The SPYDER

(Gnutella Crawler)

EECE 411 Project – Phase 3

Cheuk Dong	17113036
Andrew Wong	19814037
Ying Yin	75821041

Table of Contents

1.0 Introduction

In this part of the project, we extend the Gnutella network crawler from phase 2 to harness multiple crawler nodes and employ a master/worker approach with non-blocking I/O on the worker side.

Starting from the same bootstrapping point the Spyder will crawl the Gnutella network as fast as possible for a specified period of time and collect information about the nodes participating in the network and the files shared.

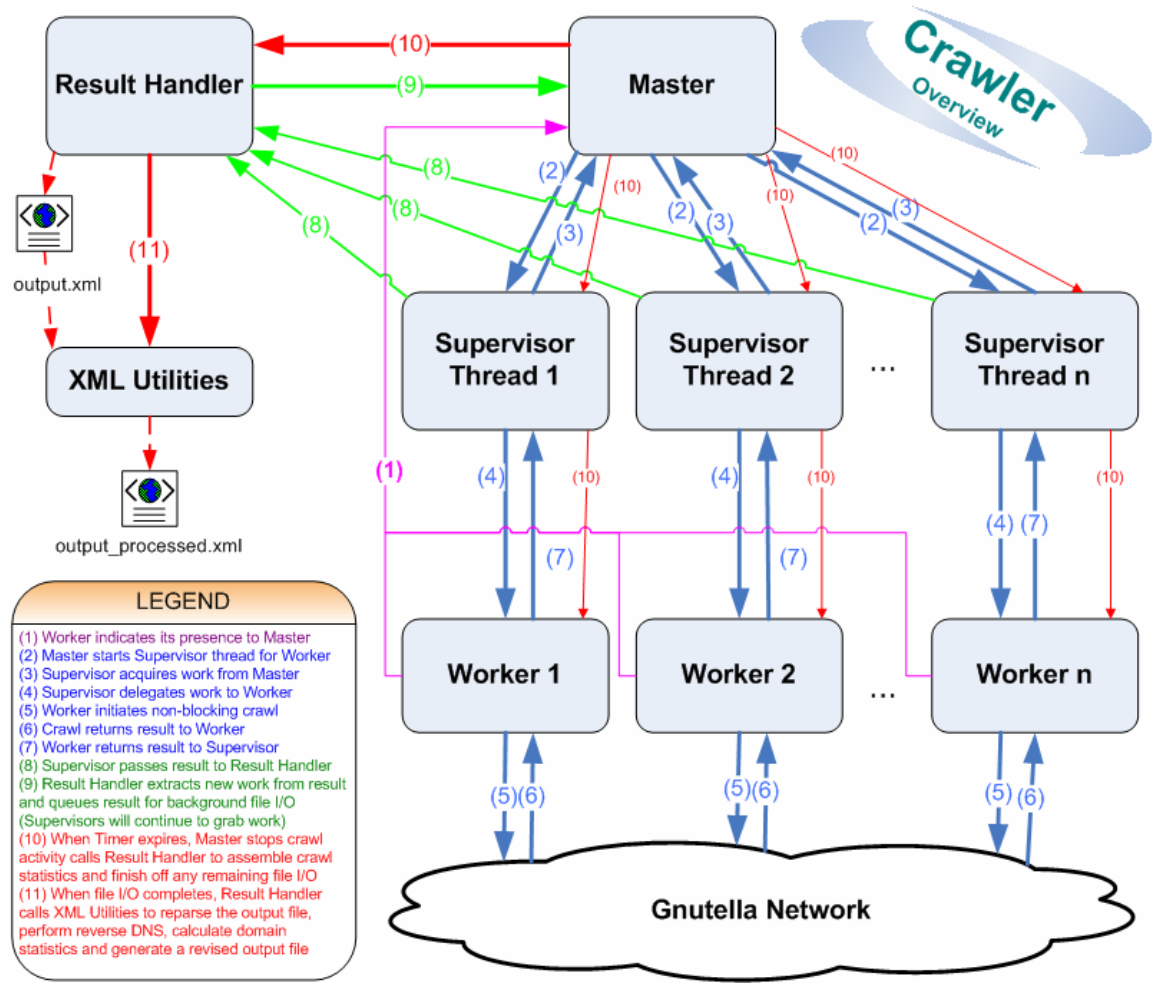
We run the Spyder for 30 minutes and it collects the same information about each Gnutella node as in the previous phase of the project. We will be providing performance reports about the number of nodes crawled, average nodes crawled per hour, statistics on all nodes crawled, and performing reverse DNS.

2.0 Spyder Features

- ***Non-blocking I/O*** is used for both crawling the Gnutella network and connection from worker to master
- ***Modifiable Timer*** that can issue immediate stop command to master and workers
- ***Dynamic work distribution*** between Master & Client using *"Server-Client Design Pattern"* to maximize performance and promote scalability
- ***XML*** format outputs (formatted with ***XSL/CSS*)** for data analysis and easy viewing in either ***IE or Firefox***
- ***Post-execution XML parsing*** and ***reverse DNS***
- ***Graphical User Interface*** for Master to promote usability
- ***Python Script*** to initialize 50+ workers on PlanetLab nodes

3.0 System Overview

Our overall system design leverages Java non-blocking I/O and multithreading capabilities as much as possible to maximize crawl efficiency and performance. A number of features and tweaks have also been included to optimize the crawl process. An overview of the system is shown below.



In the beginning, the master starts up (either through the GUI or through command line), initiates a timer thread and crawls the initial node while waiting for workers to connect. Individual worker nodes, when ready, will indicate their presence to the master, which will then create a new “supervisor” thread to handle communication with the worker. These supervisor threads help ease the load on the master so that it can concentrate on distributing work and overlooking the entire crawl process.

Each supervisor acquires work (a number of nodes to crawl) from the master and passes this on to its corresponding worker node. The worker node initiates a number of non-blocking crawl calls to the Gnutella network and passes individual results back to the supervisor as they arrive. Upon receiving a result, the supervisor passes it to a result handler thread, checks if its worker has enough work and, if not, acquires more work from the master to delegate to the worker.

The result handler thread is unique design choice in itself. This thread specializes in processing crawl results. All incoming results are dumped in a queue to enable synchronization (since the list of pending work must be synchronized anyway). The result handler repeatedly obtains a result from the queue, extracts any new work (nodes to crawl), passes non-duplicate work to the master for distribution and writes the result to file. By default, results are converted into valid XML and then written to file, but an option is available to write a plain text file.

When the crawl timer expires, the master notifies all the supervisors to stop all crawl activity and then tells the result handler to handle ending procedures, which include writing the remainder of the queued results to file and generating overall crawl statistics. When finished, the result handler will then call an XML utilities class to reparse the XML output file, perform reverse DNS on the IP addresses of all crawled nodes in the file, generate domain statistics and finally write a revised XML output file.

4.0 Design Choices and Decisions

4.1 "Server-Client" Design Pattern

"Server-Client" design pattern is used between Master (Server) and Worker (Client) to maximize the performance of our crawler.

Reason for Decision:
The reason why we used this design is because the master can start up a server socket and wait for all clients to join. When installing and running the worker (client) on 50 or more of the PlanetLab nodes, the worker only needs to know the IP of the master, as well as the port it is waiting for connection on. This implementation is optimal because each worker will only need the same IP/port-number. If the master location was to change, only a small modification is necessary to change the connection.

This promotes scalability as it is very easy to add more workers to our system. This also enables the master to handle worker failure better as the master can integrate workers that join or fail dynamically during the computation.

4.2 Non-Blocking I/O on Worker Side

Non-blocking I/O is used for both crawling the Gnutella network and connection from worker to master

Reason for Decision:
Non-blocking I/O has better performance than multi-threading because it doesn't have overhead for context switching.

4.3 Use of Attachments

Because we are using non-blocking I/O for our worker, a nice feature in Java is to allow attachments to be added to help us keep the state of our NIO implementation.

Reason for Decision:
The use of attachment makes the implementation of state machine for the non-blocking I/O at worker easier. For each connection to the Gnutella node, there is a Crawler attached to the SocketChannel. The Crawler keeps a CrawlerState enum to track its state and perform the appropriate actions for sending requests and receiving responses (see Figure 4.1). For the SocketChannel to the master, there is a WorkLoad attachment to the channel that keeps the state and the nodes to crawl and the CrawlResult to be sent to the master (see Figure 4.2).

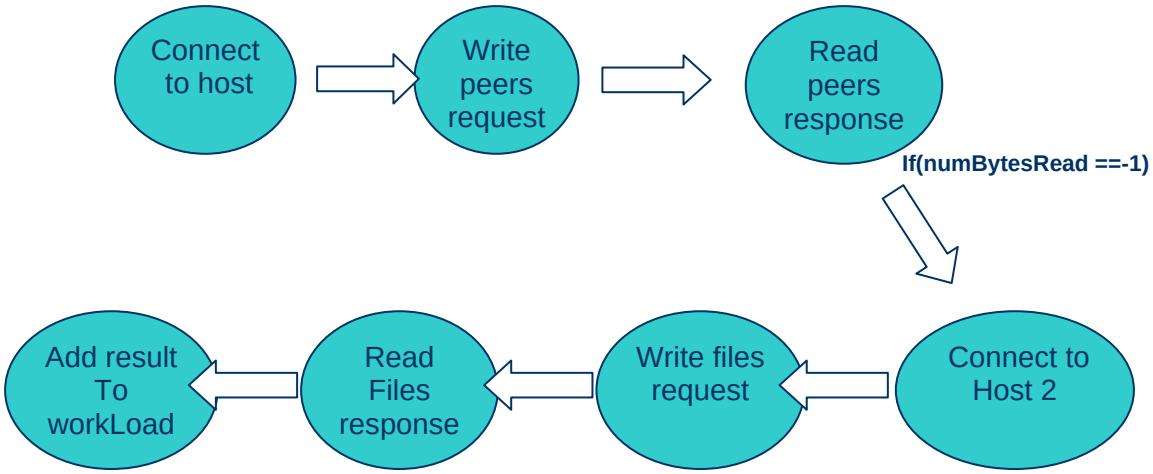


Figure 4.1: State diagram for Crawler attachment

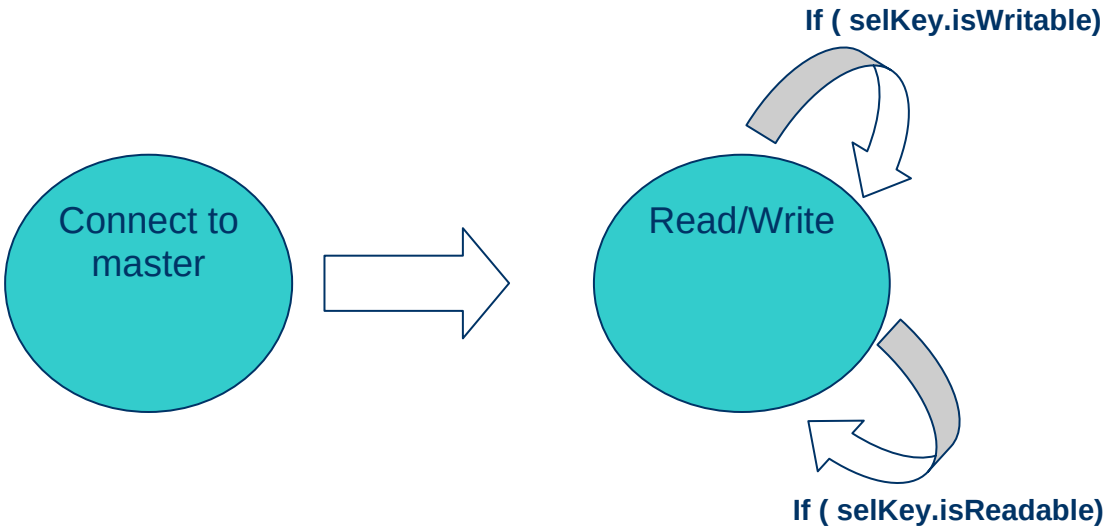


Figure 4.2 State diagram for WorkLoad attachment

4.4 Modifiable Timer for Stopping Crawling

We have a modifiable timer in our system that can issue immediate stop command to master and workers.

Reason for Decision:
The reason we have a modifiable timer is that because 1) it is required in the specification, and 2) it allows us to easily specify the length of time we wish to crawl. With the help of the GUI, this timer also allows us to stop the crawl at any time during the crawl process, and we can quickly obtain the performance of our system and make adjustments to optimize accordingly.

4.5 Avoiding Cycles

The worker doesn't decide which nodes to crawl. It only crawls the nodes sent by the master and returns the new list of peers and leaves. The master keeps a nodesCrawled list and a nodesToCrawl list, and it checks the two lists before adding new nodes to the nodesToCrawl list to prevent cycles.

Reason for Decision:
As we use multithreading, that list may be checked and updated by threads concurrently. Hence, the block of code that checks the list and adds new element is synchronized. nodesToCrawl is a ConcurrentLinkedQueue which provides good concurrency.

4.6 Dynamic Work Distribution

When distributing the work for the various workers that connect to the Master, we use a dynamic work distribution method where we spawn a thread for each worker connection. We then issue to each of the connected worker up to 100 nodes to crawl at a time.

Reason for Decision:
Maximize performance by sending each worker a workload of 100. This will ensure that the worker will always have nodes to crawl, but will not have so much work to do that their workload will accumulate. We do not want a situation where the worker output is slower than the master input. The maximum workload is adjustable, and 100 is the highest we tested.

Because the maximum workload distributed to each worker can be changed easily, we can promote scalability because if we find out that the workers are running into a situation where it is crawling faster than the supervisor's work sending rate, we can easily increase the workload limit and gain performance. Likewise, if we find that the workers are working slower than the work sending rate, we can reduce this limit. This change can be based upon the amount of workers we have connected to the master.

4.7 Initial Crawl Performed by Master

The master launches a temporary thread to crawl the first node while waiting for workers to connect.

Reason for Decision:
At the beginning of the crawl process, a significant amount of time is spent waiting for worker nodes to connect to the master. We quickly realized that this down-time could be put to better use by launching a temporary thread to crawl the first node. This thread would then dump its results into the result handler and more work would be available sooner for distribution to the first worker node that connects. By using a thread, the master remains free to accept incoming worker connections as before.

4.8 Minimizing Re-creation of Local Variables

There is only one single ByteBuffer for read and write. As the buffer size is large (4096 bytes), we don't want to allocate the buffer every time for read and write which will affect performance. So the ByteBufer is reused as much as possible.

The file list can be very large, so we let the worker to print out the crawl result instead of returning the large file list to the master. Each worker only needs to return the list of ultrapeers and leaves and agent and file numbers to the master so that the master can allocate work. The actual file list is not necessary for the master.

Reason for Decision:
This is to increase performance and minimizing memory usage. As the master can be a bottleneck, we want to minimize the work done at the master. The console output results at the worker is piped to a file and later copied from the PlanetLab nodes and concatenated together.

4.9 XML, XSL, CSS and reverse DNS

Our system supports XML formatted output (associated with a joint XSL/CSS style sheet) for effortless data analysis and viewing in either IE or Firefox as well as post-execution XML parsing, reverse DNS and statistics generation.

Reason for Decision:
The XML component is a non-trivial design choice as it enables quick file parsing (i.e. the IP addresses can be located immediately) and data analysis (i.e. the data can be loaded directly into Excel). Also, the fact that XML is loaded as a DOM tree enables concurrent modification of different parts of the data. This enabled us to write a multithreaded, semaphore-controlled algorithm to perform up to 50 reverse DNS operations at a given moment. Each DNS query may take up to a few seconds if the result is not cached locally, so executing hundreds of these sequentially would take a significant amount of time. For maximum clarity when viewing results, the output XML files are automatically associated to XSL and CSS style sheets and can be viewed in either IE or Firefox.

4.10 Graphical User Interface for Master Side

Although not required, our system also supports the use of a Graphical User Interface. This feature is a nice to have feature and can be enabled or disabled. We offer text fields to input the parameters of the crawler (initial nodes, port, IP address) as well as a field to enter the timeout of our crawler. We also display the number of workers connected to our system, and their IP address. Finally, we provide a log of all important things that occur in our system.

Reason for Decision:
The reason why we used a GUI on the master side is to make the system look more complete. It is also easier for us to debug our system by allowing us to easily look at what workers are connected, and whether we are sending work to it. In addition, we are able to alternate and adjust our workload according to the dynamic performance of our system. If we see that the workers are working faster than the work distribution of the supervisor, we can easily adjust and up the rate. With the help of the GUI, we can also start and stop the crawler at any time we please, and collect statistics for analyzing and data comparison.

4.11 Python Script for Starting Workers

Python script is used to initialize 50+ workers on PlanetLab nodes.

Reason for Decision:
Pythohn scripts are used for copying jar files to the PlanetLab nodes, starting the worker, copying back the output file from PlanetLab nodes and concatenating them together. This makes deploying the programs more efficiently.

5.0 Overall Statistics

5.1 Version with Txt Output (No XML and Reverse DNS)

This statistics is for a maximum workload of ***100*** nodes and ***50*** workers to crawl at a given instance for every worker for a total of 30 minutes. This is equivalent to maximum of ***100*** socket opening at a given instance of each worker. *This is the version without "xml output format and reverse DNS", but yields maximum performance/crawl speed.* The total output with individual nodes information and file lists is 59MB. As the file is too large, it is not included. Only a partial output from one node is included.

For raw file:
See Detailed_Statistics\TXT_Format\output.txt
See Detailed_Statistics\TXT_Format\out_partial.txt

Overall statistics:
Start time: 03:59:48
End time: 04:29:50
Total crawl time: 1801 seconds
Total active nodes crawled: 23863
Total nodes crawled: 61974
Total workers: 50
Nodes crawled per hour: 123879.0
Nodes crawled per worker per hour: 2477.58
Nodes crawled per socket per hour: 24.7758
Agent stats:
<AgentStats>
LimeWire(acqlite) : 1 - 0.00%
CitrixWire : 1 - 0.00%
LimeWire : 11078 - 47.32%
gtk-gnutella : 27 - 0.12%
LemonWire : 2 - 0.01%
Gnucleus : 2 - 0.01%
LimeWire Music : 2 - 0.01%
Phex : 1 - 0.00%
LimeWireTurbo : 4 - 0.02%
WinMX : 1 - 0.00%
Shareaza : 74 - 0.32%
360Share : 9 - 0.04%
A : 37 - 0.16%
MP3Torpedo : 3 - 0.01%
Gnutella : 1 - 0.00%
WinMX Music : 4 - 0.02%
FrostWire : 224 - 0.96%
BearShare : 11896 - 50.82%

360Share Pro : 14 - 0.06%
giFT-Gnutella : 23 - 0.10%
1 : 2 - 0.01%
eTomi Pro : 4 - 0.02%
</AgentStats>
Total reported files: 862747
Total actual files parsed: 608635

5.2 Version with Xml Output and Reverse DNS

For the version with xml output and reverse DNS, the file list is send back to the master by the worker. *This requires more memory for the master and hence the maximum work load is set to 20-30 for it to run 30 minutes (Not maximum performance).*

For formatted xml file:
See Detailed_Statistics\XML_format\output_processed.xml

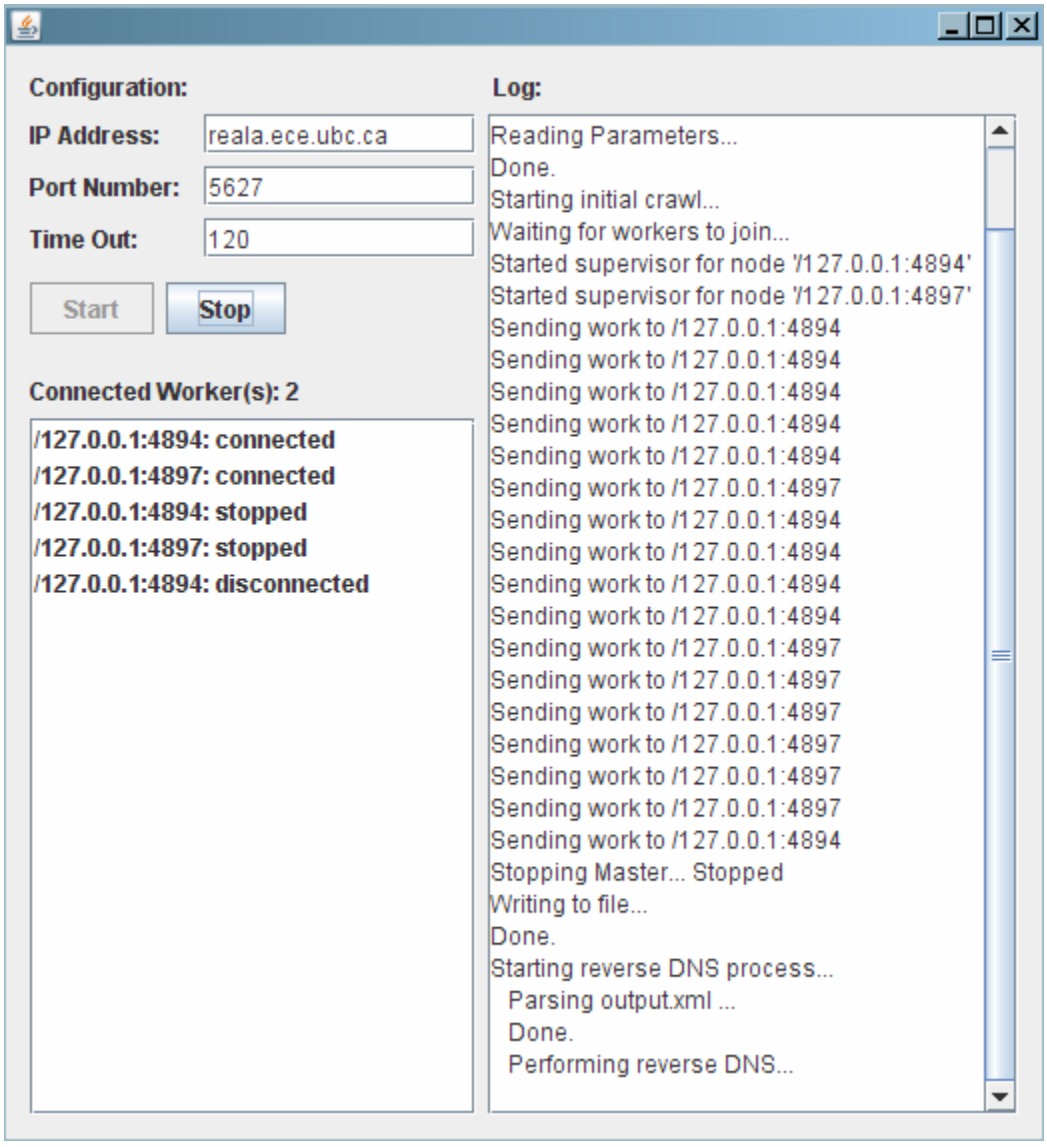
Total Crawl Time: 1800 seconds
Start Time: 21:29:38
End Time: 21:59:39
Total Number of Workers: 50
Total Active Nodes Crawled: 7544
Total Nodes Crawled: 20573
Average Nodes Crawled Per Hour: 41146.0
Average Nodes Crawled Per Worker Per Hour: 822.92
Average Nodes Crawled Per Socket Per Hour: 41.146
Agent Statistics:
LimeWire : 3412 - 46.05%
LimeWireTurbo : 1 - 0.01%
A : 27 - 0.36%
Gnucleus : 3 - 0.04%
360Share Pro : 6 - 0.08%
BearShare Turbo : 1 - 0.01%
Phex : 1 - 0.01%
Shareaza : 22 - 0.30%
FrostWire : 109 - 1.47%
360Share : 8 - 0.11%
WinMX Music : 3 - 0.04%
LimeWire Music : 2 - 0.03%
gtk-gnutella : 21 - 0.28%
BearShare : 3792 - 51.17%
1 : 2 - 0.03%
Domain Statistics:
net : 2697 - 41.65%
aw : 1 - 0.02%

biz : 2 - 0.03%
au : 82 - 1.27%
at : 14 - 0.22%
ar : 3 - 0.05%
my : 5 - 0.08%
mx : 8 - 0.12%
com : 2264 - 34.96%
gr : 5 - 0.08%
pt : 1 - 0.02%
jp : 34 - 0.53%
dk : 19 - 0.29%
yu : 3 - 0.05%
ae : 9 - 0.14%
de : 82 - 1.27%
pl : 518 - 8.00%
mg : 1 - 0.02%
sk : 2 - 0.03%
ph : 1 - 0.02%
si : 1 - 0.02%
sg : 6 - 0.09%
se : 18 - 0.28%
cz : 4 - 0.06%
ma : 4 - 0.06%
cy : 2 - 0.03%
ve : 1 - 0.02%
edu : 29 - 0.45%
cr : 1 - 0.02%
fr : 8 - 0.12%
it : 157 - 2.42%
co : 2 - 0.03%
lt : 3 - 0.05%
ru : 1 - 0.02%
ch : 14 - 0.22%
il : 8 - 0.12%
us : 1 - 0.02%
fi : 8 - 0.12%
ca : 203 - 3.13%
ie : 3 - 0.05%
uk : 77 - 1.19%
br : 12 - 0.19%
nz : 1 - 0.02%
es : 1 - 0.02%
hu : 1 - 0.02%
org : 1 - 0.02%
hr : 1 - 0.02%
bm : 1 - 0.02%

tv : 1 - 0.02%
no : 25 - 0.39%
be : 33 - 0.51%
nl : 93 - 1.44%
ee : 3 - 0.05%
ni : 1 - 0.02%
Total Reported Files: 273496

6.0 Graphical User Interface

Below is the GUI for the master that was used during debugging and performance adjustment.



7.0 How to Run

7.1 Version with Txt Output (No XML and Reverse DNS)

All files for this version are in the folder “TXT_version”.

7.1.1 Master

The executable for the master is GnutellaNIOCrawler\TXT_version\jar\master_g7.jar.

The command line to run the master without GUI is:

java –jar master_g7.jar <bootstrap node ip> <bootstrap node port number> <timeout in secondes> [master port number]

[master port number] is *option* for the user to specify the port number of master as a server for worker to connect. The default port number for master is 2000.

Example:
java –jar master_g7.jar reala.ece.ubc.ca 5627 1800

The command line to run the master with GUI is:

java –jar master_g7.jar –gui -off

-off is to turn off the xml output format.

The overall statistics is output to output.txt

7.1.2 Worker

The executable for the worker is GnutellaNIOCrawler\TXT_version\jar\worker_g7.jar.

The command line to run the worker is:

'~/jre1.6.0_03/bin/java -jar worker_g7.jar <master ip> <master port> > out.txt &'

7.1.3 Scripts

The list of PlanetLab nodes is nodes3.txt.

The python script to copy jar file to the PlanetLab nodes is scpCommand.py. It reads nodes3.txt
To run it, type: python scpCommand.py

The python script to start the worker is sshCommand.py. It reads nodes3.txt and sshCommandList.txt which contains the commands to be executed through ssh (In this case it should contain '~/jre1.6.0_03/bin/java -jar worker_g7.jar <master ip> <master port> > out.txt &').
To run it, type: python sshCommand.py

The python script that copies back the output file is fininish.py. It reads nodes3.txt. As the output file is large, it copies the file to a temporary folder on the ece linux machine. In this script, the output file is copied to /tmp/yy/out.txt and concatenated to /tmp/yy/result.txt. So need to create a directory "yy" in the /tmp directory first.
.

7.2 Version with Xml Output and Reverse DNS

All files for this version is in the folder "XML_version".

7.2.1 Master

The executable for the master is GnutellaNIOCrawler\XML_version\master_g7.jar.

The command line to run the master without GUI is:

java –jar master_g7.jar <bootstrap node ip> <bootstrap node port number> <timeout in secondes> -xml

Example:
java –jar master_g7.jar reala.ece.ubc.ca 5627 1800 -xml

The command line to run the master with GUI is:

java –jar master_g7.jar –gui

The overall output before reverse dns is output to output.xml. The output after reverse dns is to output_processed.xml. The overall_stats.txt is for a quick preview of the overall statistics and is not final results.

7.2.2 Worker

The executable for the worker is GnutellaNIOCrawler\XML_version\worker_g7.jar.

The command line to run the worker is:

'~/jre1.6.0_03/bin/java -jar worker_g7.jar <master ip> 2000 > out.txt &'

2000 is the default master port which cannot be changed in this version.

7.2.3 Scripts

The list of PlanetLab nodes is nodes3.txt.

The python script to copy jar file to the PlanetLab nodes is scpCommand.py. It reads nodes3.txt
To run it, type: python scpCommand.py

The python script to start the worker is sshCommand.py. It reads nodes3.txt and sshCommandList.txt which contains the commands to be executed through ssh (In this case it should contain '~/jre1.6.0_03/bin/java -jar worker_g7.jar <master ip> 2000 > out.txt &').
To run it, type: python sshCommand.py

For Xml_version, all the output is at the master, so there is no need to copy back the output at the worker.