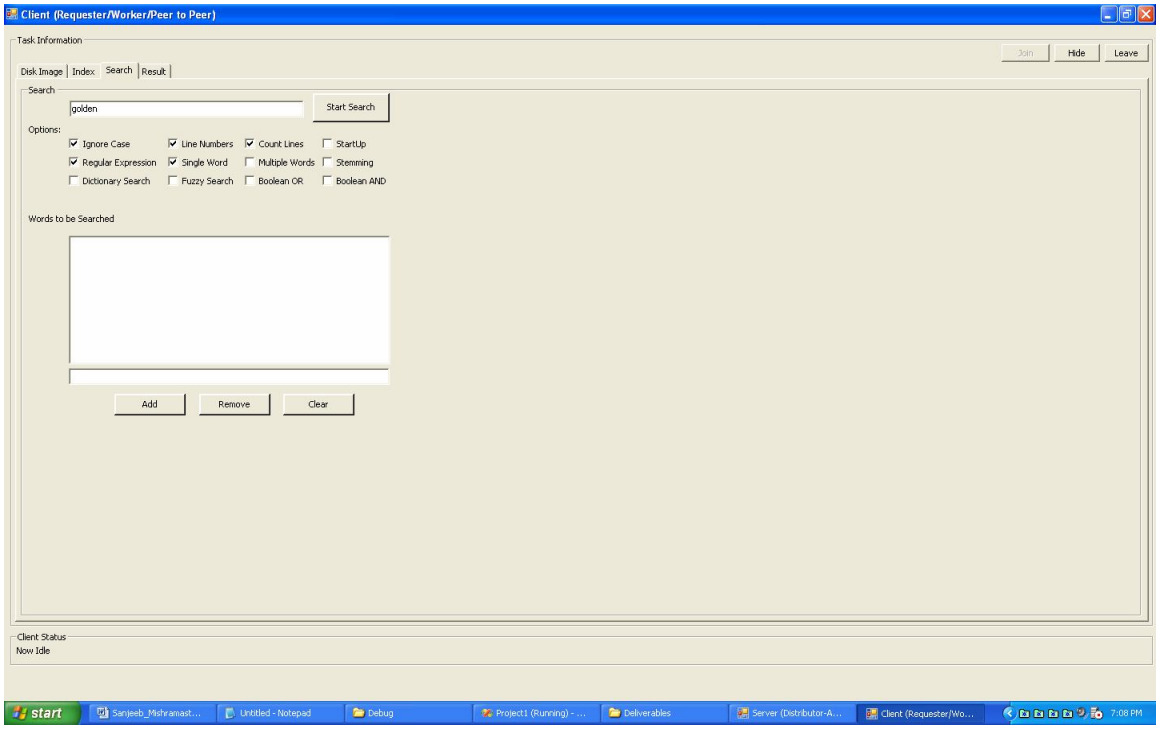
2.7.2.4.1 Submit Task for Indexing



2.8: Index Task Assigned

The above figure shows how index task is submitted to the available workers.

2.7.2.4.2 Submit Task for Searching



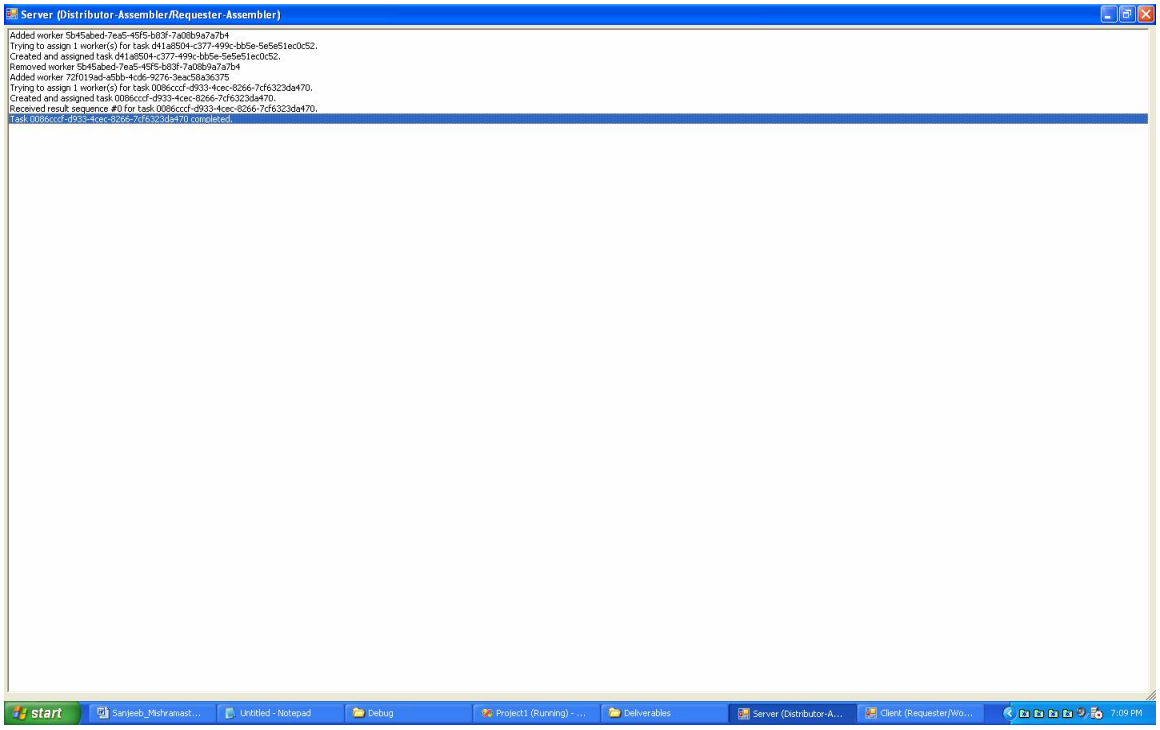
2.9: Search for a string "golden"

The above figure shows how search task is submitted from the server to the available workers.

2.7.2.5 Completing Tasks

Work Manager's ReceiveTaskCompleteIndex() and ReceiveTaskCompleteSearch() methods for indexing and searching receive completed TaskSegmentIndex and TaskSegmentSearch objects, add them to the corresponding Task (Tasks Collection), and mark the worker as available since the worker finishes its assigned task. When the number of received results equals the number of task segments, the task is declared complete and a message is sent to the task requester with the list of results and the task is removed from the memory collection.

2.7.2.5.1 Receive Task Complete for Indexing

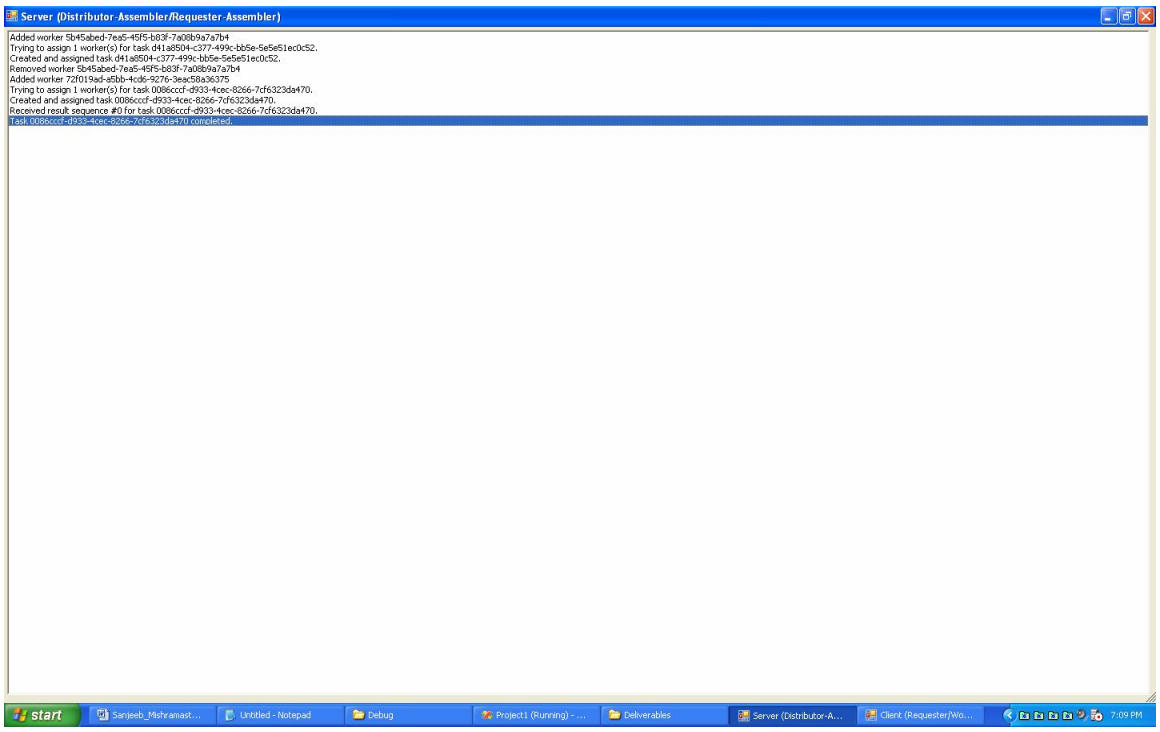


2.11.1: Index Complete Notification

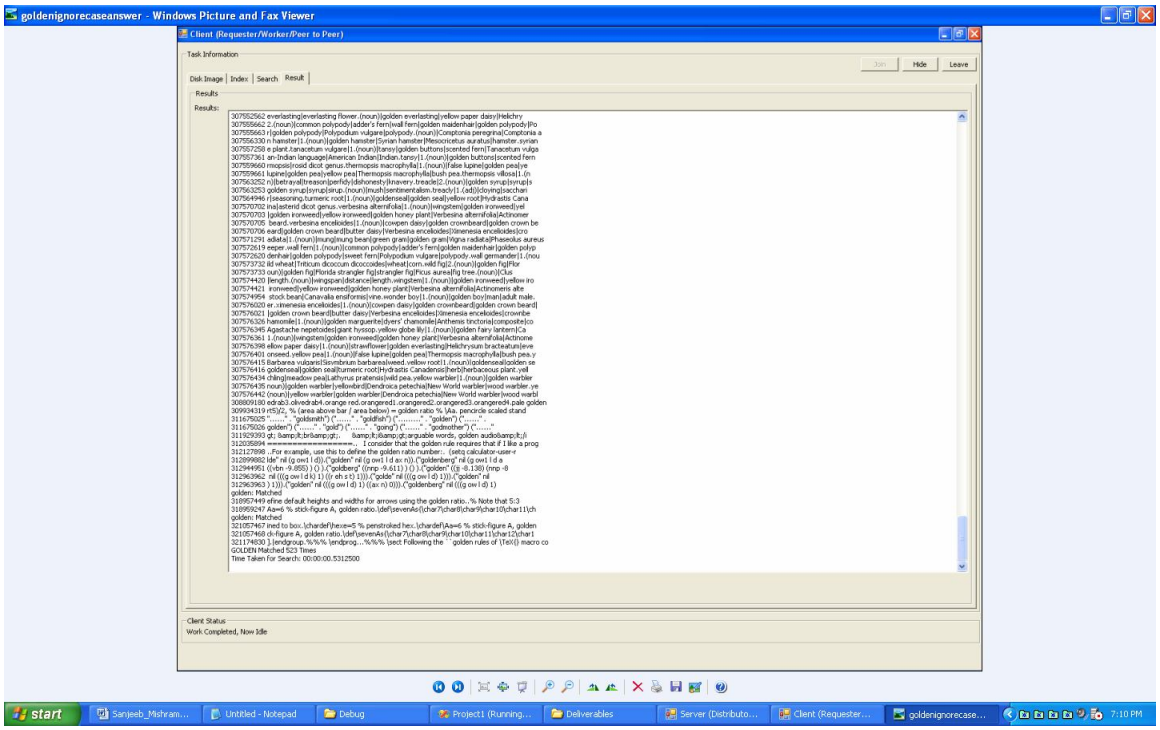
The above figure shows how index complete notification is received when indexing is completed.

ReceiveTaskCompleteIndex() method for indexing is implemented as a one-way method for maximum performance and efficiency since the workers do not need to receive any information back.

2.7.2.5.2 Receive Task Complete for Searching



2.10: Index Task Complete Notification



2.11: Search Task Complete Notification

The above figure shows how search complete notification is received when search is completed.

ReceiveTaskCompleteSearch() method for searching is implemented as a one-way method for maximum performance and efficiency since the workers do not need to receive any information back.

2.7.3 Task Worker

- Creating the Task Worker

  It allows users to submit task requests for indexing.

  It allows users to submit task requests for searching.

  - Client Startup

This is started when the client form loads.

  - Find Button

    This describes the index button and its associated index features in the main form.

  - Search Button

    This describes the search button and its associated search features in the main form.

- The System Tray Interface

  This holds logic for the display controls which is loaded when the application starts. This makes it certain that display icons are visible all the time for gathering status information to verify whether the client/worker computer is in the process of something or is idle.

- ClientProcess

  - Client Process

    This describes the client constructor, which tells about the configuration of the client/worker computer and the server IP port it should be connected to. This is invoked when the client starts since the client will be connected to a specified server.

- Login

  This adds a worker to the workers collection.

- Logout

  This removes a worker from the workers collection.

- Life Time

  This describes how lease time of client/worker computer is affected.

- Index Results

  This mentions the index results received by the client/worker process.

- Search Results

  This lists the search results received by the client/worker process.

- Indexing

  Index information is submitted as an index task to the server.

- Searching

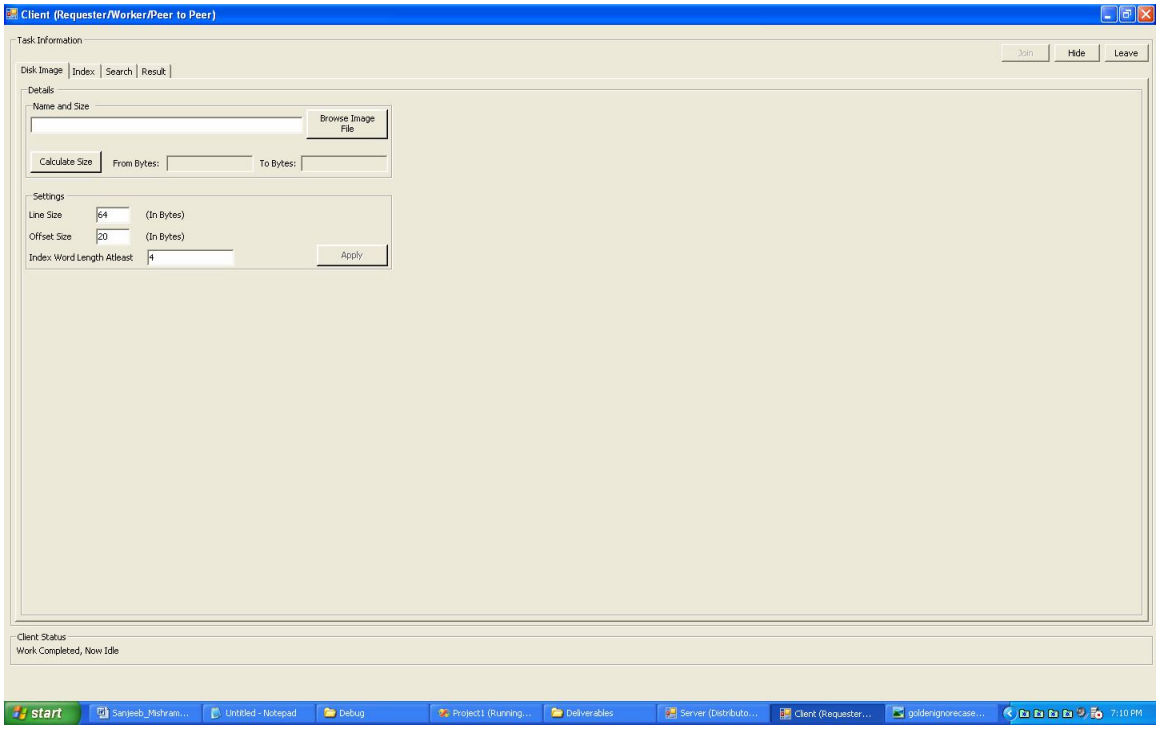
  Search information is submitted as a search task to the server.

2.7.3.1 Creating the Task Worker

It allows a user to submit task requests for indexing.

It allows a user to submit task requests for searching.

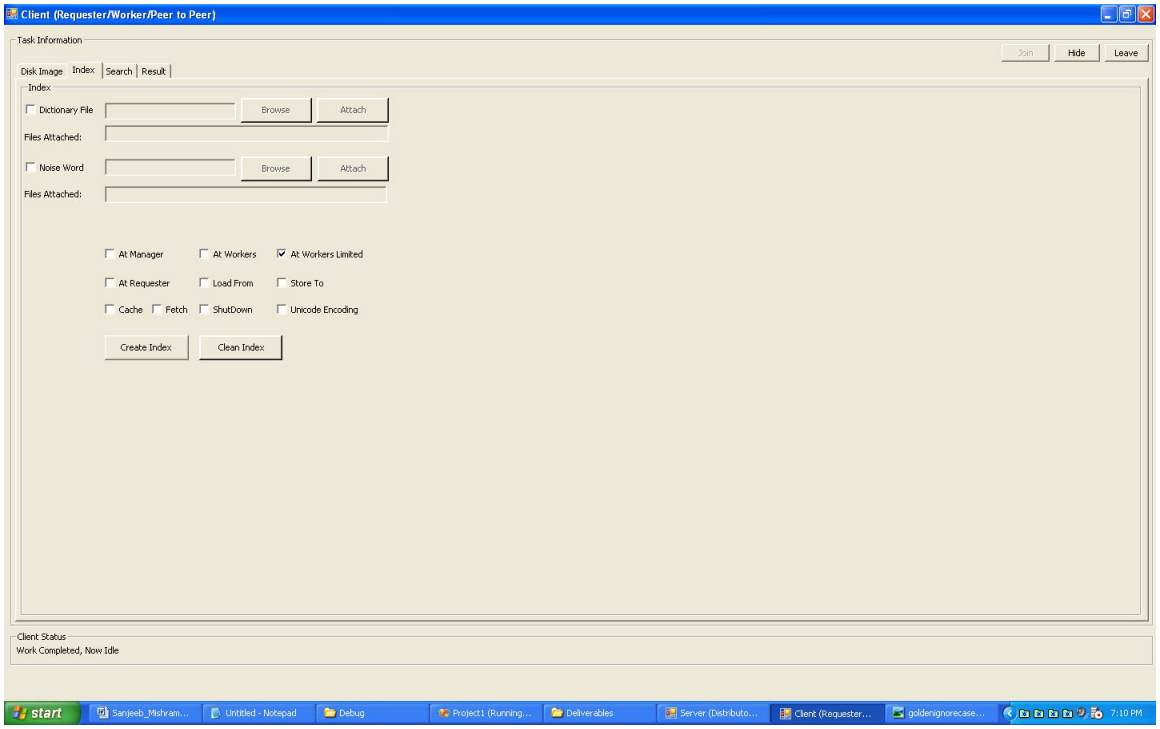
2.7.3.1.1 Client Startup



2.12: Client/Worker Startup Form

Client start up form shown above is started when the client form loads.

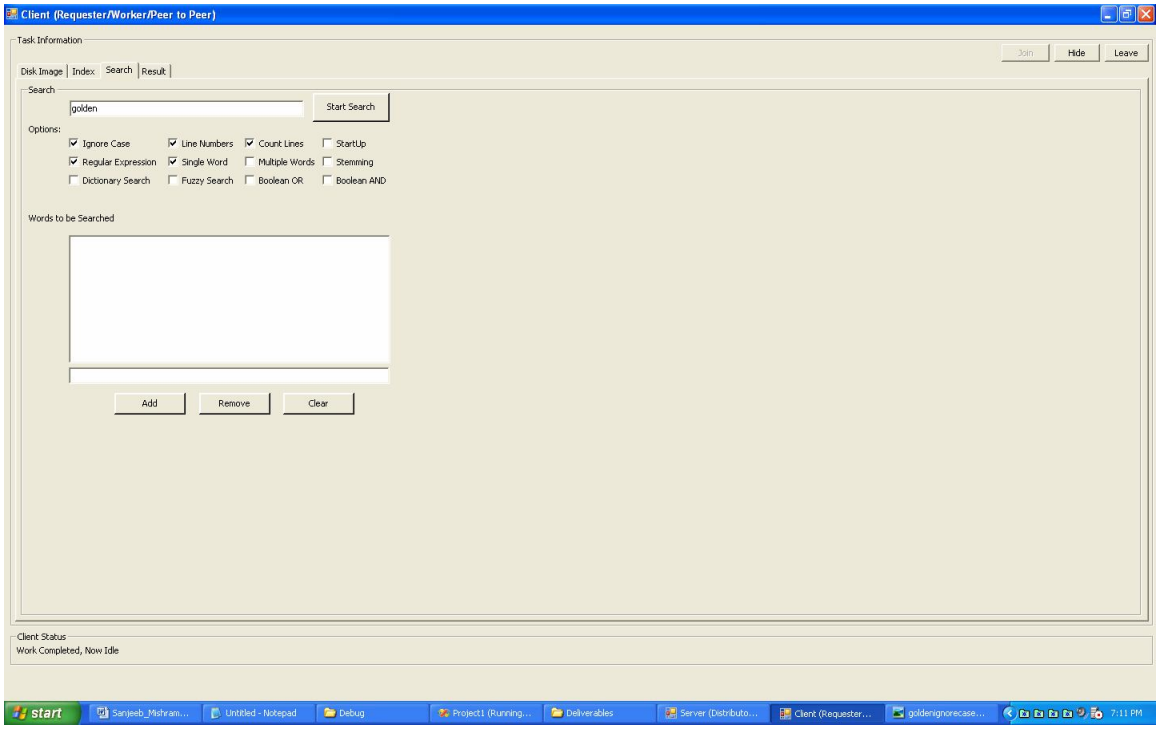
2.7.3.1.2 Find Button



2.13: Index Button

The above figure displays the Index Button and its associated features in the MainForm.

2.7.3.1.3 Search Button



2.14: Search Button

The above figure shows the Search Button and its associated features in the MainForm.

2.7.3.2 The System Tray Interface

The class that holds the logic for ContextMenu and NotifyIcon controls is created. This makes it certain that the NotifyIcon control is visible.

2.7.3.3 ClientProcess

2.7.3.3.1 Client Process

The following code mentions the ClientProcess constructor which tells about the configuration of the client/worker computer and the server IP port it should be connected to.

public ClientProcess()

```
{

RemotingConfiguration.Configure("TaskWorker.exe.config", false);
string ServerIP_Port = ConfigurationSettings.AppSettings.Get("ServerIP_Port");
Server = (ITaskServer)Activator.GetObject(typeof(ITaskServer), ServerIP_Port);

}
```

It calls server methods to request a new task, receives task complete notifications or task requests. It also includes two read only properties, which provide the server-generated GUID and the current status. Two status values exist which the client/worker can possess: the client can be idle or the client can be in the process of performing some task/work. Status values are provided in an enumeration as shown by the following:

```
public enum BackgroundStatus

   {

      Processing,

      Idle

   }
```

ClientProcess class works as a task worker by implementing ITaskWorker and as a task requester by implementing ITaskWorker interfaces.

The ReceiveTaskIndex() and ReceiveResultsIndex() methods for indexing and ReceiveTaskSearch() and ReceiveResultsSearch() methods for searching are implemented as one way methods to ensure that the server will not be put on hold while the client is performing. ReceiveTaskIndex() and ReceiveTaskSearch() methods perform their task in the method body and return the completed segment to the server.

2.7.3.3.2 Log in

This mentions the log in sample code of the worker which adds the worker to the workers collection, i.e. the worker is added.

```
public void Login()
{
        Control.CheckForIllegalCrossThreadCalls = false;
        _ID = Server.AddWorker(this); //Add Worker
```

```
}
```

2.7.3.3.3 Log Out

This mentions the logout sample code of the worker which removes the worker from the workers collection, i.e. the worker is removed.

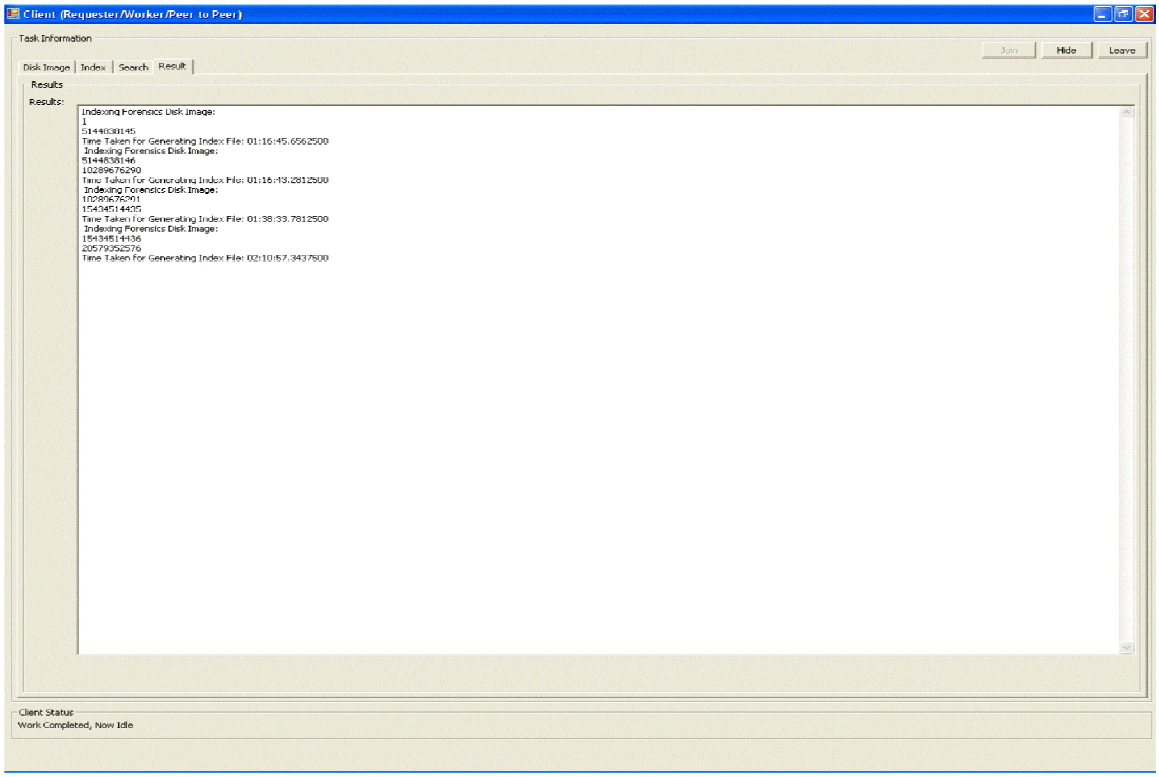
```
public void LogOut()
{
        Server.RemoveWorker(ID); //Remove Worker

}
```

2.7.3.3.4 Life Time

The following code sample shows how the lease time of client/worker is affected. As per the code the client/worker never dies and continuously tries for connection to the server. If one closes the worker application, its lease ends and the application closes down as expected. In the sample code below, “return null” shows that connection to the server will always be there.

```
public override object InitializeLifetimeService()
{
return null; //Connection never dies
}
```

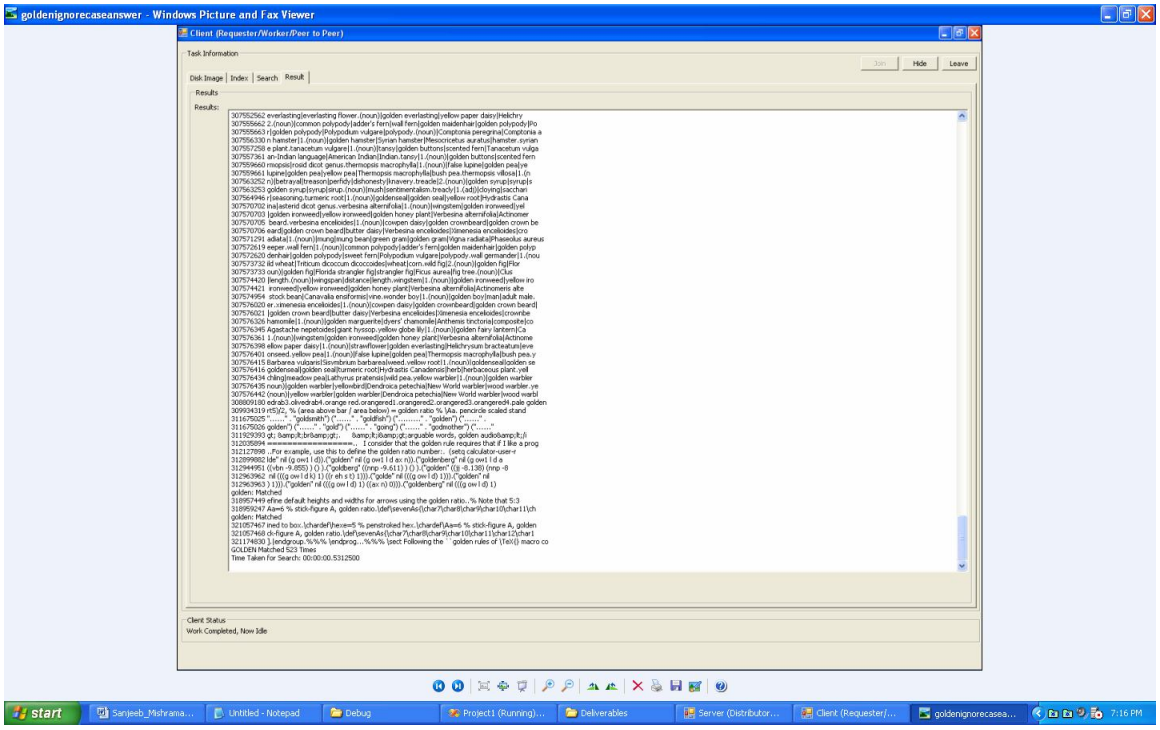
2.7.3.3.5 Index Results Received



2.15: Index Completed

The above figure mentions the index result received by the client/worker process.

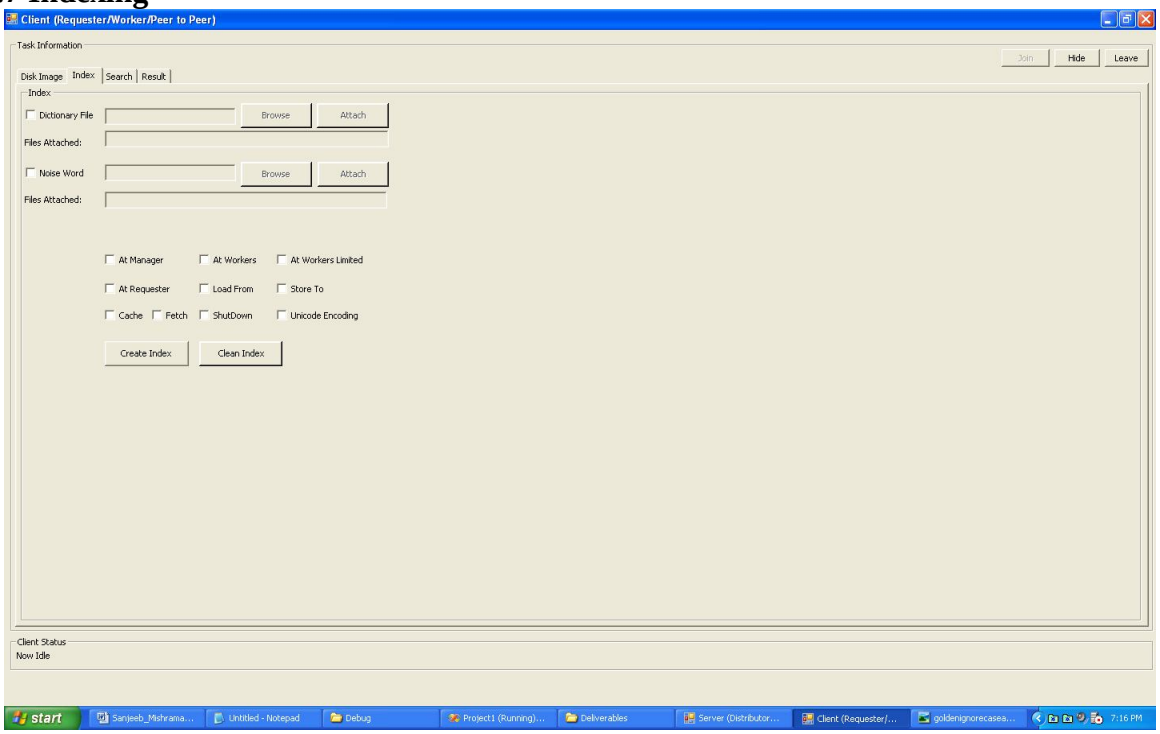
2.7.3.3.6 Search Results Received



2.16: Search results for “golden”

The above figure lists the search result received by the client/worker process.

2.7.3.3.7 Indexing

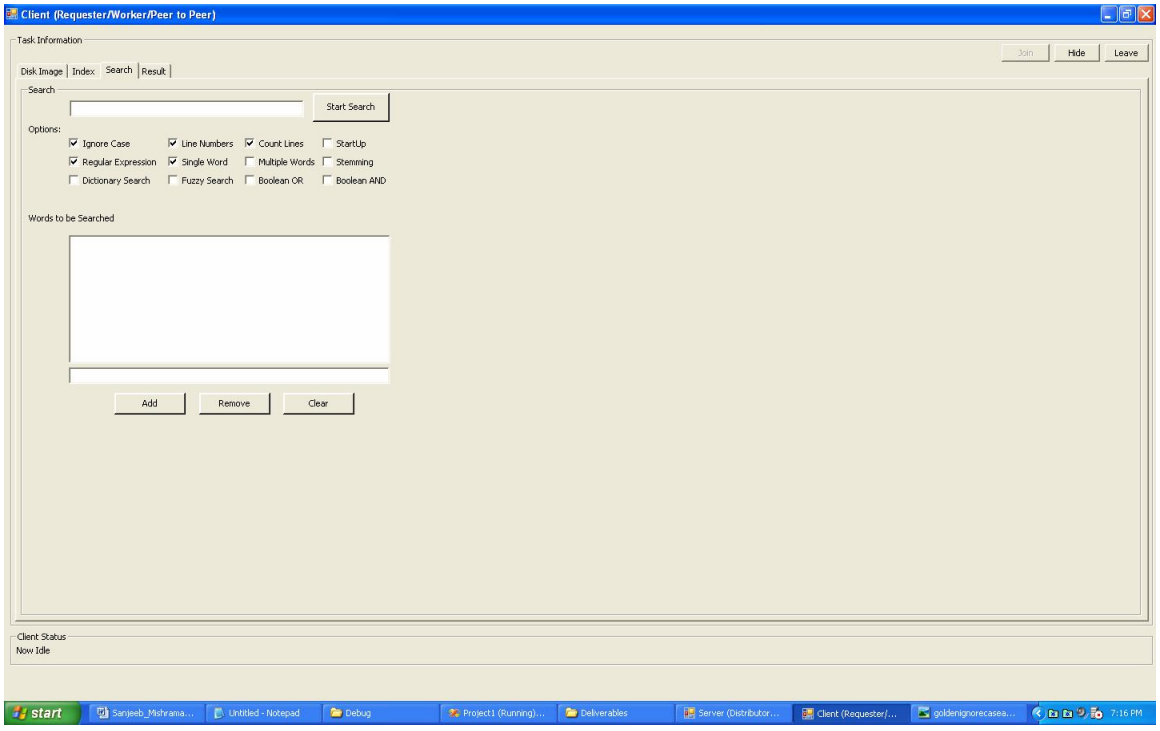


2.17: Indexing

Index information is submitted as a task to the server. Code sample shows the index method in MainForm.

```
public void FindIndex(long fromNumber, long toNumber,
Index.IndexCombination indu)
{
Server.SubmitTaskIndex (new TaskRequestIndex(this, fromNumber, toNumber,
indu));
}
```

2.7.3.3.8 Searching



2.18: Searching

Search information is submitted as a task to the server. Code sample shows the search method in MainForm.

```
public void FindSearch(long fromNumber, long toNumber,
Search.SearchCombination stru)
        {
                //MessageBox.Show("Inside"); //
                Server.SubmitTaskSearch(new TaskRequestSearch(this,
                fromNumber, toNumber, stru));
```

```
}
```

2.8 Summary

Distributed work manager provides improved performance and efficiency by dividing the work and reassembling it, thus reducing the overall time for finishing the assigned task. But it requires more programming than a single stand-alone application. Moreover, its benefits increase as the number of available clients increase and the work load mounts for the application, thus reducing the overall time. It divides the work load into multiple parts and sends them to available workers separately, thus reducing the overall time of execution.

We can see that distributed computing is always dependent on the problem or task in hand since some solutions are well-suited for certain types of problems while other applications are more oriented towards distributed computing. Most distributed supercomputers have their own individual approaches, which are customized based on the task and type of data they deal to provide an efficient and improved solution.

Chapter 3. FORENSICS APPLICATION DEVELOPED USING THE DISTRIBUTED FRAMEWORK

3.0 Motivation

Lack of indexing before keyword searching is a drawback in some forensics applications. Many applications search for occurrence of a specified keyword within the forensics image and take considerable amount of time. Due to lack of indexing, keyword searches cannot be carried out at a fraction of a second. Whenever keywords are searched, the whole disk image is read and if there is a match for the specified keyword, it is printed.

Many available forensics applications consume too much time for indexing and keyword searches are no faster. They lack efficient algorithms for faster indexing and the search operations lack depth while searching for keywords.

In many forensics applications which are based on keywords indexing and searching of disk images, when we search for keywords in index file, we miss some keywords as some are case sensitive and the software does not allow case sensitive searches, some are typed wrongly and cannot be searched as there is no provision in the software to make them correct, some are derived from their root words and although root words exist these derived words can't be searched in the index, some words are synonyms to other words and cannot be searched effectively, regular expression search is also sometimes not feasible as search functions do not have regular expressions facility, boolean search is also not possible with two or many keywords, and Unicode foreign language text search is also not possible when we deal with disk images seized in foreign countries as they are dealt in ASCII but not in Unicode formatting.

So the need of the hour is to design a forensics application which will employ efficient indexing techniques and while searching for indexed keywords, will care for stemming, fuzzy logic, boolean, and thesaurus search as well as regular expression search with provision for single word and multiple words search to increase the number of keyword hits. It should employ various search features to make keyword searches efficient and faster.

3.1 Application Goals

- Indexing algorithm should be efficient.
- Introduction of Index File in binary format is required for faster search.
- To remove unnecessary words from the index file, Index words of length at least 5/more are to be considered.
- Noise words are to be removed.
- The index has to run against a dictionary file.
- Index file should consume less storage.
- Overhead code should be removed (comparisons, redundant variable declarations/array declarations, disposing objects when not needed etc).
- Application should have good User Interface.
- Application should demonstrate Faster Index and Search Time.
- Effective Utilization of Appropriate Data Structures for faster index time is required.
- To get less than a second search time, Binary Search should be implemented.
- Faster Regular Expression Search Time (less than a minute) is needed depending upon the dataset size.

3.2 Design

Best solutions for indexing and searching not only employ faster indexing time, but less than a second of search time and a few seconds of regular expression search time.

Forensics Disk Image is a massive chunk of data imaged from a suspect's hard disk drive. A dictionary is a list of words one word per each line. This list may exist in a txt file or flat file one word per line. The dictionary file is used during indexing to minimize words in the index file. We can create a dictionary file with 3 letters or more to be used in the indexing phase [10].

An index is a binary file which stores a list of offsets for each word in the dictionary. Searching the index amounts to looking up the index file for a list of offsets. Metadata is compiled to index file and metadata contains information about locations, frequency, and so on about all significant words [10]. The algorithm for indexing takes approximately the same time for indexing a large number of keywords as it does to index a single keyword [10].

Typically when indexing the entire image indiscriminately, the tool tries to distinguish key words from random data in the image. A common technique is to simply index the disk image file through a strings program - i.e. we assume that key words consist of sequences of 4 or more printable characters. This arbitrary restriction on the type of keywords which are indexed is required in order to limit the complexity and size of the index. This restriction breaks when considering keywords which are binary in nature. Such keywords may occur in foreign language systems using Unicode for example [10].

Although it may seem more convenient to index the entire image for all the possible keywords that may occur within it, it is extremely inefficient. Most investigators are unlikely to search for arbitrary sequences of printable characters which may appear within the image. Usually investigators have a list of words - a hit on these words signifies an area of interest [10].

In order to improve the index file we have to consider many things:

- Index file should consume less storage.
- Index file should be user defined.
- Words are to be filtered out from the index file as much as possible.
- Noise Filters are to be used to minimize words in the index file.
- Dictionary files are used while indexing.

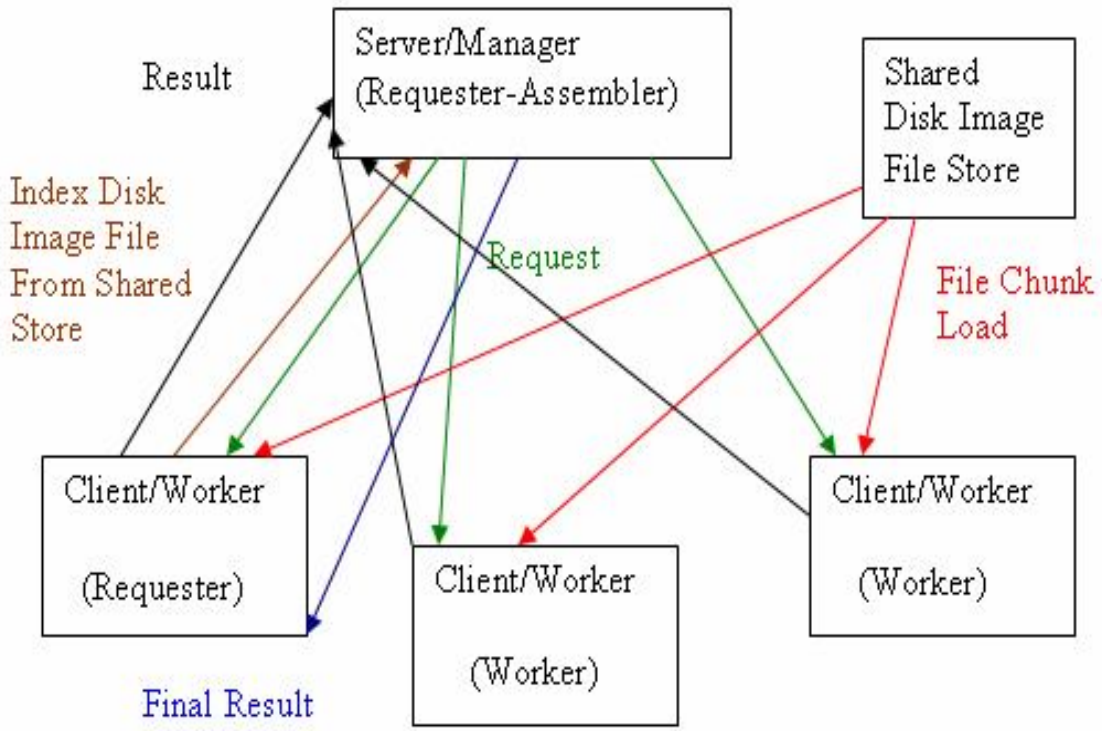
In the searching phase, the index file which is a list of offsets is read and the indexer prints the offset of each hit with few lines or single line contexts. While searching, the source file should be provided for the indexer to print some context

around each hit. It is impossible to search for a word which has not been previously indexed.

In order to improve searching we have to consider many things:

- Boolean Searching
- Stemming
- Thesaurus Search
- Fuzzy Search
- Text Categorization
- Information Retrieval

The architecture of the system is as shown below:



3.1: Architecture of the Forensics Application

Note:

Brown Arrow: A Client/Worker Requesting a task for server to perform using other available Clients/Workers

Green Arrows: Server requesting task/work to be performed at available clients/workers

Red Arrows: Available Clients/Workers getting their file chink from the shared store

Black Arrows: Available clients/workers giving results back to the server

Blue Arrow: Server sending results back to the Requester: the client/worker who had requested the server to handle the task

The overview of the tasks for our Digital Forensics Application is as follows:

- Distributing Forensics Disk Image
- Indexing
- Searching
- Important Features supported during Indexing/Searching
- Results Display

3.2.1 Distributing Forensics Disk Image

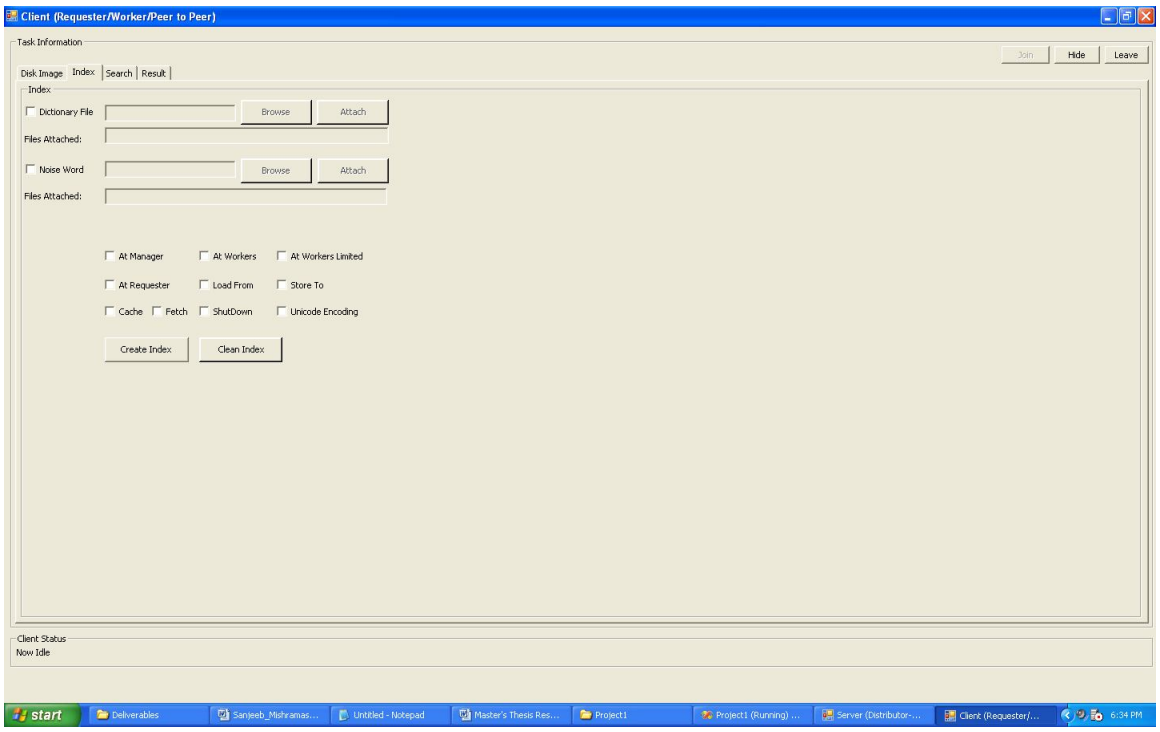
Disk Image can be distributed at three places such as:

- Done at Coordinator/Requester
- Done at Server
- Done at Client

When done at client, the image file chunk is downloaded from the shared store, which is accessible to all client computers. The individual clients start indexing their respective image file chunk.

When done at server, the image file is divided into number of parts equal to the number of clients available and respective parts are transferred to the available clients over TCP\IP.

When done at Requester or Coordinator, the requester divides the original forensics image file into number of parts and respective parts are transferred to the available clients over TCP\IP.



3.2: Disk Image distribution; At Client, At Server, At Coordinator

•Done at Coordinator/Requester

A Worker requests to perform a task utilizing other available Worker computers. This Worker becomes a Coordinator/Requester. This Requester divides the original forensics image file into a number of parts and respective parts are transferred to the available Worker computers over TCP\IP.

When this Requester does not act as a Server, it requests a task for the Server to perform using other available Workers. Server assigns part of the work to this Requester since it is available for work and this Requester in turn gets the needed file chunk from a shared store. Other Workers get their own image file chunk and start indexing and searching on this chunk sending results back to the server. After the Server collects all results from all individual Workers, it sends the final result back to the Requester for display. Performance depends upon where the image file is getting divided: if it is done at the Server, performance is slow and if it is done at the Worker, performance is higher.

When this Requester acts as a Server, it requests to perform a task using other available Workers. The image file is divided into a number of parts equal to the number of Worker computers available and respective parts are transferred to the available Workers over TCP\IP under supervision of Requester which in turn also gets a part of the work since it is available for work. Performance is slow as Workers get their respective file chunk from the Server and the Server gets the whole file from the shared store. This increases the performance overhead as it would have been much better had the Workers got their file chunk from the shared store directly.

- Done at Server

When done at Server, the image file is divided into a number of parts equal to the number of Worker computers available and respective parts are transferred to the available Workers over TCP\IP under supervision of the Server. Server computer is the node where server is running.

Server assigns part of the work to available Workers and Workers in turn get the needed file chunk from a shared store. Workers carry out the desired indexing and searching task on their chunk sending results back to the Server. After the Server collects all results from all individual Workers, it sends the final results back to the Requester for display.

Performance is slow as Server reads the image file, divides the file into number of available Workers, and sends the respective file chunk to available Workers. This is an extra overhead, which could have been eliminated had the Workers got their parts from the shared store directly rather than getting it from the Server.

- Done at Client

The file chunk is downloaded from the shared store to all individual Worker computers directly and the individual Workers start indexing on their image file chunk.

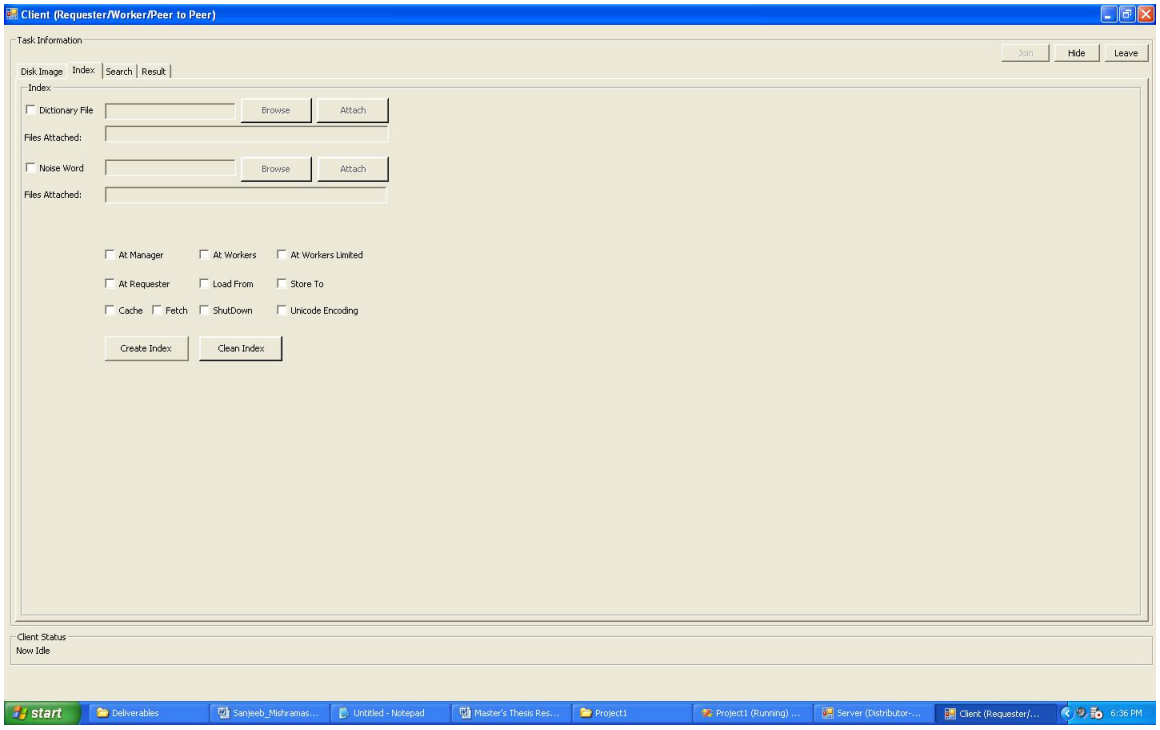
Performance is higher as Workers get their respective file chunk from the shared store directly rather than involving the Server.

3.2.2 Indexing

Various indexing options are available to be given as input before starting the indexing phase such as whether to do indexing at server, at client, or at coordinator/requester. Many indexing parameters are required for indexing such as requirements for dictionary file and noise word list file. Others include specifying the given file name, whether to send file contents in an array, whether startup or shutdown is required, mentioning line and dictionary word length, whether Unicode encoding is required for image file and mentioning cache, fetch, load and store options for image file at either server or client computer.

The parameters passed for indexing are as follows:

- FileName: File Name as a string
- Array: File Contents sent in an array
- DictionaryFile: Denotes a Boolean whether a Dictionary File is needed
- DictionaryFile: Dictionary File
- NoiseFile: Denotes a Boolean whether a Noise File is needed
- NoiseFile: Noise File
- Cache: RAM to Disk
- Fetch: Get the required file from Disk to RAM
- Load: Load the file from a shared storage to RAM
- Store: Store the file in shared storage from RAM
- StartUp: At Startup of the client/worker processes
- ShutDown: At shutdown of client/worker processes
- AtServer: At Server
- AtClient: At Client
- AtCoordinator: At Coordinator/Requester
- LineSize: Line Size (64 bytes/128 bytes etc)
- OffsetSize: Offset Size in calculation of a line (64 bytes/128 bytes etc)
- DictionaryWordLength: Index Word Length
- UnicodeEncoding: Denotes a Boolean whether Unicode encoding is required



3.3: Parameters for Indexing

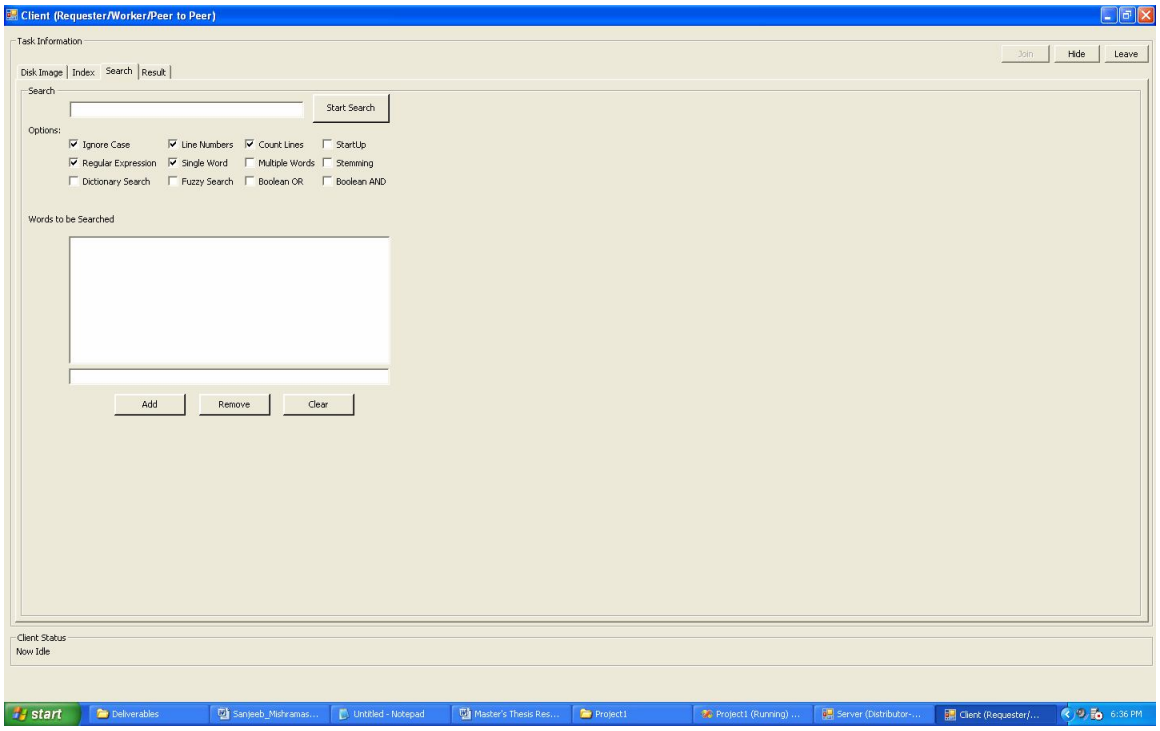
3.2.3 Searching

Various search options are available to be given as input before starting the search phase such as regular expression or non-regular expression searches. Search parameters are needed while searching for keywords in the index file such as: single word search, multiple word searches, boolean search, fuzzy search, thesaurus search, stemming, ignore case search, and regular expression search.

The parameters passed for searching are as follows:

- CleanIndex: Clean the Index File
- SingleWord: Single Word/String Search
- MultipleWord: Multiple Words/Strings Search
- SearchStrings: Search String Array
- RegularExpression: Denotes a Boolean whether Regular Expression is required
- IgnoreCase: Denotes a Boolean whether Case is ignored
- LineNumbers: Denotes a Boolean whether Line Numbers is required
- CountLines: Denotes a Boolean whether line count is required
- BooleanSearch: Denotes a Boolean whether Boolean search is required
- Stemming: Denotes a Boolean whether stemming is required
- FuzzySearch: Denotes a Boolean whether fuzzy search is required
- HexEditor: Denotes a Boolean whether hex editor implementation is required

•DictionarySearch: Denotes a Boolean whether dictionary search is required
•StartUp: Denotes a Boolean whether to search at start up for worker process
•LineSizeSearch: Denotes a Line Size Length
•OffsetSizeSearch: Denotes a Offset Size Length



3.4: Parameters for Searching

3.2.4 Important Features during Indexing and Searching:

Other important features added to the toolkit are as follows:

- Ordinary Search

This search is done without regular expression search capabilities and is used only for normal keyword search. This is a basic search and does not give satisfactory results when we deal with collection of strings or regular expression searches.

•Regular Expression

Regular expression search is a powerful feature which is used for performing regular expression searches.

•Single Word Search

Only a single word is searched with no features for thesaurus, fuzzy and stemming search available.

- Multiple Words Search

Multiple words search is enabled in order to increase the keyword hits as this search is corroborated by thesaurus, fuzzy, and stemming searches.

- Word Selection Search (Select the words you want to search and delete others)

This search option is allowed to delete all unnecessary keywords from the search list, which the user does not want to search after they are added to the list during fuzzy, thesaurus, and stemming searches.

- Dictionary File

Dictionary file is used to reduce the size of the index file which takes considerable storage. If there is a hit on any word in the dictionary file, that word is indexed otherwise it is passed on.

- Noise Word List File (I, me etc words not in Index file)

Noise word list file is used to reduce the size of the index file. If there is a hit on any word in the noise word list file, that word is not indexed. It eliminates the noise words such as the, he, she, me etc during the indexing phase.

- Alphabet File (Which characters are not printable etc and how to show them)

This demonstrates which non printable characters are to be displayed by which character set.

- Stemming

Stemming is used for getting the root words of the specified keyword and adds all stem words to the current list.

- Boolean Search

Boolean search is used for “ORing” or “ANDing” two keywords while searching for combination of words.

- Fuzzy Search (Words Correction and More Words in search criteria)

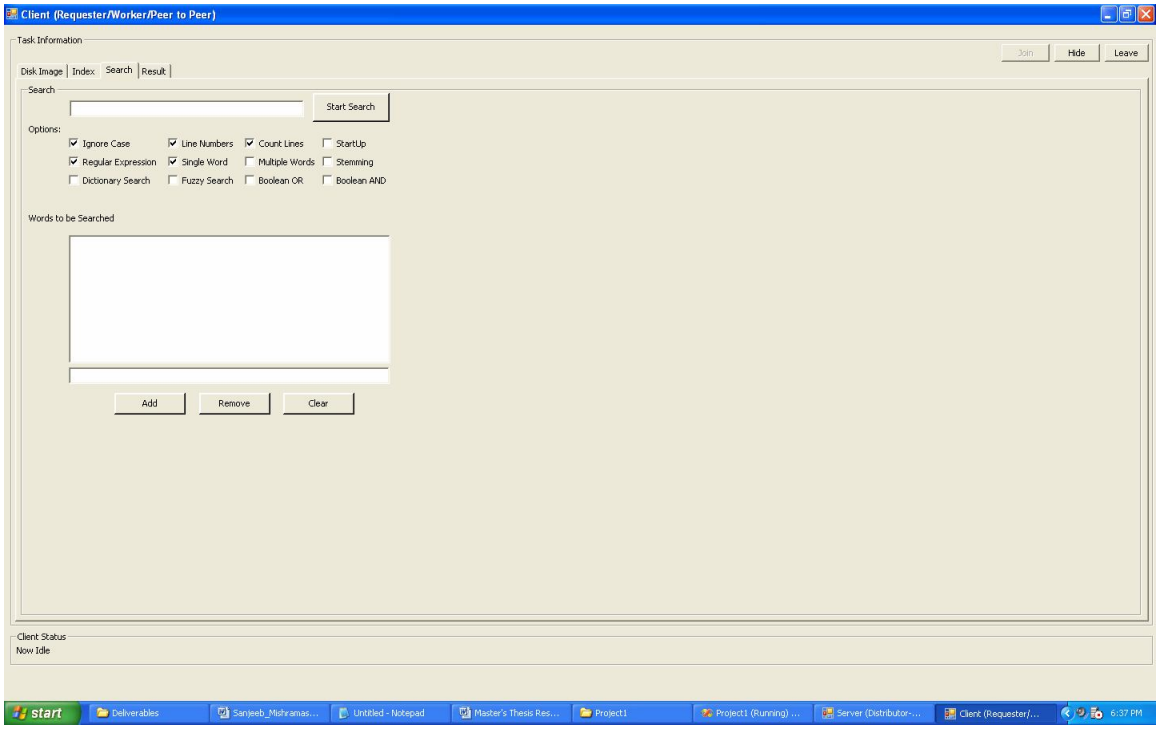
This is used for correcting the misspelled keywords. Fuzzy Search adds all fuzzy words to the search list so that search hits will be more.

•Thesaurus Search (Similar Words Search)

This is used primarily for getting all similar words of a keyword. Thesaurus Search adds all these similar words to the list. Hits will be more as more keywords are searched against the forensics image.

•Unicode Implementation (Language Packs)

Unicode search is used for searching foreign language text in foreign languages.



3.5: Other Search Features

3.2.5 Results Display

This is the phase where the index and search results are displayed.

After indexing, index results are displayed. This outputs the indexed parts with their file size in bytes and other client computer information such as start time of index, end time of index, and its sequence number.

After searching, search results are displayed. This outputs the search string with its number of occurrences and some context around each hit inside the forensics image.

3.3 Implementation

The whole forensics application is written in Visual C# using Visual Studio.NET 2005. C# Windows Forms has been used to create the windows forms and modules have been developed using C# 2.0 on VS.NET 2005. C# Generics is used which is a part of VS.NET 2005 implementation and was not a part of VS.NET 2003 implementation. .NET Framework 2.0 has been used which is default with VS.NET 2005. VS.NET 2005 was officially released in 2005. The distributed framework has been designed using .NET Remoting.

.NET Remoting enables one to work with stateful objects and, hence, it is the future of distributed applications. .NET Remoting gives one a flexible and extensible framework that allows for different transfer mechanisms (HTTP and TCP), encodings (SOAP and binary), and security settings (IIS and SSL). .NET Remoting is well suited for any distributed application. One can choose between HTTP/SOAP for the Internet and TCP/binary for LAN applications by changing a single line in the configuration file. Remoting is the process of programs or components interacting across certain boundaries. These contexts will normally resemble either different processes or machines.

C# Generics in VS.NET 2005 adds type safety that was lagging in VS.NET 2003. Using C# Generics, it is no longer necessary to employ casts to translate between object and the type of data that is actually being operated upon. C# Generics expands one's ability to reuse code and it does that safely and easily.

3.3.1 Indexing Options

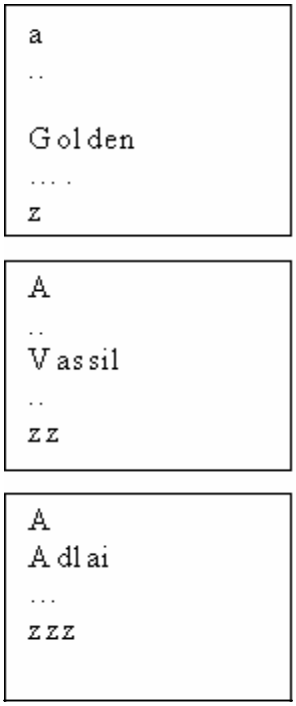
Indexing Options are various index options available to be given as input before starting the indexing phase. Many indexing options are added such as requirements for dictionary file, noise word list file, whether the forensics disk image file will be divided at client or at server or at co-coordinator. Other indexing options include file name details, file contents sent in an array, whether startup or shutdown is required, line length, dictionary word length, whether Unicode encoding is required for image file and cache, fetch, load, store options for image file at either server or client.

Many Ways of Indexing

- Multiple Sorted Index Files
- One Unsorted Index File
- One Sorted Index File
- No Index File, only Tree Structure

3.3.1.1. Multiple Sorted Index Files

In this indexing type, multiple sorted index files are created. All these sorted index files are searched for the specified keyword one by one which takes a lot of time.



3.6: Multiple Sorted Index Files

(Pseudo Code)

Create a TreeMap

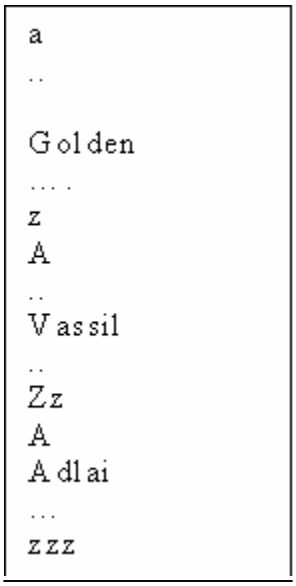
TreeMap cannot grow much due to limited RAM Size

Dump all TreeMap entries to an Index File

Create a new TreeMap and dump all its entries to a new Index File

3.3.1.2 Single Unsorted Index File

In this indexing phase, no multiple indexes are created. Only one unsorted index file is created. But this unsorted index file is a collection of many sorted index file entries. We are going to search for keywords in this unsorted index file.



3.7: Single Unsorted Index Files

(Pseudo Code)

Create a TreeMap

TreeMap cannot grow much due to limited RAM Size

Dump all TreeMap entries to an Index File

Create a new TreeMap and dump all its entries to the same Index File

3.3.1.3 Single Sorted Index File

In this indexing procedure, no multiple indexes are created. One sorted index file is needed. Indexing takes little longer but search time is fastest due to binary search. Extra overhead for sorting entries in the index file adds up the index time. The sorting of entries in the index file can be done after all entries are created or when new entries are added to the index file.

3.8: Single Sorted Index Files

(Pseudo Code)

Create a TreeMap

TreeMap cannot grow much due to limited RAM Size

Dump all TreeMap entries to an Index File

Create a new TreeMap and merge all its entries to the existing entries in the same Index File

3.3.1.4 No Index File, only Tree Structure

This indexing technique takes time for serialization and de-serialization. This type of indexing is faster for low disk sizes such as 50 MB/100 MB etc.

(Pseudo Code)

Create a TreeMap

TreeMap cannot grow much due to limited RAM Size

Serialize the TreeMap to a disk file

Create new TreeMaps one by one and Serialize them to respective disk files

While Searching for Strings

{

DeSerialize disk files into corresponding TreeMaps one by one

Search for Strings inside TreeMap //(TreeMap.contains(golden))

}

3.3.2 Searching Options

Searching Options are various search options available to be given as input before starting search phase. Many search options are added such as: single word search, multiple word search, boolean search, fuzzy search, thesaurus search, stemming, case insensitive search, and regular expression search.

Many Ways of Searching

- Binary Search (Faster)
- Linear Search (Slow)
- Multiple Sorted Index Files
- No Multiple Index Files, One Unsorted Index File (Many Sorted Index Files existing in One Big Index File one by one)
- No Multiple Index Files, One Sorted Index File
- No Index File, Only TreeMap Data Structure

3.3.2.1 Binary Search

For Non Regular Expression Search, Binary Search is the best bet. It does not matter whatever big the index is, the keyword is found in the index file in few as 7 or 8 steps and, thus, the search for a keyword in an 80 GB forensics image file takes the same amount of time as against a 20 GB forensics image file. This type of search mechanism is not employed for Regular Expression as for them we use linear search and all entries in the wordlist are compared to find the search item. This search is faster, takes less time, and is suitable for non regular expression searches. This search time is mostly less than a second for most cases as we have to deal with 7 or 8 steps in all cases irrespective of the dataset size.

3.3.2.2 Linear Search

Regular Expression Search is implemented through Linear Search as all entries in the wordlist are to be compared to find the search item. This search is not faster, takes more time as it has to compare all words in the image file, and is suitable only for regular expression searches. Here the search time depends upon the dataset size.

3.3.2.3 Searching in Multiple Sorted Index Files

In this search procedure, all sorted index files are searched for the specified keyword one by one and this increases the usual search time.

(Pseudo Code)

For each Index File.dat, StartString-EndString.dat, StringNumber-OffsetNumber.dat

{

Read the file "StartString-EndString.dat" which mentions Start String and End String Number (Start String is the first indexed string and End String is the last indexed string in the forensics image)

Read Start String and End String Number

Do Binary Search on these Start String and End String Number till you get the Search String.

If Search String is found

{

Find the Search String Number

Read the file "StringNumber-OffsetNumber.dat" which denotes String Number as well as its Offset Number

Using the Search String Number, find the Offset Number in "StringNumber-OffsetNumber.dat" file

Locate the Search String inside the Index File using the Offset Number and Print all its details: String Name, Number of Times it Appeared, Places it Appeared etc.

}

If Search String is Not Found

Continue

}

3.3.2.4 Searching in Single Unsorted Index File

In this search type, all sorted index file entries in the unsorted index file are searched one by one for the particular keyword. This adds extra overhead to the usual search time. Since we have many sorted index files existing in one big index file, searching the big index file for a particular keyword is equivalent to searching that keyword inside many sorted index files one by one.

StartString-EndString.dat (File denoting Start String and End String Number)

- 1 74903 (Start String Number = 1 and End String Number = 74903)
- 74904 149806
- 149807 224709
- 224710 284512
- ….

StringNumber-OffsetNumber.dat (File denoting String Number and its Offset Number)

- 1 …
- 2 …
- … …
- 7 31 (String Number = 7 and Its Offset = 31)
- 9 51
- 13 108
- 15 138
- … …
- … …

IndexFileName.dat (Index File)

- ….

- CreateStroke 2 39565 39968 (String = CreateStroke, Times Appeared = 2)
- CreateStrokes 2 39954 39955 (String Positions: 39954 and 39955)
- CreateTextServices 1 558198
- ….

(Pseudo Code)

Read the file “StartString-EndString.dat” which mentions Start String and End String Number.

For Each Start String and End String Number in file “StartString-EndString.dat”

{

Do Binary Search on these Start String and End String Number till you get the Search String

If Search String is found

{

Find the Search String Number

Read the file “StringNumber-OffsetNumber.dat” which denotes String Number as well as its Offset Number

Using the Search String Number, find the Offset Number in “StringNumber-OffsetNumber.dat” file

Locate the Search String inside the Index File using the Offset Number and Print all its details: String Name, Number of Times it Appeared, Places it Appeared etc.

}

If Search String is Not Found

Continue

}

3.3.2.5 Searching in Single Sorted Index File

Search time is fastest due to binary search and Google supports this type of search for faster data retrieval. This is the fastest way of finding the keyword inside the sorted index file.

For example, consider a search string “CreateStrokes” which has to be searched inside the forensics image with its number of appearances and its context around each hit.

The file denoting Start String and End String Number is described as follows where Start String is the first indexed string and End String is the last indexed string inside the forensics image:

IndexFileNameStartEnd.dat (File denoting Start String and End String Number)

- 1 74903 (Start String Number = 1 and End String Number = 74903)

Performing Binary Search to find the search string “CreateStrokes” using String Number 1 and String Number 74903 as two input values, we get the String Number 7 as the search string number since it holds the search string “CreateStrokes”.

The file denoting String Number and its Offset is described as follows:

IndexFileNameNumberOffset.dat (File denoting String Number with its Offset Number)

- 1 …
- 2 …
- … …
- 7 31 (String Number = 7 and Its Offset Number= 31)
- 9 51
- 13 108
- 15 138
- … …
- … …

We find that the search string “CreateStrokes” (String Number 7) has got Offset Number 31. This Offset Number is the offset where the search string “CreateStrokes” can be found with other important details inside the index file.

The index file is described as follows:

IndexFileName.dat (Index File)

- ….

- CreateStroke 2 39565 39968 (String = CreateStroke, Times Appeared = 2)
- CreateStrokes 2 39954 39955 (String Positions: 39954 and 39955)
- CreateTextServices 1 558198
- ….

The search string “CreateStrokes” is found out traversing the same offset value from the beginning of the index file. The search string “CreateStrokes”, its

number of appearances (2) and its context around all places/positions (String Positions: 39954 and 39955) are printed.

39954: CreateStrokes Found
I am going they are not coming I am loving them CreateStrokes car lollypop
39955: CreateStrokes Found
CreateStrokes I am fine and they are good for the sake of I am going there
CreateStrokes Found 2 Times

(Pseudo Code)

Read the file "StartString-EndString.dat" which mentions Start String and End String Number (Start String is the first indexed string and End String is the last indexed string in the forensics image)

Read Start String and End String Number

Do Binary Search on these Start String and End String Number till you get the Search String

If Search String is found

{

Find the Search String Number

Read the file "StringNumber-OffsetNumber.dat" which denotes String Number as well as its Offset Number

Using the Search String Number, find the Offset Number in "StringNumber-OffsetNumber.dat" file

Locate the Search String inside the Index File using the Offset Number and Print all its details: String Name, Number of Times it Appeared, Places it Appeared etc.

}

If Search String is Not Found

Quit the Program

3.3.2.6 Searching in TreeMap Data Structures

This search mechanism is faster as it finds its result only using a single statement such as “TreeMap.contains (Keyword)” and no other overhead is required. The search time is more when the tree size is big, but such is not the case with binary search which is the fastest way of finding the search string inside the sorted index file.

While Searching for strings:

DeSerialize disk files into corresponding TreeMaps one by one

Search for Strings inside TreeMap //(TreeMap.contains(golden))

If (TreeMap.contains(golden)) //If TreeMap contains Search String

{

“golden” Found //Search String Found

}

Else

{

“golden” Not Found //Search String Not Found

}

3.3.3 Indexing Component

Indexing component is used for describing how the index task is carried out by the toolkit. It also mentions other index features implemented by the toolkit.

Design and Implementation

Index Method is used for indexing and is the main part where indexing of the forensics disk image is performed. But before Indexing, many other operations are carried out such as Image File Division to divide the image file, testing for Unicode Operation to find whether Unicode functionalities are to be carried out, Dictionary File attachment to index words found in the dictionary file, Noise Word File attachment to eliminate unnecessary words from the index file. Image File division is used for disk image file division and its distribution among clients. For faster indexing, the file division calculation is performed at the client and parts are fetched from a common storage for each client. For Unicode encoding,

the disk image file is converted to Unicode format. Here we have considered only UTF-16, but other Unicode formats also can be considered such UTF 8 and UTF 32. Before the client shuts down, whatever the client has in its RAM is stored into its ROM and when the client starts, the ROM contents get back to RAM. The dictionary files are attached and the words which are found in the dictionary files are only indexed. The noise wordlist files are used for removing noise words from the index file.

The core part where indexing of the forensics image is done is very simple. It searches for strings inside a line of some byte length in the forensics image and indexes these strings one by one. TreeMap is a sorted binary tree implementation of maps and contains MapEntry objects, where each MapEntry object contains two parameters: one is a string and the other one is a TreeSet (contains integer parameters). Moreover, TreeSet is a binary tree implementation of sets. So, TreeMap stores the string with its line numbers/positions (TreeSet) in each of its MapEntry object. The TreeMap contains a number of MapEntry objects and their number is equal to the number of strings indexed in the forensics image.

The forensics disk image is divided into lengths of 64/128/256/512/1024 bytes. These byte length units are called lines. First an empty TreeMap is declared. The forensics disk image is read line by line, the chosen line is divided into possible strings, and all strings are put into an array. For each string in the given line, if the string is non-empty and the declared TreeMap doesn't contain the string, TreeMap stores the string and its line number in a new MapEntry object. If the string already exists, TreeMap adds the line number of the string to the existing TreeSet entry of the existing MapEntry object. In summary, MapEntry object is used for storing string and its line numbers, TreeSet is used for storing all line numbers inside the MapEntry object, and TreeMap stores all MapEntry objects.

```
String[] res = delim.Split(line); //divides the line into strings and stores them into an array
lineno++; //Console.WriteLine(lineno);
foreach (String s in res)  //for each string found in array
{
        if (s != "")  //if string is not null, then process that
         {
                if (!index.Contains(s))  //if string doesn't exist
                        index[s] = new OTreeSet<int>(); //Create new TreeSet
                index[s].Add(lineno); //Add Line No.
        }
}
tlineno++;
}
```

Index file stores strings with their offsets for future retrieval. "wordlist.Key" is the string and "wordlist.Value.Count" is the number of times the string appears.

Using a foreach statement the lines where the string is found can be found out. As the code snippet shows below, using a foreach statement all MapEntries objects in the TreeMap data structure are iterated and “wordlist.Key”, “wordlist.Value.Count” and its different line numbers are printed. MapEntry object is a collection of string and its line numbers. For each MapEnrty object, “wordlist.Key” is the string, “wordlist.Value.Count” is the number of times the string appears, and “ln” is its line number in the forensics image.

```
foreach (MapEntry<String, TreeSet<int>> wordlist in index)
{
try
{
        dataOut.Write(wordlist.Key); //string
        dataOut.Write(wordlist.Value.Count); //Number of times string appears
        foreach (int ln in wordlist.Value)  //Lines where the string is found
        {
                dataOut.Write(ln);
        }
}
catch (IOException exc)
{
        MessageBox.Show(exc.Message);
}
}
```

(Pseudo Code)

Get Forensics Image File as Input

Divide it into 64 Bytes length (Can Changeable: 128/256/512/1024 etc)

Create a TreeMap

Store all Strings found in the 64 bytes length image file chunk into an array

Scan the array for each String

{

If the String doesn't exist in the TreeMap

        {

                Create a new MapEntry object with the String and its Line Number
                //TreeMap contains different MapEntry objects

```
        }

        If the String exists in the TreeMap

        {

                Add a new Line Number value to the existing TreeSet entries of
                the MapEntry object //MapEntry stores string and TreeSet entries
                //TreeSet contains different Line Numbers/Positions of the string

        }

}

For Each MapEntry objects in TreeMap

{

Print “wordlist.Key” //string

Print “wordlist.Value.Count” //Number of times the string appears

For Each (Entries in wordlist.Value) //Lines where the string is found

{

Print "wordlist.Value" //Line where the string is found

}


}
```

3.3.4 Data Structures for Indexing

This explains how various data structures are used to index strings.

Use of Specialized Data Structures

- Specialized data structures have to be used for faster indexing and searching of forensics data.
- Since we are dealing with keywords and their offsets inside the forensics image, use of special data structures which can store key and value pairs is needed.

- Data Structures are stored in memory and are read when needed. To make these read and write operations faster, use of Generics is highly required which makes data conversion easy and more flexible. Data Structures designed using Generics perform faster and have many storage benefits.

Use of C# Generics

- In C# everything is an object. To store strings and their places in the nodes of a treelike data structure, most times we have to convert string and integer types into objects. To store them back from objects to their respective types, they have to be unboxed again. This boxing and unboxing running several times during the process of an application makes the application slower. To eliminate this conversion time, use of C# Generics is highly recommended as Generics add type safety to the existing data structures and make the process easier and faster without considering boxing and unboxing overheads.

- Using Generics, code can be generalized so that it can be used by many types of data

- Performance is higher since boxing and unboxing of data is eliminated.

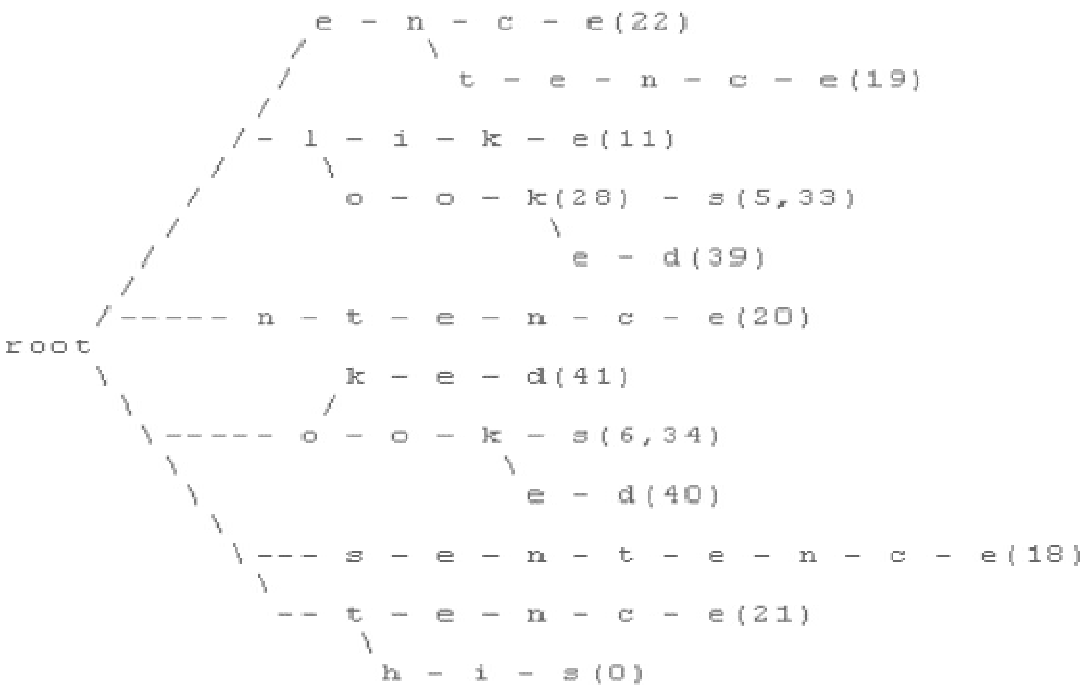
- More clarity to code as many data types can be used with a single generalized data type which removes a lot of redundant code.

Design and Implementation

In order to visualize the data that is contained in an index, a small example is presented. Consider an example where we have a line "This looks like a sentence: look looks looked". If we index this line considering indexed words of length at least 4, we get the following index information [68]

0 this	22 ence
5 looks	28 look
6 ooks	33 looks
11 like	34 ooks
18 sentence	39 looked
19 entence	40 ooked
20 ntence	41 oked
21 tence	

When we represent this in a tree structure, we get the index tree as follows:



3.9: Index Tree

The tree representation uses the letters of the indexed strings as nodes in a tree and at the node of the final letter of an indexed string, the offsets of that string are located. When we search for the string "look", only the letters of the string have to be walked in the tree from the root node to see that the string is present at location 28. If all strings starting with "look" are to be found, all nodes beneath that node have to be accounted for too, thus resulting in the locations 5, 28, 33 and 39. Although various data structures exist for faster indexing, an efficient data structure is described below on which based is our forensics application.

For faster indexing, TreeMap is used. TreeMap is a list of items which have a tree like relation to each other. It has one root, several branches and leaves. To add a new item to the TreeMap, a suitable place is found out and the new item is inserted there. If the TreeMap does not have a root, the first item will be inserted at the root. When new items are inserted to the TreeMap, the key of the new item is compared to the key of the root. If it is greater than the root, then it will be added to the right node. If it is less than the root, it will be added to the left node. If the left or right node is occupied by an item, it will be added to that item. TreeMap stores items in their sorted natural order since 2 is stored before 5 and "cat" is stored before "zebra" for example. TreeMap helps to find stuff quickly in memory. To make it easier to work with, C# Generics can be utilized to design TreeMaps.

A TreeMap is implemented using red-black balanced ordered binary trees. It performs Add, Remove, and Contains operations to operate on data inside it. TreeMap is used to create maps/dictionaries. A map/dictionary is a data structure which stores key and value pairs. In our forensics application key is the string and the value is its position inside the forensics image. The Key-Value pairs are stored as MapEntry objects inside the TreeMap. Key and Value methods are used to retrieve the actual values in a MapEntry object. These methods return the stored values when a key–value pair is entered into the map/dictionary (TreeMap). While searching for a keyword in the index file, the word which is searched is the key and its offset is the value. When a keyword is found at different places inside the forensics image, the MapEntry object uses TreeSet to store different line numbers inside its value entry. Thus, MapEntry object stores two parameters: one is a string and the other one is a TreeSet (contains line numbers). TreeSet is a binary tree implementation of sets and contains integer parameters. TreeSet can contain one or many value entries depending upon the number of times the keyword has appeared inside the forensics image. In summary, TreeMap stores the string with its line numbers/positions (TreeSet) in each of its MapEntry object. Since TreeMap and TreeSet are implemented using red-black balanced ordered binary trees, design and implementation of red black trees is of utmost importance to understand the critical data structures used during indexing phase of the forensics application.

A Red Black tree is an ordered binary tree and each node of the Red Black tree uses a color attribute: red or black to keep the tree balanced. While an AVL Tree uses balance factor to balance the tree, a Red Black tree uses color attributes to balance the tree [70].

The Red Black tree has the following properties:

1. Its root is black

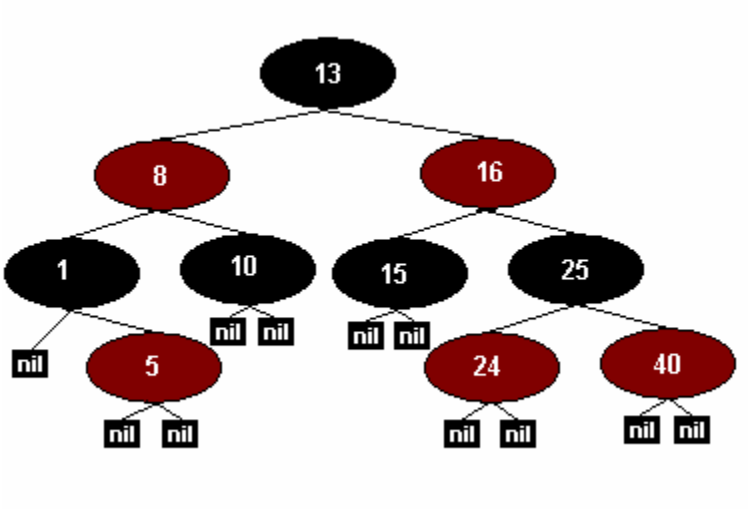
2. All of its leaves are black

3. All of its red nodes can only have black children

4. All paths from a node to its leaves contain the same number of black nodes

Since all paths of a Red Black tree from any node to its leaves have the same number of black nodes, this fact keeps the tree height short and increases the breadth of the tree. This is in turn makes the tree to grow horizontally and increases the performance of the tree. When all leaf nodes of a tree are at the same level, it is said to be perfectly balanced. The performance of a tree depends on how perfectly balanced it is.

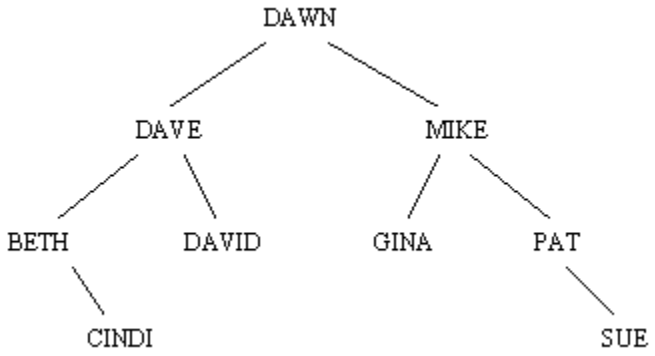
Red Black tree leaf nodes are called sentinel nodes, contain null values, and are always black. Sentinel nodes are not always displayed but they are implied in Red Black trees. A sample Red Black tree figure is shown below:



3.10: Red Black Tree

Red Black trees perform well in time bound applications. Red Black trees are used in applications where the data changes constantly. Since our forensics application deals with large amount of data to be indexed and stored inside the index file, red black tree data structure is of utmost importance to perform faster and effective indexing and storage of data.

In our forensics application, indexed strings and their offsets are stored as nodes in a tree and the tree stores them in their natural order. If we go on indexing the line “DAWN DAVE MIKE BETH DAVID GINA PAT CINDI SUE” and try to display the index file using a tree structure, we get the index file as:



3.11: Keywords inside the Index File

If one tries to read all the words from the index file, the words are read out in a sorted order as: BETH, CINDI, DAVE, DAVID, DAWN, MIKE, GINA, PAT and SUE. The above diagram demonstrates how keywords are placed in the index tree and how they are dumped into the index file in a sorted manner once the indexing is complete.

3.3.5 Searching Component

Searching Component is used for describing how the search task is carried out by the toolkit. It also mentions other search features implemented by the toolkit.

The search functionality which has been implemented to search the index file for possible keywords is very simple. First the index file is searched for a particular keyword and if the keyword is found, its position and count are found out from the index file and then the keyword is printed with its position and count. The keyword is searched using Binary Search and it does not matter how big the index is, the keyword is found out in the index file in as few as 7 or 8 steps. So the search for a keyword in an 80 GB forensics disk image file takes the same amount of time when it is searched against a 20 GB forensics disk image file. Binary Search does not exist in regular expression searches because in regular expression, one has to search for patterns in the index file comparing all words one by one and the search time depends on the index file length.

3.3.5.1 Binary Search for Non-Regular Expression Search

Binary Search for Non Regular Expression Search is the best bet for getting search results quickly because it does not matter how big the index is, the keyword is found in the index file in as few as 7 or 8 steps. Thus the search for a keyword in an 80 GB forensics image file takes the same amount of time as against a 20 GB forensics image file. Binary Search is not implemented for Regular Expression Searches because in regular expression all entries in the wordlist are compared to find the search item. Binary Search is faster, takes less time, and is most suitable for non regular expression searches. Binary Search should be less than a second in most cases as we have to deal with 7 or 8 steps in all cases irrespective of the dataset size.

Design and Implementation

For non regular expression search, firstly the specified keyword is found using binary search method and then its offset is determined. That offset is used to find all occurrences of the word in the index file which in turn displays the results in an interactive way. The design and pseudo code of the binary search component is described below.

For example, consider a search string “CreateStrokes” which has to be searched inside the forensics image to find its number of appearances and context around each hit.

The file denoting Start String and End String Number is described as follows where Start String is the first indexed string and End String is the last indexed string inside the forensics image:

IndexFileNameStartEnd.dat (File denoting Start String and End String Number)

- 1 74903 (Start String Number = 1 and End String Number = 74903)

Performing Binary Search to find the search string “CreateStrokes” using String Number 1 and String Number 74903 as two input values, we get the String Number 7 as the search string number since it holds the search string “CreateStrokes”.

The file denoting String Number and its Offset is described as follows:

IndexFileNameNumberOffset.dat (File denoting String Number with its Offset Number)

- 1 …
- 2 …
- … …
- 7 31 (String Number = 7 and Its Offset Number= 31)
- 9 51
- 13 108
- 15 138
- … …
- … …

We find that the search string “CreateStrokes” (String Number 7) has got Offset Number 31. This Offset Number is the offset where the search string “CreateStrokes” can be found with other important details inside the index file.

The index file is described as follows:

IndexFileName.dat (Index File)

- ….

- CreateStroke 2 39565 39968 (String = CreateStroke, Times Appeared = 2)
- CreateStrokes 2 39954 39955 (String Positions: 39954 and 39955)
- CreateTextServices 1 558198

•….

The search string “CreateStrokes” is found out traversing the same offset from the beginning of the index file. The search string “CreateStrokes”, its number of appearances (2) and its context around all places/positions (String Positions: 39954 and 39955) are printed.

39954: CreateStrokes Found
I am going they are not coming I am loving them CreateStrokes car lollypop
39955: CreateStrokes Found
CreateStrokes I am fine and they are good for the sake of I am going there
CreateStrokes Found 2 Times

(Pseudo Code)

Read the file “StartString-EndString.dat” which mentions Start String and End String Number (Start String is the first indexed string and End String is the last indexed string inside the forensics image)

Find Start String and End String Number

Do Binary Search on these Start String and End String Number till you get the Search String

If Search String is found

{

Find the Search String Number

Read the file “StringNumber-OffsetNumber.dat” which denotes String Number as well as its Offset Number

Using the Search String Number, find the Offset Number in “StringNumber-OffsetNumber.dat” file

Locate the Search String inside the Index File using the Offset Number and Print all its details: String Name, Number of Times it Appeared, Places it Appeared etc.

}

If Search String is Not Found

Quit the Program

3.3.5.2 Linear Search for Regular Expression Search

Linear Search is implemented for Regular Expression Search because all entries in the wordlist are to be compared to find the search item. Linear Search is not so fast, takes more time because it has to compare all entries, and is most suitable for regular expression searches. Linear Search time depends upon the dataset size but care should be taken to make it more efficient using appropriate algorithms.

Design and Implementation

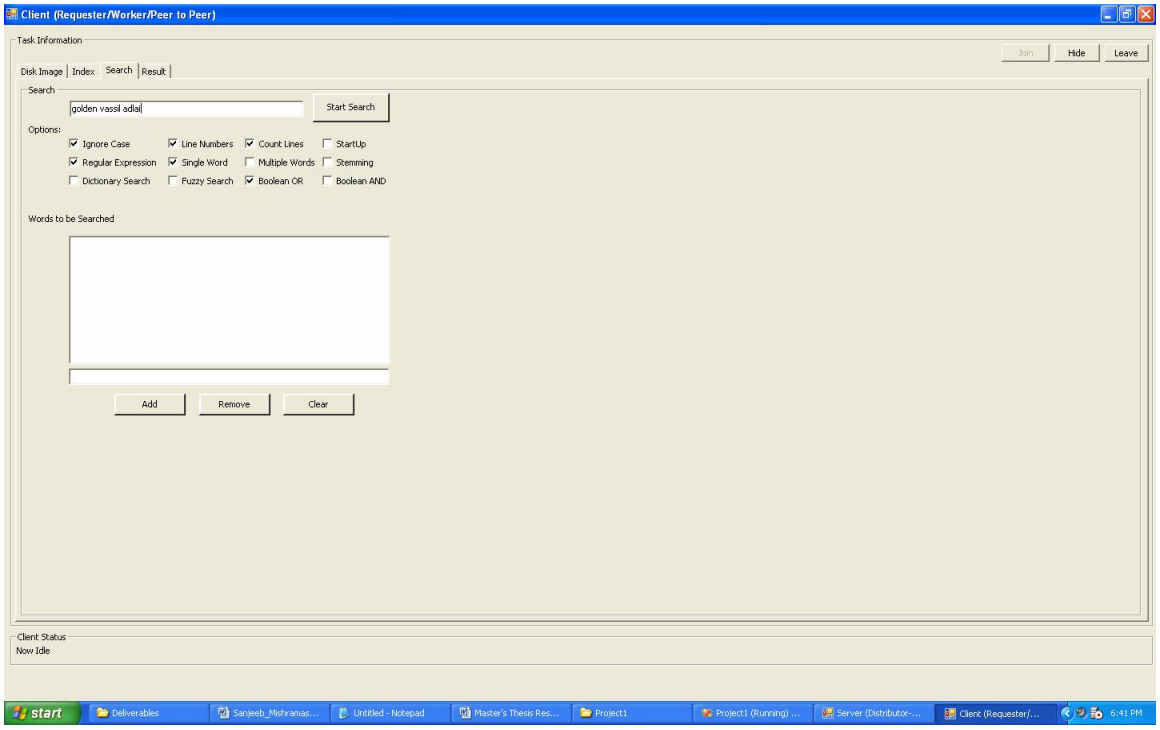
For Regular Expression search, a pattern is created first and that pattern is compared to all strings inside the disk image one by one and if there is a match for the pattern, that string is added to the search list. The pseudo code for the regular expression search module is described below.

(Pseudo Code)

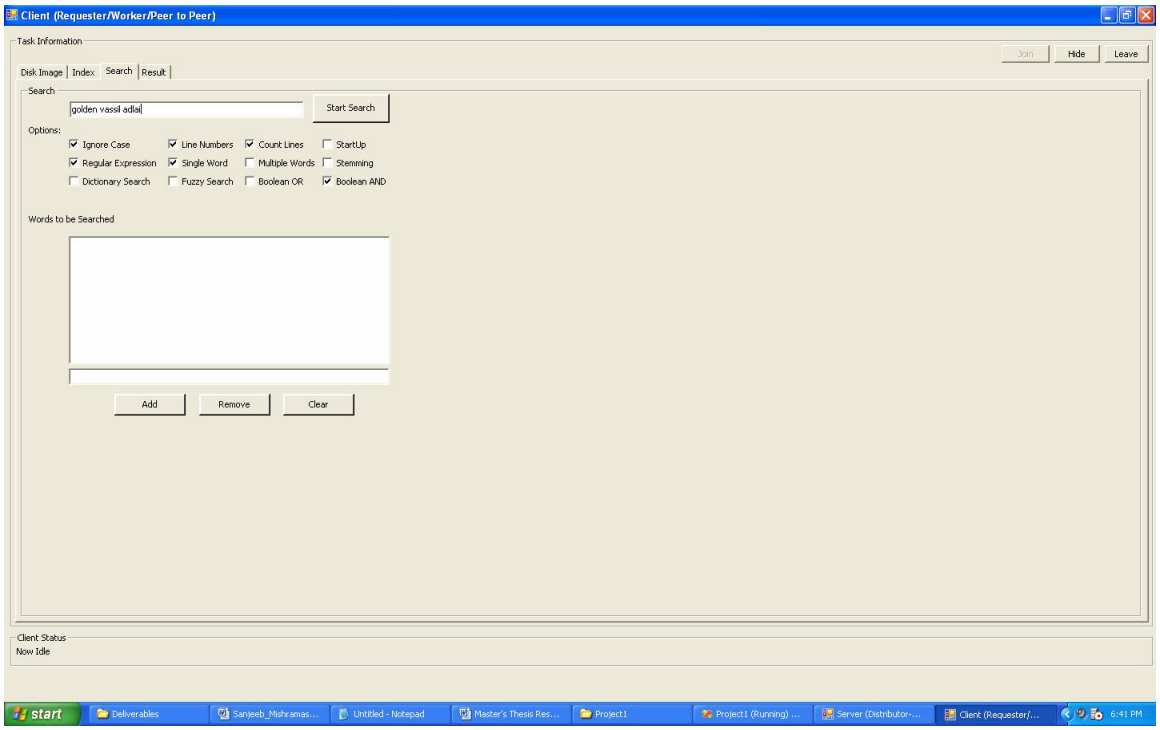
```
Declare and define a string named pattern to hold a regular expression pattern
For each string inside the disk image
{
If the pattern string matches the string inside the disk image
{
String matches pattern
        }
        Else
        String does not match pattern
}
```

3.3.5.3 Various Search Features

The search features implemented in the forensics application make the search hits more probable. Boolean search of two or more keywords is implemented. Boolean search such as “and” and “or” has been implemented implicitly. To ignore case so that case insensitive search can be performed on the keyword, ignore case feature has been implemented.



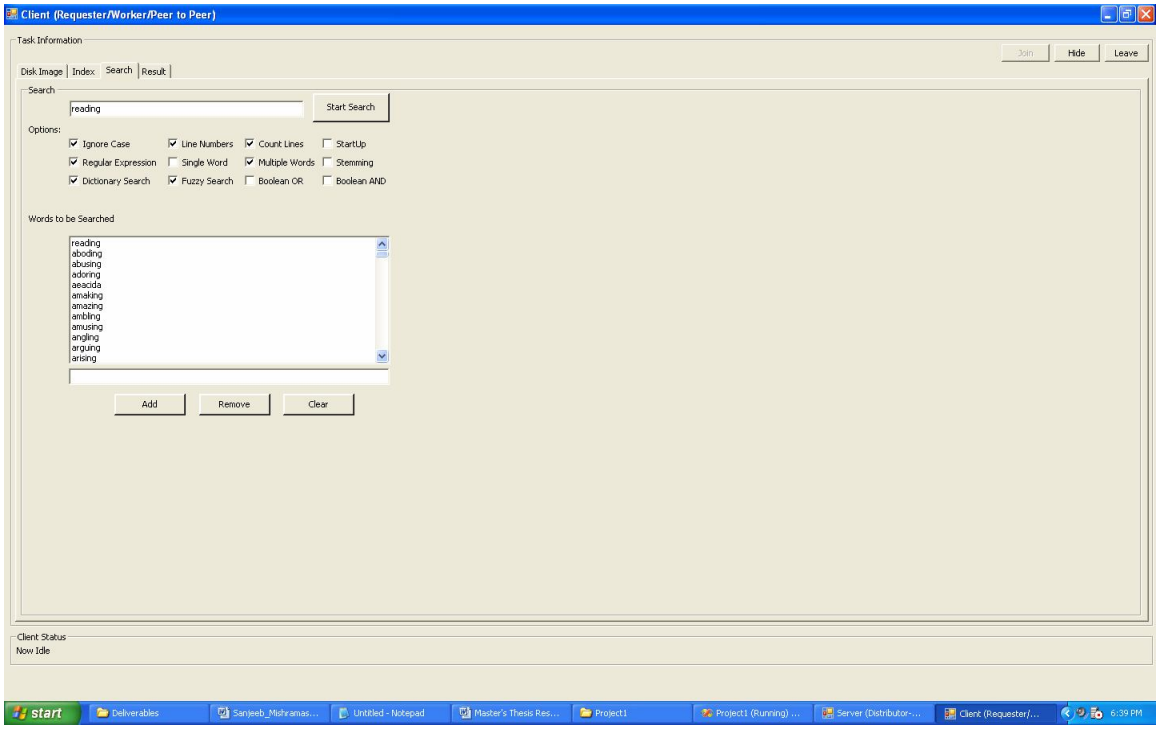
3.12: Boolean OR “golden vassil adlai”



3.13: Boolean AND “golden vassil adlai”

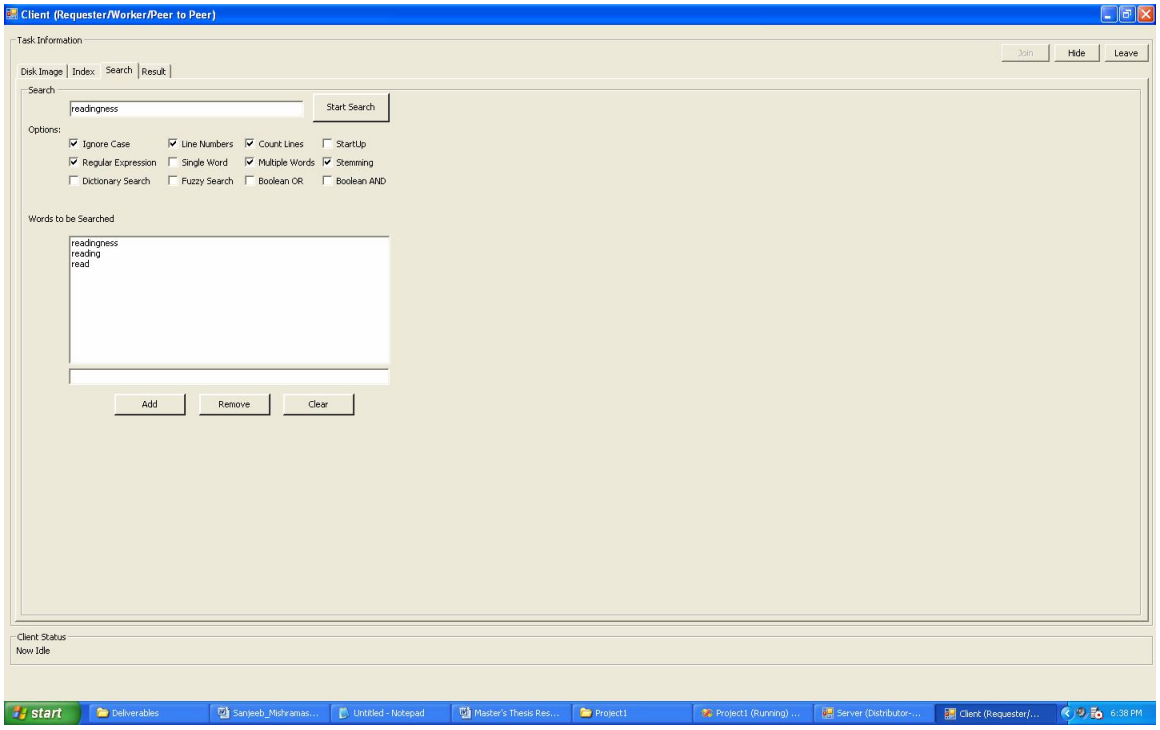
Fuzzy search, stemming, and thesaurus search are carried out to increase the keyword hits while searching for keywords. Fuzzy search is used to find all words which are very close to the misspelled keyword. Stemming is used to find all words of the given keyword which are its origin and thesaurus search is incorporated to find all similar words of the given keyword.

Fuzzy Logic is used for fuzzy search. When the search words are misspelled, fuzzy search returns the appropriate words, which are very close to the misspelled keywords [12].



3.14: Fuzzy Search lists all words close to “reading” ending in “ing”

Stemming searches for root words of the given keyword. It transforms the word into its root forms and adds all the stem words to the list.



3.15: Stemming of the word “readingness”

Thesaurus Search includes all the words which are similar in meaning to the search word and adds all the similar words to the search list.

3.3.6 Other Features Implemented

Many other features are implemented including byte to string conversion to convert byte data to string data, Unicode encoding to encode to Unicode format, hex editor implementation to display hexadecimal contents, string to byte conversion to convert string data to byte data. To convert a byte to string, byte to character conversion is carried out first and then all characters are combined together to create the string. Unicode encoding is used for foreign language investigations as ASCII does not represent all character sets. An Alphabet file, a user generated file, is used to print non-printable characters using printable character sets. Non-Printable characters are represented by “.” character in our solution. Hex Editor is used for printing the hex contents of a particular line in disk image in case one wants to do further investigation or wants to display the hexadecimal contents of parts of that disk image. Saving Memory Stream feature is used for saving the memory stream to a file. String to byte conversion feature takes a string, reads all characters in the string, and converts all characters to its respective byte representation.

Apart from these features, many other application specific operations are carried out such as deleting the index file and making the application to restart after the client shuts down. When “clean the index” button is pressed, the stored index file is deleted. Once the index file is deleted, one has to create a new index file to search for keywords. When the client restarts, there are two things which can happen. If the index file exists, the client will search for keywords in the existing index file and if the index file does not exist, the client/worker will go on indexing the image and then search for keywords in the new index file.

Chapter 4. RESULTS AND PERFORMANCE FIGURES

For a 20 GB Image File, the index and search results are as follows:

Number of Clients	1	2	3	4	5
Total Index Time (Image Copy Time + Index Time)	4 Hr 44 Min	2 Hr 15 Min	1 Hr 50 Min	1 Hr 40 Min	1 Hr 36 Min
Keyword Search Time (Secs)	< 1	< 1	< 1	< 1	< 1
Regular Expression Search Time (Secs)	< 1	< 1	< 1	< 1	< 1

For a 20 GB forensics image, we see that with the increase of client computers the total index time which is a combination of image copy time and index time, reduces depending upon the number of clients.

We have, Total Index Time= Image Copy Time + Index Time, where Image Copy Time is the time it takes to copy chunks of original forensics image to individual clients connected to the server and Index Time is the time it takes to index the image file chunks for individual clients. The search time is less than 1 sec as we have incorporated binary search for keyword searches and the regular expression search is also around a few seconds to list all regular expression keywords.

For an 80 GB Image File, the index and search results are as follows:

Number of Clients	3	4	5
Index Time	4: 30 hr	3 Hr	2. 20 Hr
Keyword Search Time (Secs)	< 1	< 1	< 1
Regular Expression Search Time (Mins)	< 1	< 1	< 1

For an 80 GB forensics image, we see that with the increase of client computers the index time reduces depending upon the number of clients. The search time is less than 1 sec and

the regular expression search is also around a few seconds to list all regular expression keywords.

Chapter 5. CONCLUSIONS

We have many digital forensics toolkits (FTK, DNA, Encase etc.) available in the market now, but when we mention distributed digital forensics toolkits performing indexing and searching of disk images, there is no available prototype. Our implementation is one of a kind and is based on Distributed Computing using Microsoft .NET Remoting.

Forensics toolkits for keywords indexing and searching available in the market now support only single desktop processing and are very slow for indexing as well as simple search operations. Although Non-indexed search operations are very slow, even with indexed searches without the use of networked systems, the search activities are considerably slower and index time is enormous. There are no commercial distributed digital forensics toolkits available for indexing and searching dealing with large gigabytes of forensics data. The designed distributed forensics toolkit here is faster and, since we deal with a number of computers rather than a single machine, this toolkit raises the forensics analysis procedure to new heights. Rather than having a cluster of machines, we have dealt with Distributed Grid Computing to build a system, where we have disparate systems running different OSs and these systems performing as workers to carry on with a portion of the assigned task.

When we created the index file using multiple computers, the average index file creation time decreased while the size of individual index files created by respective workers reduced considerably. Therefore, when we deal with large gigabytes of disk images, index file creation with single workstation is time-consuming. Use of multiple systems is needed as they save a lot of time and index file is also created at a faster speed depending upon the number of machines or workers available.

For a 20 GB forensics image, we see that with the increase of client computers (1, 2, 3, 4 and 5) the total index time, which is the combination of image copy time and index time, reduces (4 Hr 44 Min, 2 Hr 15 Min, 1 Hr 50 Min, 1 Hr 40 Min and 1 Hr 36 Min) depending upon the number of clients where Image Copy Time is the time it takes to copy the chunk of original forensics image to individual clients. Index Time is the time it takes to index the supplied image file chunk. The search time is less than 1 sec as we have incorporated binary search for searching the keywords and the regular expression search is also around a few seconds to list all regular expression keywords.

For an 80 GB forensics image, we see that with the increase of client computers (3, 4, and 5), the index time reduces (4.30 Hr, 3 Hr, and 2.20 Hr) depending upon the number of clients. The search time is less than 1 sec and the regular expression search is also around few seconds to list all regular expression keywords.

While creating the index file, filters were added to make the index file consume less storage and dictionary files were used for index file creation. Although it took a little more time for index file creation, due to fewer words in the index file, its size was decreased and it consumed less storage. Due to the use of filters and dictionary files, small sized index files were created and keyword searches happened relatively faster.

Without this one would have got the same size index file as the forensics image file. Two types of filters were maintained while creating index files: dictionary file and noise word list file. Dictionary file is a list of keywords which describes a list of words to index rather than indexing the whole disk image. Noise word list file describes a list of words which are not indexed in the index file as they are used for noise reduction in the index file, e.g., me, this, he, she etc. Indexing techniques are vital to the success of text analysis since they provide speed and overall efficiency. Index files were opened as binary ones so that keyword searches would be faster.

In order to get tremendous speed in indexing, proper indexing techniques should be chosen. When image file was distributed among available workers for index file creation, overlapping of image contents (Offsets) were considered so that all words in the disk image were chosen for indexing and no words were left unnoticed which would have been the case had one simply divided the image file and had sent the parts to available workers for indexing.

Although Sleuth Kit and Autopsy are great forensics toolkits available in the market now, they lack a lot of key features while performing text analysis. “grep” and “strings” commands in Autopsy and Sleuth Kit search commands are not able to match up with the sophisticated text analysis tools available in the market now as they lack a lot of high end features, which are required for speed and overall efficiency for text analysis. Sleuth Kit lacks text analysis based indexing which makes the life of an examiner difficult because every time he hits the search button, a new search is generated and he has to wait long enough to get the search results when the data to be searched is of thousands gigabytes of length.

While searching for indexed words, we tried to increase the hits of those words in index file using stemming, fuzzy search, thesaurus search and boolean search. Fuzzy search enabled search for misspelled words. Thesaurus search specified all synonyms or similar words for the given search word. Stemming added more words to the search list as it always refers to the root words of the search word. Boolean search added combination of words to the search list. Regular expression and normal search operations also worked superbly.

Rather than having single word search, we implemented multiple words search which increased productivity. During the time these tasks were running on the foreground, background tasks were performed by the user. We also tested Unicode word search, which can be possible for image files in Unicode format, which is required for investigation in foreign countries where the language of computer is not English but its culture specific language.

PVM (Parallel Virtual Machine) and MPI (Message Passing Interface) have their own architecture for how to make an application distributed, but utilizing the architectural design this toolkit adheres to, any application can be customized to be distributable if it allows for task parallelization.

Chapter 6. FUTURE WORK

New algorithms can be added for faster indexing and unnecessary code can be removed to avoid extra overhead. Indexing techniques are vital to the success of text analysis as they provide greater advantage through their speed and overall efficiency.

Various search features and new algorithms for fuzzy logic, stemming, and thesaurus search can be added to make the keyword searches more robust. By eliminating overhead code, these algorithms can be optimized to be more efficient. In order to expand the search capabilities further, one can think of adding more search features such as natural language processing, phonic searches, and text categorization.

The current system runs only on windows machines. To run it on Linux machines, one has to install VMware on Linux host and inside that he has to install Windows OS as guest OS. To run the application inside Beowulf Cluster of machines, he has to install VMware followed by Windows OS in all cluster processors and then he has to copy the VMware image file to all machines one after another. This is not only time-consuming, but the installation and setting up phase is also tiresome on the part of even a seasoned network administrator.

In order to make it widely accepted, one should try to make it run on .NET Mono, the Open Source Implementation of .NET. Using this, he will be able to run the system using any machine not specific to any OS. Rather than running the forensics application using Visual Studio.NET 2005 for Windows, he can run the application using .NET Mono for Windows and .NET Mono for Linux at the same time. For the forensics application to run in .NET Mono, the application has to go through many code modifications from Visual Studio.NET to .NET Mono. .NET Mono also exists for other UNIX flavors and support for new flavors are evolving everyday. .NET Mono is funded by Novell Corp. and its wide-spread marketing is underway to make it run on any OS platform [29].

For Unicode search, many language packs should be added to search for keywords in foreign languages. After Unicode search features have been added, testing should be carried out by respective language experts or linguistics for error detection, which might have crept into the system due to ignorance on the part of the programmer while trying to handle foreign languages. Keywords can be binary in nature in foreign languages using Unicode and efficient search mechanisms should be employed to tackle this situation. While using Unicode, various strategies should be implemented to change the internal structure of the index file that recognizes different Unicode supported languages otherwise it will be of very large size due to their Unicode structure.

The system is designed to run only in department or in-house networks. Using XML Web Services, it can be made to run anywhere with forensics investigators or researchers doing online collaboration while searching for keywords. If one is able to do this, then he will have a running version of a web application operating exclusively on XML Web Services.

To effectively utilize the idle CPU time, Multithreading can be utilized on available client or worker machines if they run on Intel Core Dual or Other High Speed Multithreading Processors. Multithreading makes index and search time faster.

Chapter 7. REFERENCES

[1] Computer Forensics
http://www.louisville.edu/speed/cecs/research/compforensics/index.html

[2] Resource Sharing Over Wireless Grid
http://www.eecs.tufts.edu/~hchang/wirelessgrid.html

[3] IEEE Distributed Systems Online
http://dsonline.computer.org

[4] Achieves DS Online IEEE
http://dsonline.computer.org/archives/index.htm

[5] Grid Resources
http://search2.computer.org/advanced/simplesearch.jsp

[6] Grid Reading Topics
http://www.ece.rutgers.edu/~parashar/Classes/02-03/ece572/readings-grid.html

[7] Grid Computing Course Online
http://www.cs.wcu.edu/~abw/CS493F04/

[8] Programming and Design Grid Computing
http://dps.uibk.ac.at/index.pl/grid

[9] Grid Application Samples
http://www-106.ibm.com/developerworks/grid/library/gr-exist/?ca=dgr-lnxw16EnableGridApp

[10] Keywords Indexing & Searching
http://pyflag.sourceforge.net/Documentation/articles/indexing/index.html

[11] Microsoft .NET
http://msdn.microsoft.com/netframework/

[12] Fuzzy Logic C# Sample Application
http://www.codeproject.com/csharp/fuzzy_dot_net_fuzzy_word.asp

[13] Thesaurus Search
http://www.codeproject.com/csharp/spellcheckparser.asp

[14] Porter Stemmer
http://www.tartarus.org/~martin/PorterStemmer/csharp.txt

[15] Grid Computing Class Notes

http://www-csag.ucsd.edu/teaching/cse225w03/cse225-readings.html

[16] Grid Computing Sample Code
http://www.java201.com/resources/browse/45-2004.html

[17] Grid Computing Sample Server Java Code
http://www.cs.binghamton.edu/~mgovinda/courses/introToGridComputing/assignments/assign1/Server.java.html

[18] Grid Computing Sample Client Java Code
http://www.cs.binghamton.edu/~mgovinda/courses/introToGridComputing/assignments/assign1/Client.java.html

[19] Grid Papers
http://apples.ucsd.edu/hetpubs.html

[20] IT Research Grid Computing
http://ostg.bitpipe.com/

[21] Alchemy Documentation Grid Computing
http://www.alchemi.net/doc/0_6_1/index.html

[22] Resource Sharing Over Wireless Grid
http://www.eecs.tufts.edu/~hchang/wirelessgrid.html

[23] Code Project
www.codeproject.com

[24] MSDN Library
www.msdn.com

[25] C5 Code Documentation
http://www.itu.dk/research/c5/

[26] Dictionary Code Sample
http://www.codeproject.com/useritems/Dictionary.asp

[27] Research Grid Computing
http://mrccs.man.ac.uk/research/grenade/

[28] Search Tree C# Implementation
http://www.codeproject.com/vb/net/searchtree.asp

[29] .NET Mono
http://www.mono-project.com/Books

[30] Grid and Peer to Peer Information
http://dsl.cs.uchicago.edu/

[31] Grid Thesis Report
http://www-d0.fnal.gov/computing/grid/papers/Thesis-A.S.Rana.doc

[32] Grid Thesis Report
http://www.caip.rutgers.edu/TASSL/Thesis/MsThMain_vincent.pdf

[33] Ian Foster Presentation Future of Grid
http://csce.uark.edu/~aapon/courses/gridcomputing/notes/IanFoster2004-9-14.ppt#23

[34] Grid Computing
http://wwwd0.fnal.gov/computing/grid/

[35] Wireless Grid Information with complete implementation
http://www.eecs.tufts.edu/~brchen/wirelessgrid.html

[36] Grid Information
http://www.cs.unc.edu/~alok/

[37] Grid Information
http://informatik.uibk.ac.at/users/c703251/bachelor.html

[38] Publications Thesis
http://www.logos.ic.i.utokyo.ac.jp/phoenix/publications.shtml

[39] Publications Thesis
http://hst.home.cern.ch/hst/publications.html

[40] Research Thesis
http://www.cise.ufl.edu/~ppadala/research/thesis/

[41] Wireless Grid
http://www.eecs.tufts.edu/~brchen/wirelessgrid.html

[42] Publications on Grid
http://www.cs.nwu.edu/~pdinda/papers.html

[43] Publications on Grid
http://hst.home.cern.ch/hst/publications.html

[44] C# Notes
http://dynamo.ecn.purdue.edu/~cath/ee647/notes.html

[45] Code Project

www.codeproject.com

[46] Microsoft
www.microsoft.com

[47] MSDN Library
www.msdn.com

[48] MIT Bayanihancomputing
http://bayanihancomputing.net/BayanihanAsiaStudentPaper.pdf

[49] Sample C# Code
http://www.fsl.cs.sunysb.edu/docs/sca/node3.html

[50] Code Project Wingrep Search Sample Code
http://www.thecodeproject.com/csharp/wingrep.asp

[51] Sample Code
http://www.daimi.au.dk/~gerth/webalg02/

[52] dtSearch Indexing
http://www.dtsearch2.com/webhelp/dtengine/default.htm#indexing_options.htm

[53] dtSearch Indexing Materials
http://support.dtsearch.com/webhelp/dtsearchCppApi/frames.html?frmname=topic&frmfile=Creating_an_index.html

[54] C# Materials on C# Generics
http://www.dina.kvl.dk/~sestoft/gcsharp/index.html

[55] Research Paper on Grid Computing
http://www.dfrws.org/2004/bios/day2/Golden-Perfromance.pdf

[56] Keyword Indexing & Searching of Forensics Images
http://pyflag.sourceforge.net/Documentation/articles/indexing/index.html

[57] Text Analysis Master's Thesis
http://www.fukt.bth.se/~uncle/papers/master/thesis.pdf

[58] Article on Keyword Indexing & Searching of Forensics Images
http://pyflag.sourceforge.net/Documentation/articles/fuse.html

[59] Microsoft Express Editions Downloads
http://msdn.microsoft.com/vstudio/express/

[60] Microsoft Developer Centre

http://msdn.microsoft.com/developercenters/

[61] Microsoft Downloads
http://msdn.microsoft.com/downloads/

[62] MSDN Library
http://msdn.microsoft.com/library/default.asp

[63] C# Generic
http://www.dina.kvl.dk/~sestoft/gcsharp/index.html

[64] C# Generic Documentation
http://www.itu.dk/research/c5/Release1.0/c5doc/frames.htm

[65] C# Book
http://www.dina.kvl.dk/~sestoft/csharpprecisely/

[66] Red Black Tree Creation C#
http://www.codeproject.com/csharp/redblackcs.asp

[67] Tree Dictionary
http://www.koders.com/csharp/fidF2CAD8075C711A9E6AC6F9E797F1D94706BB26C7.aspx

[68] Indexing Example
http://www.brainspark.nl/articles/searchtools-introduction

[69] Peer to Peer Computing with VB.NET
http://www.apress.com/author/authorDisplay.html?aID=111

[70] Red Black Trees
http://www.codeproject.com/csharp/redblackcs.asp

VITA

Sanjeeb Mishra was born on March 13th 1978 in Bhubaneswar, Orissa, the temple city of India. He graduated from Bangalore University, Bangalore, the Silicon Valley of India and worked as an Intern at DIGITAL Equipment/COMPAQ Ltd. Bangalore, in C/C++ Programming Group.

He joined University of New Orleans for Master of Science in Computer Science with high academic scores and despite all his efforts was unable to find any school jobs but landed directly in company internship jobs.

Now he works as a C# .NET Programmer Specialist with Programming expertise in C# WinForms, ASP.NET Web Forms, ADO.NET, and VC++.NET.

He is Specializing in Microsoft related Technologies and Tools and has got expertise in .NET Enterprise Application Development(.NET Remoting, XML Web Services, SOAP, UDDI, WSDL), Web/E-Commerce cum Database Development (SQL Server 2000 and Oracle 9i/10g), and Systems Software Development (C/C++/VC++).

He has Volunteered Programming Assignments for various organizations in New Orleans and is highly admired by his employers for his motivation, dedication, and commitment to his work. He is highly popular among his friends and juniors.

He plans to pursue PhD and MBA in the near future and would like to return to his home country to realize his other dreams.