2009

Brute Force Grid

Detail Report – CTP 2009

Brute Force Grid is a system developed to demonstrate the efficiency of the Peer to Peer parallel processing, using the application demonstration of brute force key recovery from hashing algorithms.

This is an extended version of the thesis which is included in the final report. The detail report includes more technical aspects and views rather than of consumer's perspective.

M.F.F.Faraj
SLIIT | Curtin University Of Technology
5/11/2009

Contents

1. Project Components

1.1 Peer application

Peer application is the software piece which runs on the host that makes that specific host, the peer to the overlay Peer to Peer network of the brute force Grid. Peer application is a set of modules working together to produce the protocol a working object. These modules are further divided into three categories.

- Data Collections
- Working Components
- GUI

The peer application is developed using the modular concept used in most of the Operating Systems. This eases the process of debugging and integrity check. So when the application software starts, the initialization function of each module is called by the startup module. And when the application quits, all the modules where turned off using the shut down functions available.

Each module is like a separate demon which is executed using the timers of java. So each demons execution interval can be easily managed.

So in the above mentioned working components in the peer application, there may exist several modules which runs separately. And the Data Collections are the shared memory interface which is used by the thread demons.

1.1.1 Data Collections

Data collection is the location to store and manipulate the variables and other data structured need to execute the Peer Application. The data collection Component has several modules which is used to store different types of data models needed for the execution of the application.

- **Available Decryptor List**

  This module consists of the plug-in list that is stored in the plug-in folder of that specific host. The check is done periodically after a specific time interval. This enables the detection of any new plug-ins installed on run time of the application. The function is available in this Data module which should be called in order to find new decryptor on run-time.

  ```
  ArrayList<Decryptor> decryptors;
  ```

  The data about the decryptors were stored in the Decryptor data type.

  ```
  int decryptorID;
  String description;
  ```

  This ID is used to search the decryptor in the execution time. The description is used to name the decryptor when storing in the hard disk.

- **Common Flags**

  Consists of the flags and other variables that is used to signal the inter thread or inter service modules on various occasions. This is not used in many places. But to keep the modular concept I created this data module to provide the flagging between the demons.

  ```
  boolean IP_LIST_EMPTY = true;
  boolean GRID_PEER_EMPTY = true;
  boolean GRID_PEER_FULL = false;
  ```

- **Common Register**

  Common Register is the main sub module which contains all the configuration variables of the application. This includes the entries found in the settings GUI interface which is shown before. The variables are stored in the configuration file when the application closes, and restore from that file when the application restarts. This is also used to handle the configuration file.

  ```
  String PEER_NAME = "MINI_HOST";
  ```

  - Used to give the simple name for the peer

  ```
  int MAX_PARELLEL_CONNECTIONS = 5;
  ```

  - The maximum number of parallel connections can be tried out at once

String LOG_FILE = "log.txt";

- The file path of the logging file in the hard disk

int DEFAULT_PORT = 800;

- The default port to which the connection is made to

int CONNCETIVITY_REPLIER_PORT(){return DEFAULT_PORT + 1;}

- Port used to monitor the connectivity

int FILE_TRASFER_PORT(){return DEFAULT_PORT + 2;}

- Port used to transfer the plug-in or decryptor

int MAX_GRID_PEERS = 10;

- Maximum amount of peers connectable

int MAX_REMOTE_PEERS_PER_JOB = 3;

- Maximum amount of peers that can be assign to a job

int MAX_REMOTE_SERVERS = 10;

- Maximum amount of peers to which the local sub jobs can be sent at a particular time.

int MAX_REMOTE_CLIENTS = 10;

- Maximum amount of peers from which the remote jobs can be received at a particular time.

int MAX_ERROR_COUNT_IP_LIST = 3;

- Maximum number of errors tolerates until the particular peer to be rejected from being tried out.

String REMOTE_IP_LIST_FILE_PATH = "ril.lst";

- The temporary file created in the hard disk in which the IP addresses of the last connected peers were stored in for the next time check up.

String DECRYPTOR_LOCAL_ROOT_PATH = "decryptors/";

- The path where the decryptor is stored in the application

int REMOTE_PEER_CONNECTOR_START_DELAY = 1000;

- The delay to execute the module Remote Peer Connector after execution of the initialization.

int REMOTE_CONNECTIVITY_CHECK_DELAY = 1000;

- The delay to execute the module Remote Connectivity Check after execution of the initialization.

int COMMON_INTERFACE_START_DELAY = 1000;

- The delay to execute the module Common Interface after execution of the initialization.

```
int REMOTE_PEER_CONNECTOR_TRYOUT_PERIOD = 5000;
```

- The time interval in which the remote peer connector is tried out for a particular ip address

```
String WEB_SERVER_DOMAIN = "http://localhost/BFGServer";
```

- Web server URL

```
String SERVER_IP_FILE =  "activeip.php";
```

- File used to retrieve the peers up and ready to be connected

```
String PROBE_FILE =  "poke.php";
```

- The file which is use to probe the server to tell the peer existence

```
String PLUGIN_CHKSUM = "checksum.php";
```

- The server file used to check the hash of the decryptor which is certified

```
String SERVER_IP_FILE_PATH(){return WEB_SERVER_DOMAIN + "/" +
SERVER_IP_FILE;}
String SERVER_IP_PROBE_URL(){return WEB_SERVER_DOMAIN + "/" +
PROBE_FILE;}
String SERVER_IP_PLUGIN_CHKSUM(){return WEB_SERVER_DOMAIN + "/" +
PLUGIN_CHKSUM;}
```

- Functions use to get the server URLs with the specific files.

```
int LOCAL_SUB_JOB_SIZE = 100000;
```

- Number of passwords assigned to a sub job

```
int TIME_TO_AWAIT_UNTIL_DOWNLOAD = 3; //sec
```

- The awaiting time used to timeout the download

```
int SIZE_OF_FILE_READBLE_AT_A_TIME = 1024;
```

- Size of the file readable at a particular time

```
final int LINUX = 0;
final int WINDOWS = 1;
final int MAC = 2;
```

- Numerical representation of the Operating System

```
int SHEDULER_POLL_TIME = 5000;
```

- Scheduler polling time interval

```
int PARELLEL_JOBS_PER_PEER=2;
```

- Parallel jobs sent from a particular remote peer at a time.

```
String CONFIG_FILE = "bfgsettings.conf";
```

- Configuration file path of the BFG peer application

- **Grid Peer List**

  This contains the Peers information which is connected or connectable. Remote Peer Trier and the common Interface fills up this list. And when a peer goes down, the Grid Peer List module is updated.

  ```
  ArrayList<GridPeer> gridList;
  ```

  Grid Peer which is the class used to store the connected and connectable peers consist of,

  ```
  Socket connectionSocket;
  ```
  - The socket of the connection

  ```
  InetAddress address;
  ```
  - Store IP address

  ```
  BufferedReader reader;
  PrintWriter writer;
  ```
  - Reader and the writer objects use to transfer data to the peer

  ```
  int localPeerID;
  ```
  - The peer ID number

  ```
  Timer listenerTimer;
  ```
  - The timer demon used for listening remote requests.

  ```
  boolean connected;
  boolean blocked;
  ```
  - Flags used to store the peer states.

- **Peer Info**

  This sub module is used to produce the system information to send to other peers, and to use for the system task such as platform detection. This component can be further developed according to the need and new features.

- **Local Job**

  This is the module used to handle the local job. When a new local job is inserted to this module, it automatically fragment the job into sub jobs using the information provided in the common register. This module also contains a state for each and every sub job created and for the whole job as one. These sates are useful when determining the job statues when the program is running.

  ```
  LocalSubJob[] localSubJobs = new LocalSubJob[0];
  ```
  - The set of sub jobs that is created whena a job is initiated in the local peer.

  ```
  int size;
  ```
  - Size of the sub job that is going to be created

String decyptorDiscr;

- Decryptor descriptor of the plug in which hashing algorithm used to encrypt the key to the hash.

Timer timer;

- Timer thread used to connect to the peers and distribute the sub jobs.

boolean started,paused,success;

- Statue flags of the local job module.

String password;

- The key retrieved is stored in this variable

int localCount = 0;

- The local peer count.

The Local sub job is the object class which is used to store each and every sub jobs created. Local Sub Job and Remote Sub Job were derived from the Sub Job class. There are no new features added to the child classes from the parent. So let us focus on the Parent class that is Sub Job class.

```
public static final int NEW = -2; // when the new job is created
public static final int PENDING = -1; // when it is waiting to be set to a GP
public static final int IDLE = 0; // in GP but not started yet
public static final int STARTED = 2; // GP started
public static final int STOPED = 3; // GP stoped n kill job
public static final int FINISHED = 4; // GP finished execution
public static final int SUCCESS = 5; // GP found the password
public static final int PAUSED = 6; // GP paused

String decryptorDiscr; // the plugin information to search or download it.
int jobID; //local sub job identification
int gridPeerID; // the peer id to which the sub job is assigned to
int size; //maximum size of the password
int segmentSize; //size of the sub job
int segmentNum; //segment or sub job number
boolean checkSimple,checkCapital,checkSpecial,checkNumber; //criteria of check
int state; //state of the sub job
String hash; //hash to decrypt
int timeWait = 0; // used to make a timeout.
```

- **Remote Job List**

  This data structure module consists of the sub jobs received by the remote peers. When a remote peer is down, the sub jobs received by that specific peer is removed from this list. The list is decremented by the scheduler, when a sub job executed successfully.

  ```
  ArrayList<RemoteSubJob> remoteJobs;
  ```

- **Net Data**

  This data structure module is used to send the information from one peer to another. This module defines the way of signal and information transmission from one peer to another. We can create an object of this module; feed it with the information to transmit, and transmit to the peer. From where the peer can reconstruct the object and retrieve the information received by the peer.

  ```
  public static final String DELEMETER = " "; // delemeter
  public static final String TRUE = "TRUE"; // true
  public static final String FALSE = "FALSE"; // false
  public static final int REQUEST_PEER_LIST = 0; //0
  public static final int RESPONSE_PEER_LIST = 1; //1 <ip> <ip>
  public static final int REQUEST_TO_STOP = 2; //2 <jobid>
  public static final int RESPONSE_TO_STOP = 3; //3 TRUE/FASE
  public static final int REQUEST_REMOTE_PEER_INFO = 4;//4
  public static final int RESPONSE_REMOTE_PEER_INFO = 5;//5 <peername>
  public static final int REQUEST_START_JOB = 6;//6 <localSubJob>
  public static final int RESPONSE_START_JOB = 7;//7 TRUE/FASE
  public static final int REQUEST_TO_PAUSE = 8; //8 <jobid>
  public static final int RESPONSE_TO_PAUSE = 9; //9 TRUE/FASE
  public static final int REQUEST_TO_RESUME = 10; //10 <jobid>
  public static final int RESPONSE_TO_RESUME = 11; //11 TRUE/FASE
  public static final int REQUEST_TO_UPDATE_JOB_STATUS = 12; //12<remoteSubJob>
  public static final int RESPONSE_TO_UPDATE_JOB_STATUS = 13; //13 TRUE/FASE
  public static final int REQUEST_TO_STOP_ON_SUCCESS = 14; // 14 <output>
  public static final int RESPONSE_TO_STOP_ON_SUCCESS = 15; // 15 TRUE/FALSE
  ```

  - Types of signals that can be sent using the Net Data format

  ```
  int type;
  ```

  - The type ID of the specific Net Data signal

  ```
  String[] parameters;
  ```

  - Parameters of the signal

  ```
  String rawData;
  ```

  - The actual message string send or received

- **Error and Messenger**

  These two sub modules are used to notify the use about the events happening in the application. These are the modules used to update the console preview of the GUI.

  The list of xError messages defined in this module

```
CREATING_IP_LIST_FILE = "error creating IP LIST FILE";
READING_IP_LIST_FILE = "cannot read IP LIST FILE";
WRITING_IP_LIST_FILE = "cannot write IP LIST FILE";
OPENING_IP_LIST_FILE = "cannot open IP LIST FILE";
CREATING_SOCKET_GRID_PEER = "error creating the socket in grid peer";
REMOVING_SOCKET_GRID_PEER = "error removing peer in grid peer";
REMOVING_IP_LIST_FILE = "error removing IP list file";
CLOSING_IP_LIST_FILE = "error closing IP list file";
CONNECTING_TO_SERVER_IP_FILE = "error connecting to server's IP file";
READING_SERVER_IP_LIST_FILE = "cannot read IP LIST FILE in server";
CONNECTING_SERVER_TO_PROBE = "cannot probe to server";
CLOSING_REMOTE_IP_FILE = "cannot close IP file";
ERROR_STARTING_SERVER = "error starting server on port "+ DEFAULT_PORT;
CLIENT_CONNECTION_FAILED = "remote client connection failed";
CLOSE_SERVER_SOCKET = "couldn't close local service";
WRITE_TO_SOCKET = "unable to write data to remote peer";
READ_FROM_SOCKET = "unable to read data from remote peer";
REMOTE_STOP_REQUEST_CANCELED = "remote stop request canceled";
UNKNOWN_REMOTE_REQUEST = "remote request unknown";
PEER_CONECTION_LOST = "peer connection lost";
UNKNOWN_REMOTE_NETDATA = "unknown remote net data";
START_CONNECTIVITY_REPLIER = "error creating connectivity replier";
STOP_CONNECTIVITY_REPLIER = "error stopping connectivity replier";
CONNECTING_DECRYPTOR_DOWNLOAD_POINT = "error connecting to decryptor
download point";
CREATING_DECRYPTOR_LOCAL_FILE = "error creating decryptor local file";
CLOSING_DECRYPTOR_FILES_DOWNLOADED = "error closing decryptor file
downloading";
FETCHING_DECRYPTOR_INFO = "error getting the decryptor information";
FILE_TRANSFER_SERVER = "file transfer server error";
FETCHING_LOCAL_DECRYPTORS = "error fetching local decryptors";
```

CREATING_LOCAL_SUB_JOBS = "local sub jobs creation error";
SHEDULER_EXECUTION = "Scheduler execution error";

The list of xMessages messages defined in this module

PEER_LIST_FILLED = "Peer list Filled";
REMOTE_SERVICE_STARTED = "remote service started";
REMOTE_PEER_CONNECTED = "remote peer connected";
SERVER_PROBING = "server probing now...";
REMOTE_NETDATA_RECIEVED = "remote request received";
PEER_LIST_SENDING = "sending peer list to remote location";
NEW_IP_LIST_RECIEVED = "new IP list received and updated!";
TRYING_OUT_NEW_PEERS = "trying out new peers...";
WAITING_FOR_FILE_TRASFER = "waiting for file transfer"
FILE_TRASFER_STARTED = "transfer started";
FILE_TRASFER_COMPLETED = "transfer completed";
SENDING_FILE_REQUEST = "sending file server the request to get the file";
WAITING_FORFILE_RESPONSE = "waiting for file server to response";
ACK_FROM_FILE_SERVER = "file found response received by server";
NACK_FROM_FILE_SERVER = "file not found response by server";
SERVER_FILE_TRANSFER_FINISHED = "server finished transmitting the file";
SERVER_FILE_TRANSFER_STARTED = "server started transmitting the file";
WATING_FOR_CONFIRMATION = "server waiting for file confirmation";
FILE_NOT_FOUND_AND_NACK = "requested file not found";
FILE_FOUND_AND_ACK = "requested file found";
FILE_REQUEST_RECIEVED = "client request received";

1.1.2 Working components

This consists of the sub modules or services that executes while the program is running. These Modules use the data components to store and retrieve information.

- **Benchmark**
  This module is used to determine the statues of the peer. That is to accept the request from the remote peer or not. This can be based on many factors. We haven't implemented this part in this prototype. But we created free interface so that anyone can used to develop it later.
  public static boolean isInNewJobAcceptanceCondition()
    - Function to check before accepting a new job

- **Common Interface**

  This is the listening server components which creates the remote connections and add to the grid peer list. This is the open interface to the other peers to establish a connection. This is a single demon and no external invokes is available.

- **Connectivity Replier**

  When a peer is connected or set as connectable, the other peers send this peer periodic probes to check whether the peer is still in the connected or connectable state. This connectivity replier is the module which response to these probes.

- **Decryptor Downloader**

  Decryptor downloader is used to download the plug-in from the job initiator peer to the local peer as if the plug-in is not available. This Module is responsible to check online on the web server about the plug-ins certification before continuing.

  ```
  public static boolean Get(String DecryptorDiscr, GridPeer gridPeer)
  ```

  - The interface provider for the external modules to invoke the download by giving the decryptor descriptor and grid peer to download from.

- **File Transfer Server**

  FT server is use to response to the Decryptor downloader. This module is used to transmit the plug-in file to other peers which has requested to download the file. FT server uses a separate port other than the default port. The demon which runs separately and responses to the Decryptor Downloader

- **Initializer**

  This module is responsible to initialize each and every component available in the Brute Force Grid application. Calls the Initialization function of each module, and print the responses.

  ```
  message("Local Job",LocalJob.initialize());
  message("Remote IP List",RemoteIPList.initalize());
  message("Remote Job List",RemoteJobList.initialize());
  message("Available Decrptor List",AvailableDecryptorList.Initalize());
  message("Gridpeer List",GridPeerList.initialize());
  message("CommonRegister",CommonRegister.initialize());

  message("Logger",Logger.initialize());
  message("Connectivity Monitor",ConnectivityReplier.initialize());
  message("RIL Filler",RILFiller.initialize());
  ```

```
message("Server Prober",ServerProber.initialize());
message("Common Interface",CommonInterface.initialize());
message("Remote Connectivity Monitor",RemoteConnectivityMonitor.initialize());
message("Remote Peer Connector",RemotePeerTryer.initialize());
message("Local Job Distributor",LocalJobDistributor.initialize());
message("Benchmark",BenchMark.initialize());
message("File Transfer Server",FileTransferServer.initialize());
message("Sheduler",Sheduler.initialize());
message("Disconnector",PeerDisconnector.initialize());
message("LocalJobMonitor",LocalJobMonitor.initialize());
```

- **List Updater**

  List Updater is used to update the list available in the application according to the new peers connecting and peers disconnecting. This is the module responsible to remove the IP list and sub jobs as if the grid peer has got down.

- **Local Job Distributer**

  Local Job Distributer is responsible to distribute the local sub jobs to the specific peer connected. This module gets activated when a net local job is put to the local job component. And when the size of the local sub jobs list is zero, this module deactivates.

  ```
  public static void startNew(String decryptorDescr, String hash,int size, boolean isSimple,
  boolean isCapital, boolean isSpecial, boolean isNumber)
  ```

  - Used to start a new job and a separate demon is start on run and make the distributions on background.

- **Local Job Monitor**

  This Module is responsible for monitor the local jobs. If a particular local job doesn't get response from the remote peer or gets the failed response, this module is responsible to reset the sub job to the original state where the sub job can be assigned to another peer. Separate demon, doesn't interact or invoked by others.

- **Logger**

  Logger is the module which logs the events and errors in the log file. This can be used to determine the progress when an error occurs. Or to trace down the process for any administrative needs.

  ```
  public static void log(String val)
  ```

  - to add a log entry. Can be called from any place.

- **Net Data handler**

  When a data is received by the remote peer to the local peer, this is the module that reconstructs the Net Data object and retrieves the information sent to the local peer. And this module is responsible to initiate any activities related to the request received.

  public static void HandleResposeRemotePeerInfo(NetData data, GridPeer gridPeer)
  public static void HandleResposePeerList(NetData data, GridPeer gridPeer)
  public static void HandleResponseStartJob(NetData data, GridPeer gridPeer)
  public static void HandleRequestRemotePeerInfo(NetData data, GridPeer gridPeer)
  public static void HandleRequestStartJob(NetData data, GridPeer gridPeer)
  public static void HandleRequestToStop(NetData data, GridPeer gridPeer)
  public static void HandleRequestPeerList(NetData data, GridPeer gridPeer)
  public static void HandleResponseToStop(NetData data, GridPeer gridPeer)
  public static boolean HandleRequestToPause(NetData data, GridPeer gridPeer)
  public static void HandleResponseToPause(NetData data, GridPeer gridPeer)
  public static boolean HandleRequestToResume(NetData data, GridPeer gridPeer)
  public static void HandleResponseToResume(NetData data, GridPeer gridPeer)
  public static void HandleRequestToUpdateJobStatus(NetData data, GridPeer gridPeer)
  public static void HandleResponseToUpdateJobStatus(NetData data, GridPeer gridPeer)
  public static void HandleRequestToStopOnSuccess(NetData data, GridPeer gridPeer)
  public static void HandleResponseToStopOnSuccess(NetData data, GridPeer gridPeer)

- **Peer Disconnect**

  As said before, the connection is existed only if there are any transactions between the two peers. If not the Peer should be disconnected and kept in the connectable state. This should be done after checking the sub job lists as if there is no sub jobs exists before the peer is disconnected. This is done using the Peer Disconnect module.

- **Plug-in Interface**

  This module is responsible for the invoking of the C++ plug-in using the JAVA program. That is the peer application. This module is the platform used by the peer application as well as the plug-in to interact with each other.

  public static void RUN_PLUGIN(String path, String hash, int size, int segmentSize, int segmentNum, boolean isCapital, boolean isSimple, boolean isNumaric, boolean isSpecial)

- **RIL filler**

  RIL filler is responsible to fetch the IP addresses connectable from the web server and store it in the Remote IP list data structure module as well as in the backup file.

```
url = new URL(CommonRegister.SERVER_IP_FILE_PATH());
reader = new BufferedReader(new InputStreamReader(url.openStream()));
```

- **Remote Connectivity Monitor**

  This module is used to check the connectivity between the peers. When a peer gets itself inside the grid peer list, this module automatically send probes to that peer and waits for the reply. If the reply did not received, the peer is considers to be down, and the List updater is called. Separate demon which runs on its own.

- **Remote Listener**

  This sub module waits for the connected peers to send the signals and request to the local peer. And this module is responsible to call the net data handler when the request or some other data received.

```
case NetData.REQUEST_TO_STOP :
   NetDataHandler.HandleRequestToStop(netData, gridPeer);
   break;
case NetData.REQUEST_PEER_LIST :
   NetDataHandler.HandleRequestPeerList(netData, gridPeer);
   break;
case NetData.REQUEST_START_JOB :
   NetDataHandler.HandleRequestStartJob(netData, gridPeer);
   break;
case NetData.REQUEST_REMOTE_PEER_INFO:
   NetDataHandler.HandleRequestRemotePeerInfo(netData, gridPeer);
   break;
case NetData.RESPONSE_PEER_LIST :
   NetDataHandler.HandleResposePeerList(netData,gridPeer);
   break;
case NetData.RESPONSE_REMOTE_PEER_INFO :
   NetDataHandler.HandleResposeRemotePeerInfo(netData, gridPeer);
   break;
case NetData.RESPONSE_START_JOB:
   NetDataHandler.HandleResponseStartJob(netData, gridPeer);
   break;
```

```
case NetData.RESPONSE_TO_STOP:
    NetDataHandler.HandleResponseToStop(netData, gridPeer);
    break;
case NetData.RESPONSE_TO_PAUSE:
    NetDataHandler.HandleResponseToPause(netData, gridPeer);
    break;
case NetData.RESPONSE_TO_RESUME:
    NetDataHandler.HandleResponseToResume(netData, gridPeer);
    break;
case NetData.REQUEST_TO_PAUSE:
    NetDataHandler.HandleRequestToPause(netData, gridPeer);
    break;
case NetData.REQUEST_TO_RESUME:
    NetDataHandler.HandleRequestToResume(netData, gridPeer);
    break;
case NetData.REQUEST_TO_UPDATE_JOB_STATUS:
    NetDataHandler.HandleRequestToUpdateJobStatus(netData, gridPeer);
    break;
case NetData.RESPONSE_TO_UPDATE_JOB_STATUS:
    NetDataHandler.HandleResponseToUpdateJobStatus(netData, gridPeer);
    break;
case NetData.REQUEST_TO_STOP_ON_SUCCESS:
    NetDataHandler.HandleRequestToStopOnSuccess(netData, gridPeer);
    break;
case NetData.RESPONSE_TO_STOP_ON_SUCCESS:
    NetDataHandler.HandleResponseToStopOnSuccess(netData, gridPeer);
    break;
default:
    xError.Report(xError.UNKNOWN_REMOTE_NETDATA);
    break;
```

- **Remote Peer Trier**

  Remote Peer Trier try-outs the IP address in the Remote IP List sub module and insert the entries to the Grid Peer List module. When trying out peers, the simultaneous try-out was carried out to increase the performance.

- **Remote Requestor**

  This module is used to send signals or request to the remote peer. This module is responsible to show off the whole content of the request send.

  ```
  public static void RequestForIPList(GridPeer gridPeer)
  public static void RequestForPeerInfo(GridPeer gridPeer)
  public static void RequestToStopJob(GridPeer gridPeer)
  public static void RequestToStartJob(GridPeer gridPeer, LocalSubJob localSubJob)
  public static boolean RequestToPauseJob(LocalSubJob localSubJob)
  public static boolean RequestToResumeJob(LocalSubJob localSubJob)
  public static boolean RequestToUpdateJobStatus(RemoteSubJob job)
  public static void RequestToStopOnSuccess(RemoteSubJob job, String output)
  ```

- **Remote Response**

  when a request is received from the remote peer, this is the module used to response to those requests. The response and the request definitions are set I the net data structure, this is used to determine the exact response received.

  ```
  public static void ResponsePeerList(GridPeer sendGridPeer)
  public static void ResponseToStop(GridPeer gridPeer, boolean isStoped)
  public static void ResponseStartJob(GridPeer gridPeer, boolean isAccepting)
  public static void ResponseRemotePeerInfo(GridPeer sendGridPeer, String info)
  public static void ResponseToPauseJob(GridPeer gridPeer, boolean isPaused)
  public static void ResponseToResumeJob(GridPeer gridPeer, boolean isResumed)
  public static void ResponseToUpdateJobStatus(GridPeer gridPeer, boolean isUpdated)
  ```

- **Server Prober**

  The local Peer should probe to the web server periodically to inform that the peer is up and ready to be connected by other peers. This work is done by this module. This module connects to the web server and informs the existence.

  ```
  URL url = new URL(CommonRegister.SERVER_IP_PROBE_URL());
  ```

- **Scheduler**

  Scheduler schedules the remote sub jobs one by one to the plug-in interface to execute for the result. When a sub job executed successfully it is removed from the list of the remote sub jobs. If there are no sub jobs to execute, the scheduler is put to sleep until the remote sub job is received.

- **Shut downer**

  Shut downer is used to shut down the services and modules initialized by the initialize process. This frees the resources and shut down all the working components and the data collections.

```
message("Local Job",LocalJob.shutdown());
message("Remote IP List",RemoteIPList.shutdown());
message("Remote Job List",RemoteJobList.shutdown());
message("Available Decrptor List",AvailableDecryptorList.shutdown());
message("Gridpeer List",GridPeerList.shutdown());
message("Common Register",CommonRegister.shutdown());
message("Connectivity Monitor",ConnectivityReplier.shutdown());
message("RIL Filler",RILFiller.shutdown());
message("Server Prober",ServerProber.shutdown());
message("Common Interface",CommonInterface.shutdown());
message("Remote Connectivity Monitor",RemoteConnectivityMonitor.shutdown());
message("Remote Peer Connector",RemotePeerTryer.shutdown());
message("Local Job Distributor",LocalJobDistributor.shutdown());
message("Benchmark",BenchMark.shutdown());
message("File Transfer Server",FileTransferServer.shutdown());
message("Sheduler",Sheduler.shutdown());
message("Disconnector", PeerDisconnector.shutdown());
message("Logger", Logger.shutdown());
message("LocalJobMonitor", LocalJobMonitor.shutdown());
message("Config store", CommonRegister.shutdown());
```

1.1.3 GUI

GUI is the simplest but much time consumed section of the project. GUI consist a separate thread which updates the data and information available in the core framework and intermediate part to the form components.

```
Shower.UpdateGridListView();
Shower.UpdateLocalJobView();
Shower.UpdateRemoteJobList();
Shower.updateAlgoList();
Shower.UpdateGeneralInfo();
Shower.UpdateButtonValueBox();
Shower.updateRIList();
```

The logging view which is known as the console view in the GUI is the real-time update of the works done in the system. Continues operation of the application results in memory dumps as the console view text box gets too large value. So we limit the view to the last 20 lines of logging to view at a particular time. And the rest is permanently stored in the log file in the hard disk.

To give a cool look to the GUI interface created using JAVA, we used the free and open source look and feels.

```
jtattoo.plaf.acryl
```

1.2 Application Plug-in (MD5)

Since brute forcing takes a lot of computations, we decided to have it as the application which is run on top of the basic framework. So the user's task would be to derive the clear text password from a given encrypted hash key.

We can sub-divide the functionality of the plug-in software as follows

- Taking all the user inputs and store in the program.
- Create a data array depending on user inputs.
- Generate all the possible text combinations depending on the map created.
- Generate the hash key by sending it through the hash function.
- Hash comparison and give the final result back to the local host.

1.2.1 Storing the user inputs passed by the GUI to the program.

The user inputs are,

- The four conditional Booleans values,
  - **Symbolic** - consider symbolic values when generating the combinations.
  - **Upper** - consider uppercase characters when generating the combinations.
  - **Lower** - consider lowercase characters when generating the combinations.
  - **Numeric** - consider numbers when generating the combinations.

- Other user inputs,
  - **passLen** (integer) – The length of the password.
  - **segSize** (integer) – Size of a segment allocated for each node/host. (Number of combinations a node should process).
  - **segNo** (integer) – The segment number assigned for that particular host.
  - **PassHash** (String) - The hash key code of the real password.

1.2.2 Create a data array depending on user inputs.

When developing the brute force plug-in, one of the challenges we had to face is handling the characters using the ASCII table. When generating that combination, our plan was to use the ASCII values of the characters to do the text operations; like incrementing the ASCII value of a character by one to get the

next one. But we came across a problem when tackling special characters since there are in four places in the standard ASCII table.

We used loops and jumping methods to the catch the all special characters but this made the code was very complex and inefficient.

So we had to seek for a different approach when dealing with the characters. There we came to a solution of creating our own character map, a char array with indexed values. This enhancement worked fine and it helped us to reduce the code size and increased the efficiency of the program. This Map is created dynamically depending on the four conditional Boolean variables. Depending on what conditions are true, it creates the character map and stores the values. This way we only had to work with the index values of the characters not the ASCII value of them thus by allowing us to work without any dependency of the ASCII character map.

For example ***Symbolic*** – false, ***upper*** – true, ***lower*** – false, ***Numeric*** – true

Dynamically created Map array

1.2.3 Generate all the possible text combinations depending on the map created.

- We created a special function which returns the string component of the given element.

"String function Jump (double Val, into array Size, char arrayCharactor [], into passive)"

The input parameters are,

- **Double Val** – The position of the element to be retrieved, nth element of all the possible combinations. (egg: 100th).
- **int arraySize** – size of the character array.
- **char arrayCharactor[]** – The character array passed.
- **Int passsize** – The size of the password (for validations purposes).

- Above function is looped and called to get the string combinations within that segment range assigned for that local host.

1.2.4 Generate the hash key by sending it through the hash function.

In this phase, the string combinations retrieved in step 3 are sent though the hash function in order to get the hash key code.

MD5 plug-in consists of four c ++ code files apart from the main function. Those are,

- md5.cpp - C++ implementation of the MD5 Message-Digest
- md5.h – the header file
- md5wrapper.cpp - wrapper-class to create a MD5 Hash from a string.
- md5wrapper.h – the header file.

The hashing function,

*“md5wrapper md5;”“hash = md5**.getHashFromString (<<combination>>)**;”*

1.2.5 Hash comparison.

Then the generated hash key is compared with the original hash code. If it retrieves the correct password it sends the notification to the local host, or else it computes all the combination in the given range and informs the local host.

1.3 Web server

In this system the server is a very simple model because we are intended to make the project system as decentralized as possible. The peer IP addresses are stored in the My SQL database when a peer is probed. And when a peer requests the IP lists, it checks the database and produces the list of IP addresses.

We have put a mechanism to timeout and delete the IP entries if a specific host didn't probe for a particular amount of time. This is checked each time a peer request for IP list.

The web server consists of following modules:

- DB Connection

  This module deals with the database connection, SQL query execution and database closing. That is, this module provides the manipulation mechanism to the web server.

- Poke

  This is the page called by the application host in order to inform that the peer is active and up. So that information can be return to other peers, if the request is made by them.

- Active IPs

  This searches the DB and retrieves the up IP lists. Then the time is compared to get at what time the entry is made of. If the time period exceeds the IP entry is removed from the database.

- checksum

  This module is used to check the hash of the plug-in downloaded. When a request is made, this module produces the specific hash related to it and returns.

2 How the components work together

2.1 Peer Application and Web Server issues

Peer application is installed in the host and web server is installed in the public network within any peer or a separate host. The application plug-in is installed inside the peer application so that it is invoked using the peer application and it's under the control of it.

The Peer applications send the web server periodic probes to say that the specific peer is up and ready. So when the new peer is started. That is to run the peer application in one of the network host in the public network (or we can do this in the LAN environment too) the peer first send its existence to the web server. And this newly upped peer check its local IP list. This is the List which consists of the IP addresses of the remote peers that the local peer has previously connected to. These modules were discussed more specifically in the rest of the paper.

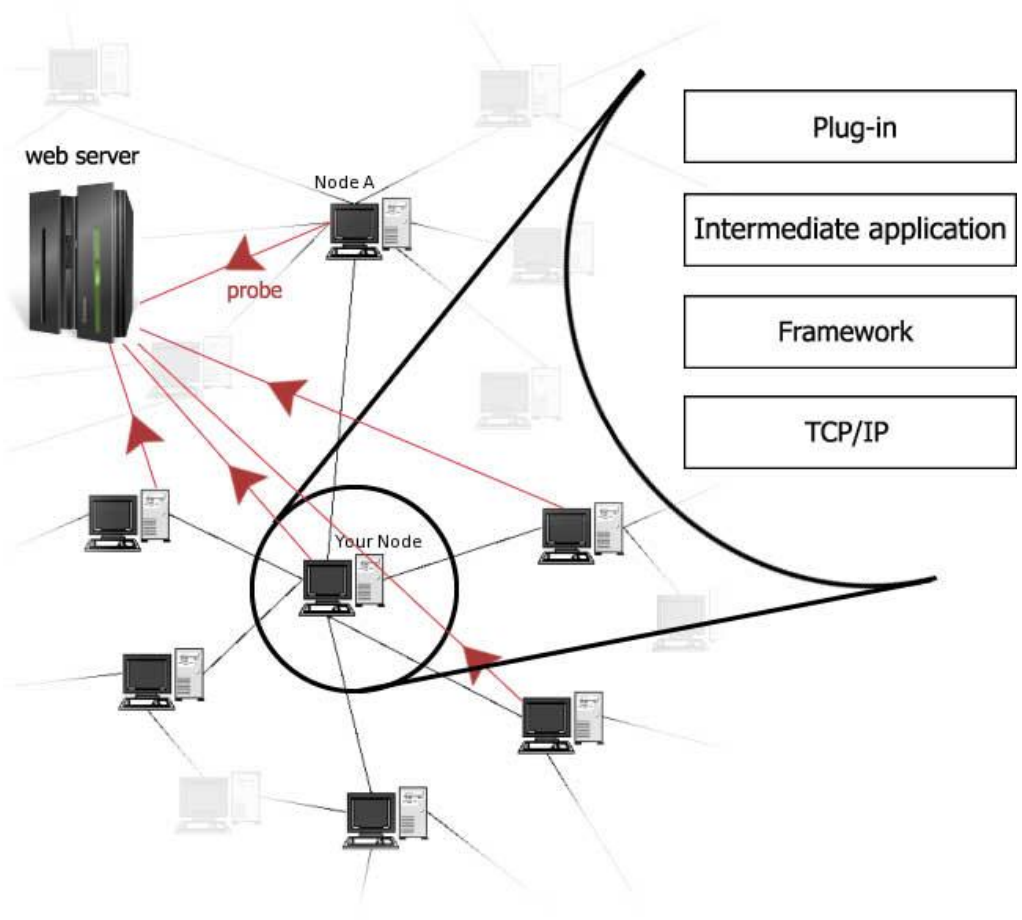

Figure 1 – overview of the component work structure

But in case of the peer application has started it's execution for the first time in that peer, there exists a problem. A problem of unknowing the IP addresses of the any of the peers in the overlay network. As if the new upped peer doesn't know any of the IP addresses of the existence network peer then it cannot be connected to the overlay network of the brute force grid. To address this problem, we have introduced the web server as the centralized body to enable the connection.

As each of the peers send the probes to the server telling that peer is in a connectable state; when a new peer is connected, it can get the other peers IP addresses from the web server and create connections with those peers and make the overlay network connection with them.

2.2 Problem of instability in the centralized systems

There is a problem of stability when the system is centralized. When the system gets centralized, and if the central body fails, the whole system has to be failed. So we were expecting to make the system as decentralized as much as we can. But to address the problem of initial setup of the network we need a central body. So there is no other way to remove this central web server from the system. This leads us to create additional methodologies to address the problems that might arise when the central web server fails. So without removing the web server from our system we gave solutions for the server fail situations, so that the system doesn't need to get crashed even if the server crashes. These solutions include:

- The backup copy of the last connected peers IP list so that when the next time the peer starts it doesn't need to rely only on the web server to fetch the IP list. Assuming that there would be even at least one peer that is being up since the local peer restarts.

- When a peer gets connected to another peer, each peer exchanges the other connectable peer IPs with each other. So that when a new connection is created by the new peer it request the remote peer, the IP list of the peers it has connected to. The remote peer responses the IP list so that the new peer can also make the connections with the peers which are being connected by the remote peer.

Below image describes the actual steps involved in the above mentioned scenario. Even though this doesn't provide a total solution for the non central web server, it does a very good job when the web server is unreachable. So that we can be sure that the system won't get crashed even though the central web server gets crashed.

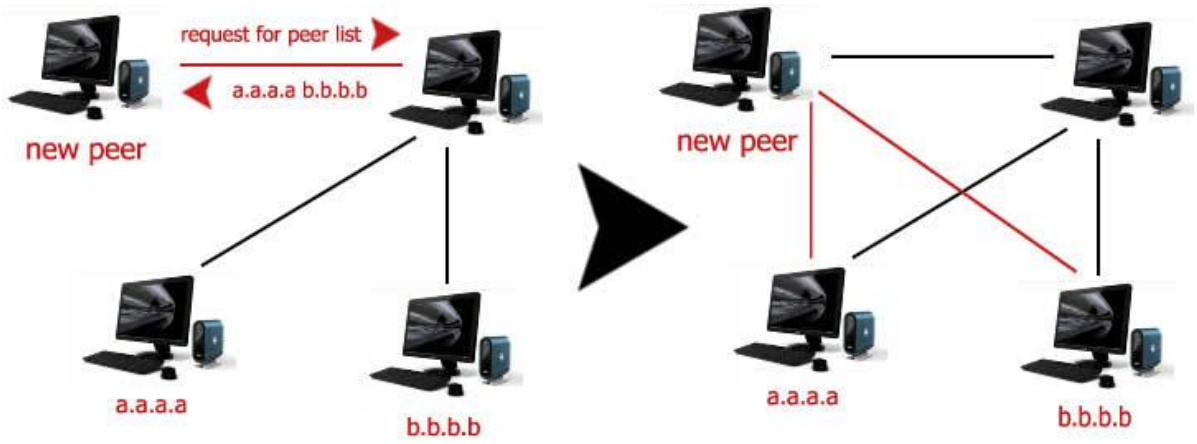

Figure 2 – Request peer-list from connected peers

2.3 Totally new peer turning up when the web server is down

Using these two mechanisms, we can ensure that even the server doesn't exist in the system, the system won't get stuck, and it can work as normal. The only remaining problem is the situation when the server crashes and there is a totally new peer which doesn't have any old connected IP list with it.

To make this problem solve, we have insert a manual way to tryout the peers. That is to enter the IP addresses manually and try to connect to it. If we know the IP address of even a single peer that is sufficient as when the local peer connects to that remote peer, it can request and gain the IP addresses that has been connected to that remote peer and initiate the connections to those peers.

2.4 Inter Peer Probing

The peer which has got turned on doesn't create the connections automatically. This is to utilize the resources available. There is no need of creating a connection when there is no job has to be done. But these peers were probed time to time to determine the number of connectable peers that is in ready on hand when a job is connected. This increase the performance when the peer wants to start a job, as it doesn't need to wait for the searching process for the peers.

2.5 Job Initiating by local peer

When a job gets initiated at a local peer, it fragments the jobs into pieces of sub jobs and keeps it in the local job list. While doing this, the peer connects to the connectable peers that were being probed and ensured upped. After connecting the peers, the local peer tries out the each segment to the peer connected. And when the peer accept the job it sets the sub job as assigned to that peer and wait for the response from that remote peer. The remote peers connected can be distributed the jobs simultaneously using threads and these threads waits till the result is received to the local peer from that remote peer.

There can be several responses after a sub job is assigned to a remote peer. According to the responses return from the remote peer, the local peer manipulates the local sub jobs. These are the responses which can be returned by the remote peer:

- Remote peer response the local peer telling that the job cannot be accepted. The local peer put the sub job to the unassigned state and assigns the sub job to another remote peer connected.

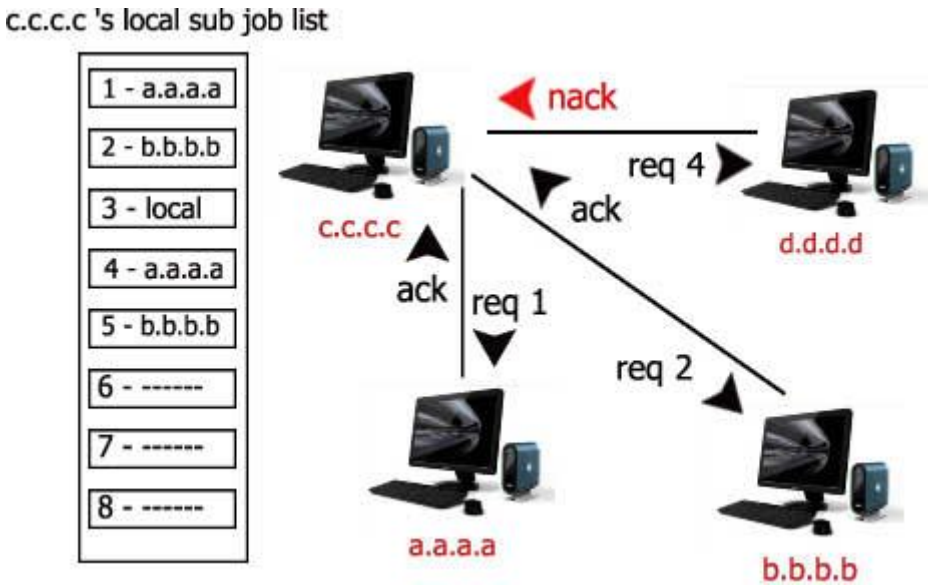

Figure 3 – Assigning sub jobs

"We assign two jobs per peer at a time. When one finishes another is sent to the remote peer and queued. This is to increase the performance by not keeping the remote peer to wait till next sub job is assigned to that peer. This is further discussed later. The local peer assigns a sub job to itself and processes it locally."

- Remote peer response with the finish state, the local peer sets the local sub job to the finish sate and this job is not assigned to another peer again.

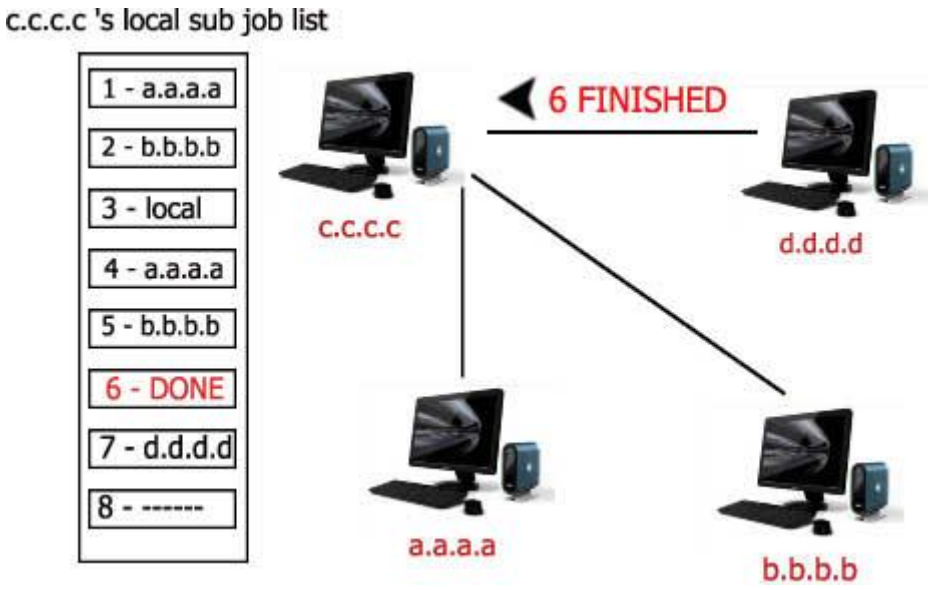

Figure 4 – Finish signal from the remote peer

- Remote host returns the response with a success and the recovered password key with it. At this time, the local peer informs all the other peers connected to stop the jobs provided by the local peer as the job already completed successfully. And the local peer removes all the connected peers and clears the local job list.

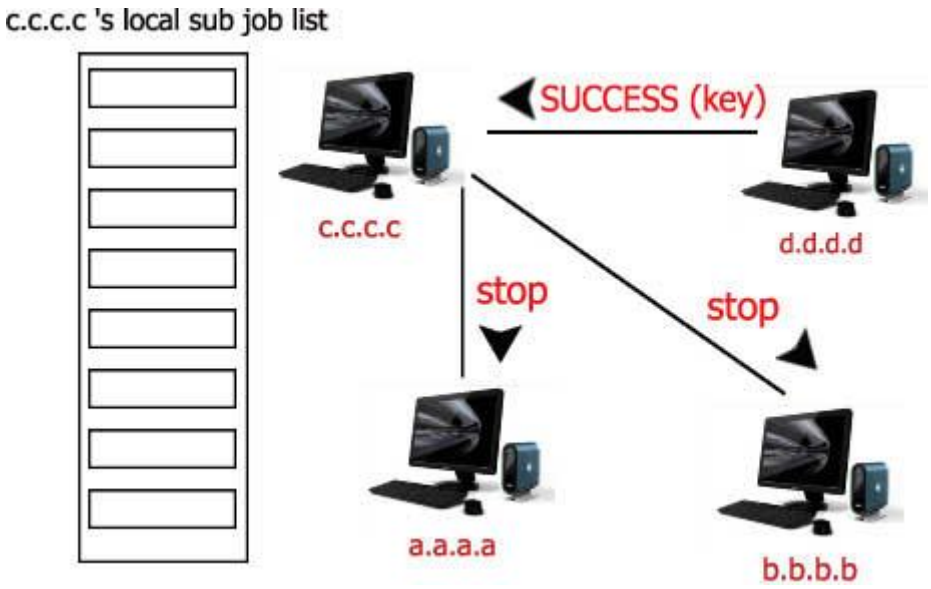

Figure 5 – Success signal from the remote peer

An initiated job can be paused or stopped at any time by the local peer. We have given the local peer the full ability to do this and inform other connected peers to stop the specific sub jobs from this local peer.

If the acknowledgment or negative acknowledgment is not received by the remote peer the local sub job is reset and considered as a new sub job and assigned to another peer.

The connectivity between the peers is available only if the job is distributed between them. If not the connection gets terminated by the peer.

2.6 Remote Job received scenario

Now let's look at the same scenario from the remote peer's perspective. A remote peer is now considered as the local peer. The local peer receives a connection establishment by the remote peer. Then the remote connection is accepted and the connected peer count is incremented by one. Then the remote peer sends the local peer a sub job to execute. The local peer checks the resources and determines whether to accept or decline the request by the remote peer.

This check can be done in various methods. As our project doesn't include this inside the scope, we have kept the interface of this to be developed further by any interested parties.

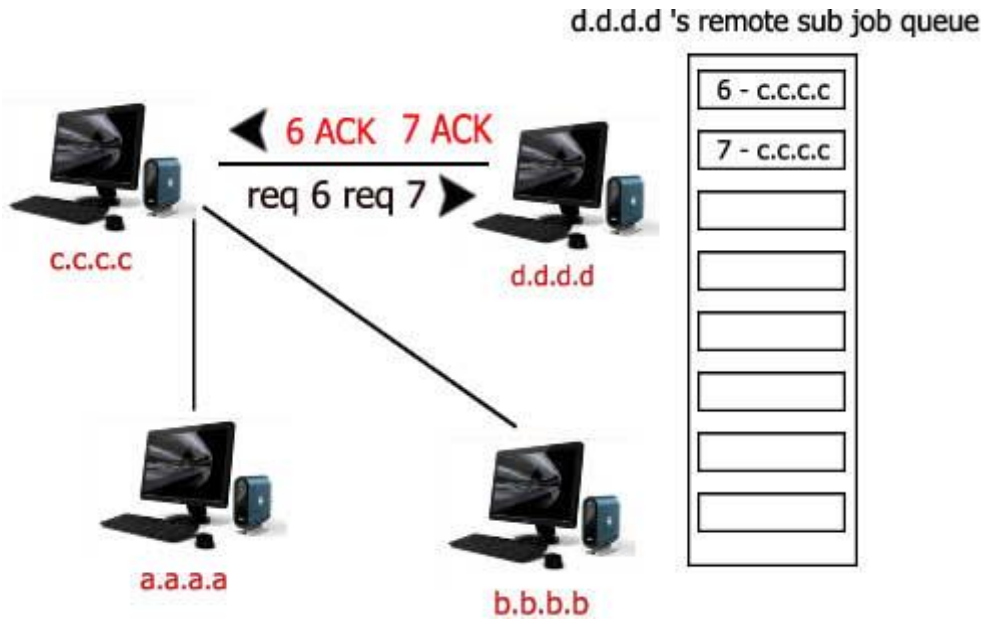

Figure 6 – Receives a new remote sub job

When the determination is confirmed the peer does two things. If accepted the peer acknowledge the remote peer or else is sends a negative acknowledgment saying that it cannot accept any sub jobs at this time.

The connected peer is always checked for the connectivity by sending probes to it. So when a probe sending fails, the remote peer is considered as the drop connected peer and the local resources are freed along with the remote job list's entries made by that peer.

So these create a situation where the started process acknowledged by the remote peer doesn't reply with a finish or a success. This sort of sub jobs were then reset as motioned before.

2.7 Automated plug-in transfer

The plug-ins was used to execute the sub jobs in a peer. So when a peer initiates the process of a specific job that specific peer should have that plug-in within hand. That is for example. If a peer wants to initiate a brute force attack for MD5 hashing algorithm then that local peer should have the MD5 plug-in to work out the sub jobs created by that job.

Apart from this all the peers that is going to execute the sub job should have that specific plug-in. To do this the remote peers should have a mechanism to get the plug-ins from a central location, which is something like a web server.

But as mentioned before such an activity might make the system to be centralized. So we planned to get that plug-in from the peer which initiates the job. But there exists a security issue. That is, the peer which initiates the process can make the plug-in like viruses and send them to other peer via this application.

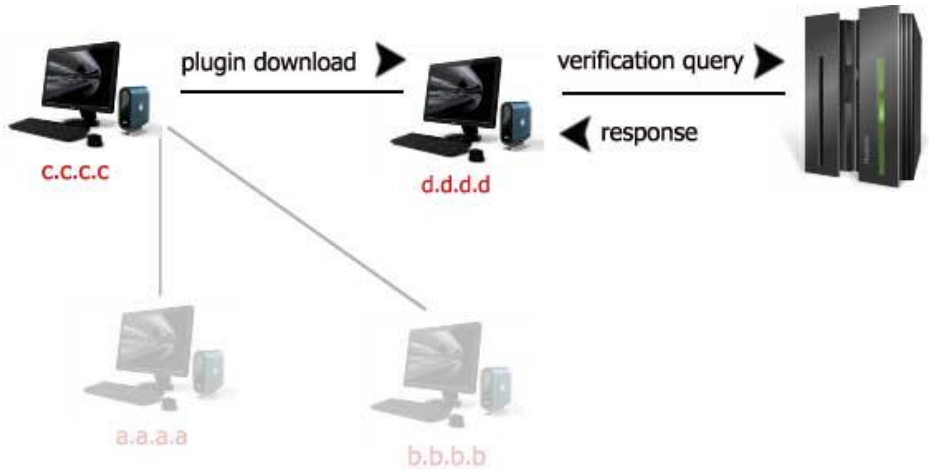

Figure 7 – Plug-in verification from the web server

So we include a security mechanism, which is all the plug-ins registered and certified by the central body, and the central body, that is the web server has the hash of each of this plug-in file. Before executing the downloaded plug-in from the peer, the peer which downloaded the plug-in check the server for the hash of that specific plug-in and confirm that the file is certified by the web server.

But again if the web server has crashed, the peer shows the user a warning message telling that the plug-in downloaded may have viruses and to use on our own risk. So at this point the user has the ability to decline the request along with deleting the remotely downloaded plug-in if he or she thinks that the plug-in may contain malicious codes inside.

4. Results

We did a lot of testing under different environments and platforms to make sure the implementation works well in most cases with no errors and exceptions. We are happy to say we were able to see the desired outcomes, thus proving our objectives of this project and making it a success.

4.1 The Experiments and the Results

The results of the testing can be analyzed under four major test cases. They are,

- The performance deviation with the implementation (application).
- The performance deviation with the segment size.
- The performance deviation with the password size.
- The performance deviation with the password type.
- The performance deviation with the number of nodes in the framework.

4.2.1 The performance deviation with the implementation

In this test case the main focus is to differentiate the performance variation with the application, the performance difference between our solution and some of other leading products which are available in the market. We choose four password recovering applications to compete with our solution.

Those are,

- Cain and Able - #1
- John the Ripper - #2
- L0phtcrack - #5
- Pwdump - #8

* Right hand side shows the current ranking of the application. [19]

The same password (“@xM1”) is given to all application and the time taken for all of them to recover the password is recorded.

Solution	Time (in seconds)

Cain & Able	**12.36**
John the Ripper	**25.09**
L0phtcrack	**1265**
Pwdump	**240.88**
BFG (our)	**Y**

Table 4.1 - The performance deviation with the implementation

4.2.2 The performance deviation with segment size

This test case is related to the one of the node parameters set by the local host, the segment size. We recovered the same password only changing the segment size assigned by the job starter while keeping the other constrains the same. The graph clearly plots out the out put we came up with.

Password = “hel12”

Segment size	Time
1000	**8min 54sec**
10000	**5min 43sec**
100000	**4min 05sec**
1000000	**2min 36sec**
10000000	**0min 18sec**

Table 4.2 - The performance deviation with segment size

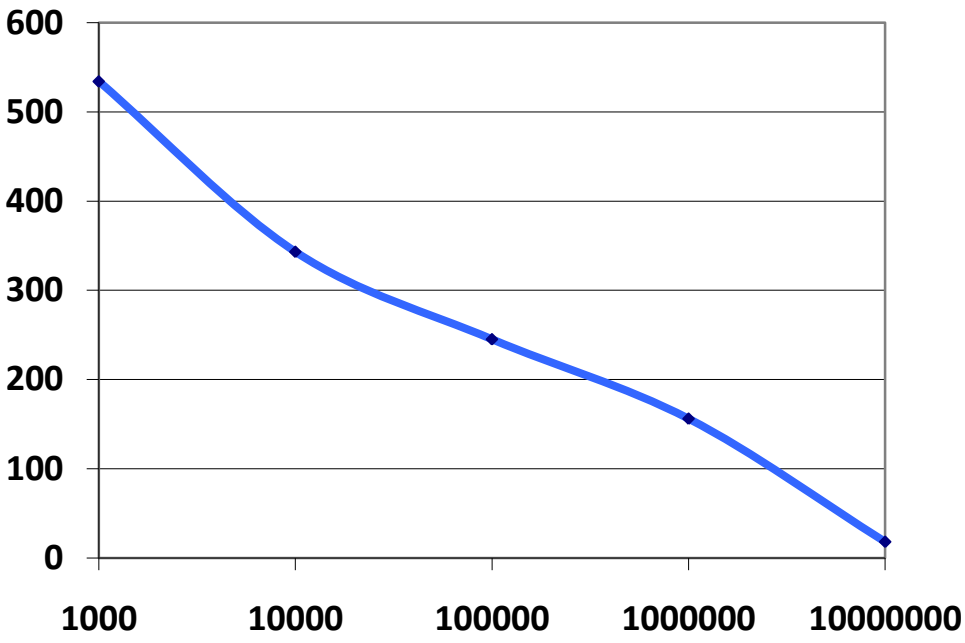

Figure 4.3 - The performance deviation with segment size graph

4.2.3 The performance deviation with password size

In this test case we wanted to check how the performance varies having the password size as the changing parameter while keeping other constrains constants. The time take to recover the range of passwords with different sizes (same type) is recorded. We used the lowercase as the password type. The graph plots out the outputs clearly.

Outputs:

Password size	Time
abc (3)	**0min 4sec**
abcd (4)	**0min 8sec**
abcde (5)	**0min 11sec**
abcdef (6)	**1min 50sec**
abcdefg (7)	**34min 32sec**

Table 4.3 - The performance deviation with password size

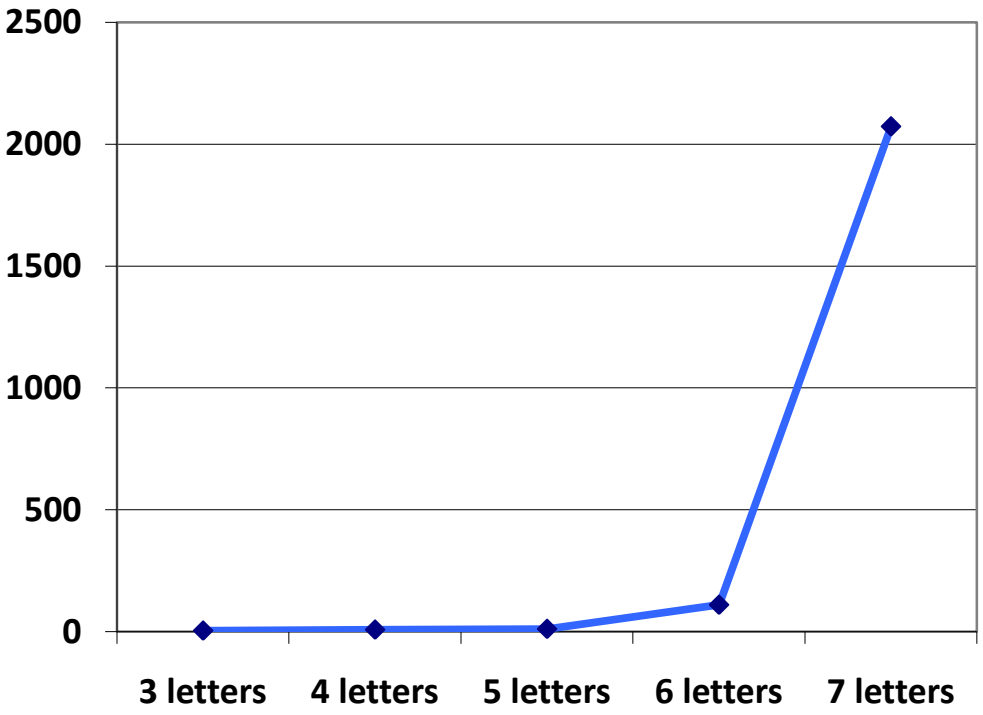

Figure 4.4 - The performance deviation with password size graph

5.2.4 The performance deviation with password type

This test case involves with the password type that the user is interested and how it affects to the performance. Different passwords (type) with the same lengths are recovered while maintaining the other factors the same and the time taken to recover is recorded. The graph plots out the out puts clearly.

* The 1st letter of each range is taken to minimize the searching delay, so it is easy to detect the time variation according to the type

3 peers, segment size - 1000000

Password	Time
aaaa (lower)	**0min 6sec**
aaAA (lower & upper)	**0min 20sec**
aAA1 (lower, upper & numeric)	**0min 53sec**
aA1! (lower,upper,numeric & symbolic)	**4min 19sec**

Table 4.4 - The performance deviation with password type

5.2.5 The performance deviation with number of nodes available

Checking how the performance varies depending on the number of nodes available in the framework is the focus of this phase. We recovered the same password with different number of nodes in the framework. The plots out the out puts clearly.

Password = “helloxy”

Number of nodes	Time
1	**40min 32sec**
3	**16min 44sec**
5	**10min 38sec**
7	**6min 12sec**
9	**4min 45sec**

Table 4.5 - The performance deviation with number of nodes available

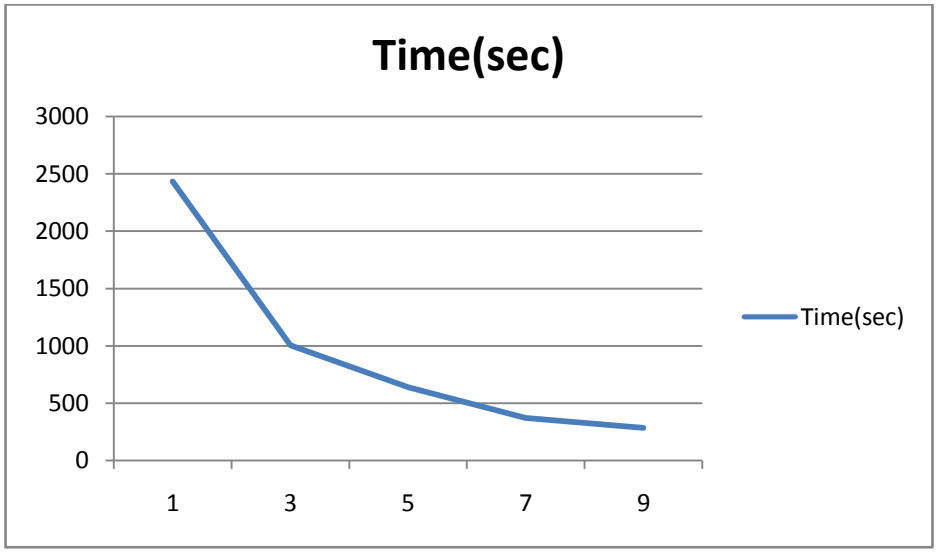

Figure 4.5 - The performance deviation with password size graph

5. Conclusion

5.1 The Analysis

The outputs shown for the experiments done under all the test cases helped us to come up with the conclusions about the performance of our implementation in different aspects.

5.1.1 The performance deviation with the implementation

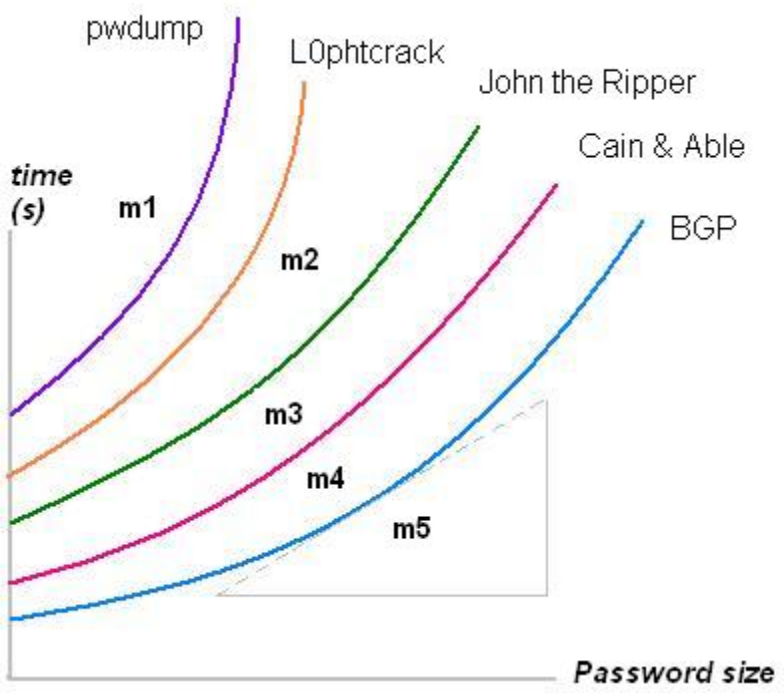

Figure 5.1 - The performance deviation with the implementation

According to the graph, we can see that BFG is the most-efficient solution in the given context by recovering the password in the least amount of time. It has the lowest gradient (m5), so in larger password sizes it shows a better performance than the others. So a user can go for BGP to gets his tasks done quickly and in an efficient manner.

5.1.2 The performance deviation with segment size

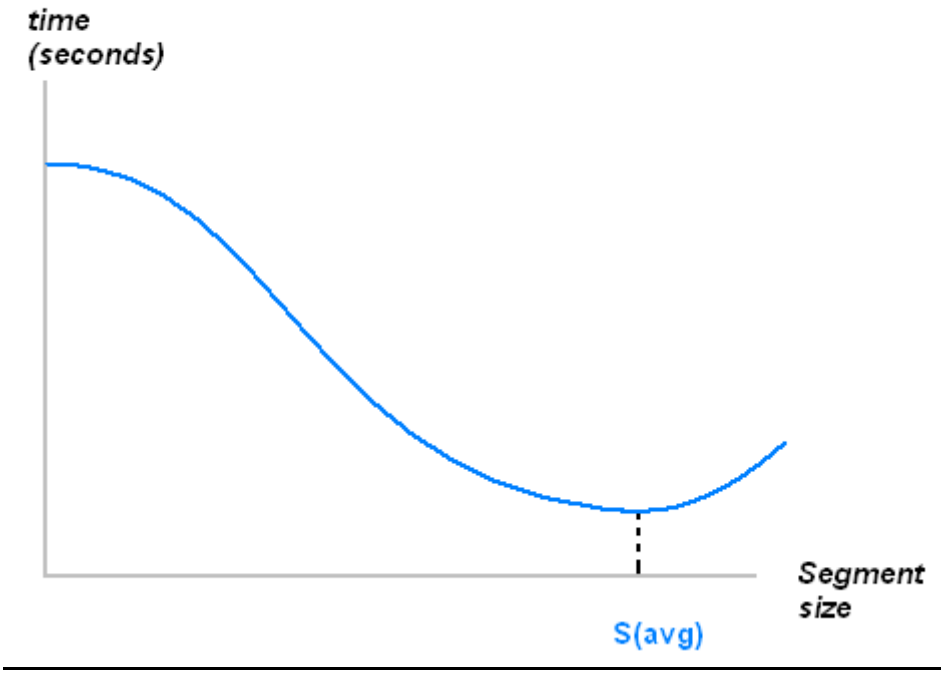

Figure 5.2 - The performance deviation with segment size

As the graph shows, in lower segment sizes the time taken to recover a password is large. This matches the least CPU scenario, thus it has a large number of combinations to compute, and lower numbers of segment allocations is the result of this case. When the segment size increases the process time taken to recover the password decreases, thus bringing the application to its maximum performance (s_{avg}). After this point if the segment size is increased the time also increases. The reason for this is the overhead of the framework implementation. Signal and message passing of the operations is done more than the actual computations done for the brute forcing in this case.

5.1.3 The performance deviation with password size

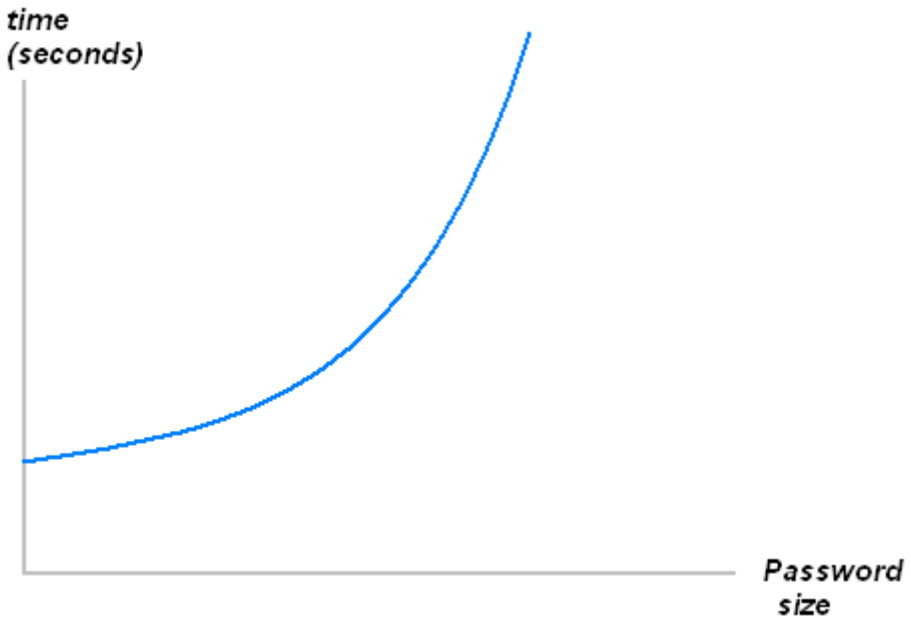

Figure 5.3 - The performance deviation with password size

According to the figure the time increases when the password size increases. But since the password is the user input and the cannot be compromised, we cannot say that “the performance increases when the password size is low”, but comparing with the other products BFG shows a remarkable performance difference at higher password sizes. The conclusion is short passwords are recovered quickly and more time is taken for the longer passwords.

5.1.4 The performance deviation with password type

This test case deals with the performance deviation with the type of the password. Again since this is a user input and cannot be compromised, we cannot say that “the performance increases when the password type is not mixed”, but again comparing with the other products BFG shows a good performance difference with mixed password. The conclusion is more time is taken to recover when the password is a mixed-typed one. Worst case is four types mixed passwords; Symbolic, Numeric, Uppercase and Lowercase. Here the application has to do a full keyboard search.

5.1.5 The performance deviation with number of nodes available

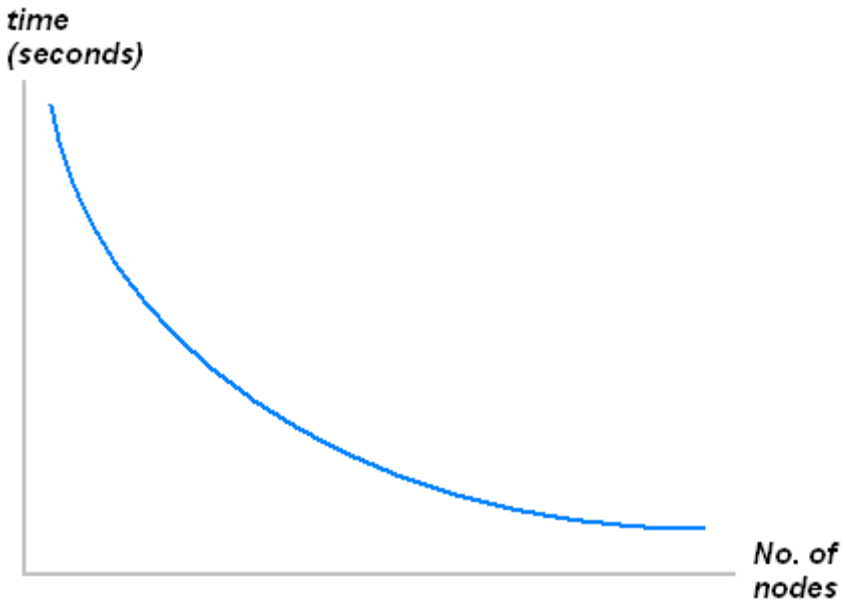

Figure 5.5 - The performance deviation with number of nodes available

How the performance varies with the number of hosts in the in the framework is discussed under this test case. As the graph shows the performance increases as the number of nodes in the framework increases. So user can have more nodes to get a better efficiency.

5.2 The Reasons why BFG is unique

- The efficiency of BFG.
- BFG is a dynamic implementation.
- It supports custom developments.
- BFG is simple
- BFG is a totally free application.

5.2.1 The efficiency

BFG is a unique because of its powerful performance. User can get his password recovered easily and efficiently. This efficiency is highlighted at longer and more complex passwords.

5.2.2 The implementation

BFG is very versatile and can be run in different platforms like LAN, WAN and public networks like internet. So it is a dynamic implementation and can be run anywhere with no architectural complexities found in other distributed solutions.

5.2.3 Supports custom developments

The support for custom developments is another good feature of BFG. Users can develop custom plug-ins and deploy them on the frame work and get their tasks done easily. This makes the framework more dynamic and less application-oriented.

5.2.4 The Simplicity

Simplicity and user-friendliness are important features of BFG. The simple and easy user interface covers up the most complex operations are done under the network level thus giving the user a refreshing feeling with ease and power.

5.2.4 Free!!

BFG is a totally free. Anybody can use to get their tasks done without any charges.

5.3 The Contribution

Peer to Peer network is been used to share files between peers that runs the P2P application. But there has been no actual implementation on the process sharing field using this P2P architecture. There are many written papers and research sheets on this field. But still there hasn't been an implementation in a distributed manner that can be used to have an overlay P2P network on the public internet.

Out project can be the first step towards the field of P2P process sharing, parallel processing technology. We give the software freely and we invite people to join hand in the field of developing this framework to provide efficient and high performance P2P for process sharing.

5.4 Future Enhancements

- Optimize the plug-in.
- Enhance the plug-in database of BFG.
- Use UDP connections for TCP.
- Strengthen the security by using encryption for the signal passing.
- Enhance to make the framework totally application-independent.

5.4.1 Optimizing the plug-in

The current plug-in will be more optimized to do the brute forcing faster. Use of library function will be minimized and the code will be improved with simpler functions with less redundancy and less CPU cycles.

5.4.2 Enhancing the plug-in database.

The current database holds only for the MD5 hashing. In future the plug-in database will be enhanced with brute forcing for more hashing algorithms like MD2, MD4, SHA1, SHA2, SHA 128, SHA 512, DES etc.

5.4.3 Use of UDP when making connections.

We have used CP to transfer signals and other information to the peers in the network. This is much inefficient when we consider about the overheads that includes in the TCP data transfer. We can include simple error checking and sequencing with UDP and use the UDP to send the information. Using UDP we can broadcast the common signals that are being uncast in the current application

5.4.4 Use Encryption for signal passing.

The system security can be increased by using the SSH encrypted connections to transfer important signals and passwords, recovered from the application.

5.4.5 Making the framework application-independent.

Making the framework application-independent will be a big challenge as well as a huge achievement. Not only for password recovering, for all the user-defined custom tasks will be processed by the framework. Dynamic user inputs will be taken, dynamic GUI will be built according to the user inputs and user developed plug-in for his task will be run, thus making the framework totally application-independent

References

[01] Anonymous, What is Hashing, www.webopedia.com, none, October 20, 2008.[Online] Available: http://www.webopedia.com/TERM/h/hashing.html, [Accessed: October 24th 2009]

[02] Anonymous, Overlay Network. Network Overlays, www.overlay-networks.info, none, 2008.[Online] Available: http://www.overlay-networks.info/info.html, [Accessed: October 24th 2009]

[03]. S. Brin, "Extracting Patterns and Relations from the World Wide Web," *Selected papers from the International Workshop on The World Wide Web and Databases*, London, 1999, pp. 172–18

[04]. D.S. Milojičić, et al., "Peer-to-Peer Computing," *Technical Report HPL-2002-57R1*, HP Laboratories, Palo Alto, California, 2002.

[05]. A.W. Loo, "The Future of Peer-to-Peer Computing," *Communications of the ACM*, vol. 46, no. 9, Sep. 2003, pp. 56–61.

[06]. M. Schrage, "Piranha processing - utilizing your down time," *HPC wire (Electronic Newsletter),* Aug. 1992.

[07]. A. Davies, "Computational intermediation and the evolution of computation as a commodity," *Applied Economics*, vol. 36, no. 11, June 2004, pp. 1131–1142.

[08]. D. De Roure, M.A. Baker, N.R. Jennings, and N.R. Shadbolt, "The evolution of the Grid," *Grid Computing: Making the Global Infrastructure a Reality*, F. Berman, G. Fox, and A.J.G. Hey, eds., John Wiley & Sons, Chichester, 2003, ch. 3, pp. 65–100.

[09]. F. Berman, G. Fox, and A.J.G. Hey, "The Grid: Past, Present, Future," *Grid Computing: Making the Global Infrastructure a Reality*, F. Berman, G. Fox, and A.J.G. Hey, eds., John Wiley & Sons, Chichester, 2003, ch. 1, pp. 9–50.

[10]. H. Stockinger, "Defining the grid: a snapshot on the current view," *The Journal of Supercomputing*, vol. 42, no. 1, Oct. 2007, pp. 3–17.

[11].S. Androutsellis-Theotokis and D. Spinellis, "A Sur-vey of Peer-to-Peer Content Distribution Technologies," *ACM Computing Surveys*, vol. 36, no. 4, Dec. 2004, pp. 335–371.

[12]. G.M. Amdahl, “Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities,” *AFIPS Conference Proceedings*, AFIPS Press, vol. 30, Apr. 1967, pp. 483–485.

[13]. J.L. Gustafson, “Reevaluating Amdahl’s Law,” *Communications of the ACM*, vol. 31, no. 5, May 1988, pp. 532–53

[14]. D.S. Milojičić, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou, “Process Migration,” *ACM Computing Surveys*, vol. 32, no. 3, Sep. 2000, pp. 241–299.

[16]. I. Foster and A. Iamnitchi, “On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing,” *Proceedings of the Second International Workshop on Peer-to-Peer Systems* (IPTPS ‘03), Berkeley, 2003, pp. 118–128.

[17]. V. Donaldson, F. Berman, and R. Paturi, “Program Speedup in a Heterogeneous Computing Network,” *Journal of Parallel and Distributed Computing*, vol. 21, no. 3, June 1994, pp. 316–322.

[18]. C. Perez, “Technological Revolutions, Paradigm Shifts and Socio-Institutional Change,” *Globalization, Economic Development and Inequality:* An Alterna-tive Perspective, E. Reinert, ed., Edward Elgar, Chel-tenham, 2004, pp. 217–242.

[19] Anonymous, Top 10 password crackers, sectools.org, none, 2008.[Online] Available: http://sectools.org/crackers.html, [Accessed: October 25th 2009]