

UNIVERSITY OF ZAGREB  
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

## **Cluster Manager**

[github.com/antivo/Cluster-Manager](https://github.com/antivo/Cluster-Manager)

*Stjepan Antivo Ivica*

*Supervisor: Prof. dr. sc. Domagoj Jakobović*

Zagreb, February, 2014

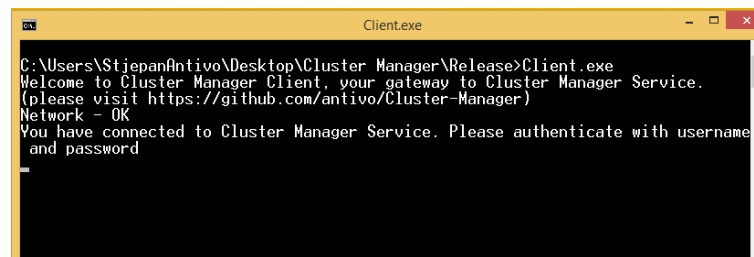


**Content**

- 1. User Manual.....1
  - 1.1 Scheduling a job .....2
  - 1.2 Retrieving info .....3
  - 1.3 Retrieving job results.....4
  - 1.4 Quit .....4
- 2. Developer's guide.....5
- 3. Code Overview.....7
- 4. Conclusion .....11

# 1. User Manual

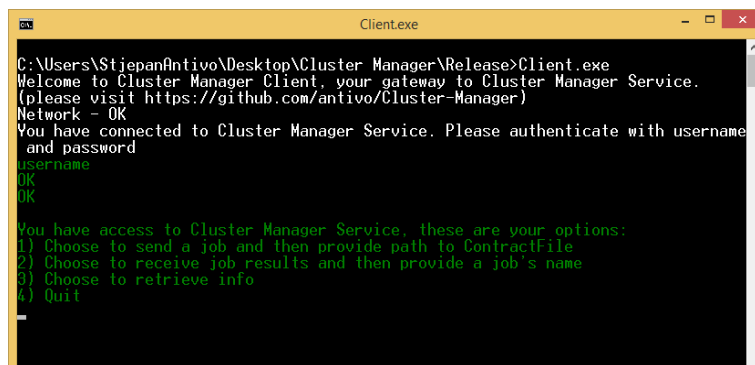
Client is available for the Windows OS. You can use Client to schedule jobs, retrieve executed job data or watch over your scheduled jobs status. Start your Client. Client will print the welcome message and try to connect to Server as shown in Figure 1 Client started up and connected to Server. When Client connects to the server it will inform you that your network connection is OK and ask you for your authentication details.



```
C:\Users\StjepanAntivo\Desktop\Cluster Manager\Release>Client.exe
Welcome to Cluster Manager Client, your gateway to Cluster Manager Service.
(please visit https://github.com/antivo/Cluster-Manager)
Network - OK
You have connected to Cluster Manager Service. Please authenticate with username
and password
```

*Figure 1 Client started up and connected to Server*

As soon as Client enters authentication step color of console text will turn to green as shown in Figure 2. You should have been provided with your authentication details. After you input your username Client will notify you your username was registered. After username input your password, letters will not appear in console, this is done to protect your privacy. After you input your password, client will notify you that your password was registered. After the successful authentication Client will provide you with a menu of choices you can make. You can schedule a job, retrieve data from finished job, obtain status of your jobs or exit. Choices are made by entering number under the desired choice.

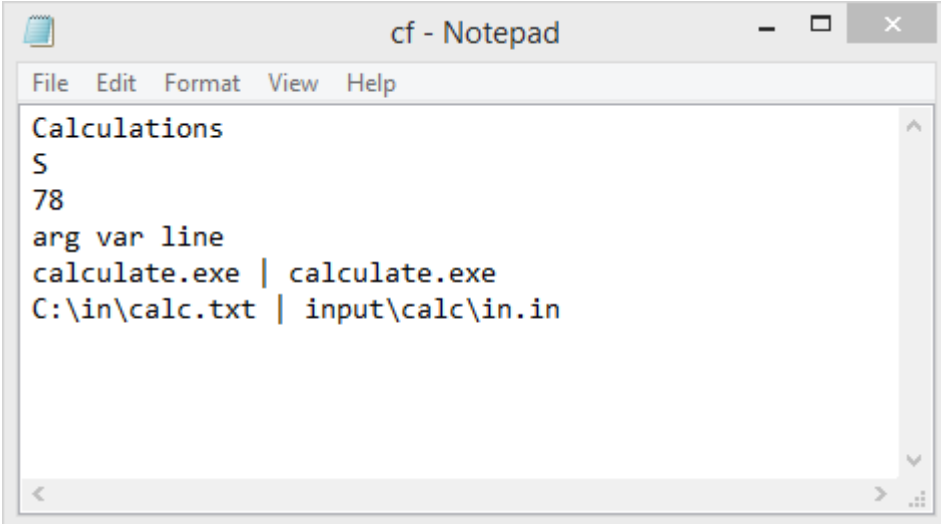


```
C:\Users\StjepanAntivo\Desktop\Cluster Manager\Release>Client.exe
Welcome to Cluster Manager Client, your gateway to Cluster Manager Service.
(please visit https://github.com/antivo/Cluster-Manager)
Network - OK
You have connected to Cluster Manager Service. Please authenticate with username
and password
username
OK
OK
You have access to Cluster Manager Service. these are your options:
1) Choose to send a job and then provide path to ContractFile
2) Choose to receive job results and then provide a job's name
3) Choose to retrieve info
4) Quit
```

*Figure 2 Client authentication step and choice menu*

## 1.1 Scheduling a job

Contract – File must be prepared in order to schedule a job. Contract file is an textual file with defined structure. Example of contract file can be seen in Figure 3 Contract – File example It must contain at least five lines, and the maximum size of lines is not limited. First line is a name of your job, name it wisely, it will be your indentifier for the job. Second line defines job type. There are two job types in the current version serial and parallel where serial jobs are defined with a character 'S' and parallel jobs with a character 'P'. Third line defines how much workers do you want to be used. Fourth line contains arguments given to every instance of your desired job. In current version it is forbidden to use character '#' in the argument line. Fifth line defines entry point for the job execution. In this version only exe files are supported. From the fifth line onwards files to be tranfered are defined. Files to be transferred definition contains relative or apsolute path to the desired path on your local machine separated by a character '|' from relative path how it will be organized on a Cluster Manager worker.

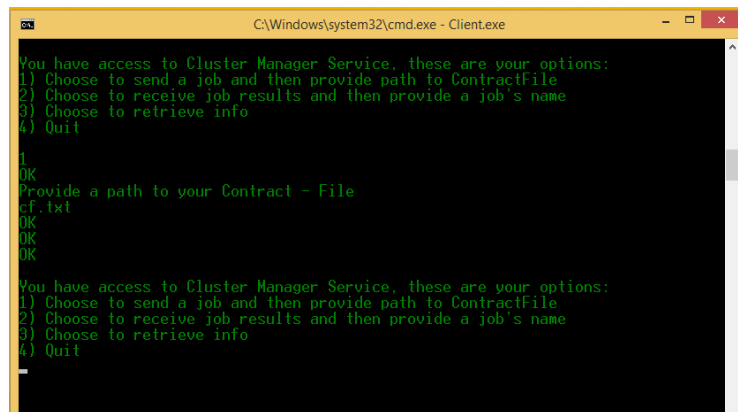


```
Calculations
S
78
arg var line
calculate.exe | calculate.exe
C:\in\calc.txt | input\calc\in.in
```

*Figure 3 Contract – File example*

After choosing to schedule a job Client will ask you to provide relative or apsolute path to the Contract – File as shown in Figure 4 Client scheduling a job. If the contract file is well formed and correct Client will inform you with success messages

for the contract file itself and for every file transferred. After scheduling a job choice menu will return.



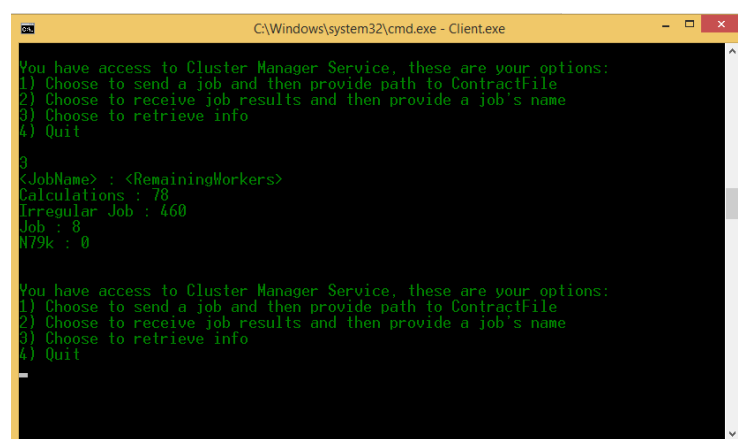
```
C:\Windows\system32\cmd.exe - Client.exe

You have access to Cluster Manager Service, these are your options:
1) Choose to send a job and then provide path to ContractFile
2) Choose to receive job results and then provide a job's name
3) Choose to retrieve info
4) Quit
1
OK
Provide a path to your Contract - File
cf.txt
OK
OK
OK
You have access to Cluster Manager Service, these are your options:
1) Choose to send a job and then provide path to ContractFile
2) Choose to receive job results and then provide a job's name
3) Choose to retrieve info
4) Quit
3
```

*Figure 4 Client scheduling a job*

## 1.2 Retrieving info

After choosing to retrieve info, Client will represent retrieved information as shown in Figure 5 Client represents job information. For every unretrieved job you have scheduled on a cluster manager Client will write a job's name and the number of remaining workers that will have to execute the job. When that number becomes zero, job is finished and ready for retrieving job results. After representation of information Client will return the choice menu.



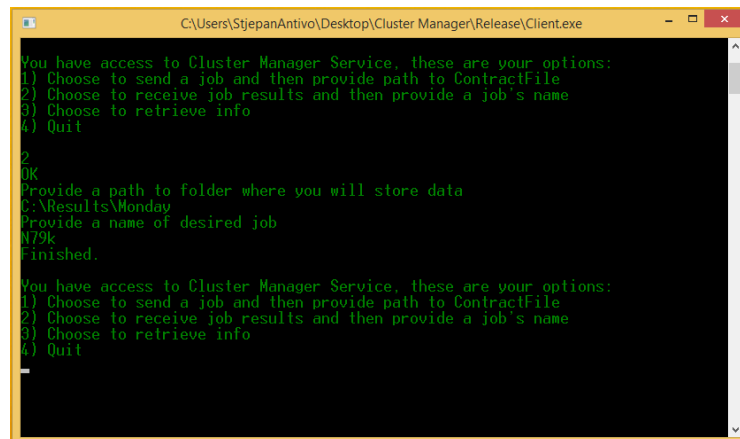
```
C:\Windows\system32\cmd.exe - Client.exe

You have access to Cluster Manager Service, these are your options:
1) Choose to send a job and then provide path to ContractFile
2) Choose to receive job results and then provide a job's name
3) Choose to retrieve info
4) Quit
3
<JobName> : <RemainingWorkers>
Calculations : 78
Irregular Job : 460
Job : 8
N79k : 0
You have access to Cluster Manager Service, these are your options:
1) Choose to send a job and then provide path to ContractFile
2) Choose to receive job results and then provide a job's name
3) Choose to retrieve info
4) Quit
3
```

*Figure 5 Client represents job information*

### 1.3 Retrieving job results

When retrieving a job, Client will expect you to input relative or absolute path to the directory where you want to store the data and the name of the selected job as shown in Figure 6 Client retrieves job results. After Client retrieves all the data he will inform you and bring the choice menu.



```
C:\Users\StjepanAntivo\Desktop\Cluster Manager\Release\Client.exe

You have access to Cluster Manager Service, these are your options:
1) Choose to send a job and then provide path to ContractFile
2) Choose to receive job results and then provide a job's name
3) Choose to retrieve info
4) Quit

>
OK
Provide a path to folder where you will store data
C:\Results\Monday
Provide a name of desired job
479k
Finished.

You have access to Cluster Manager Service, these are your options:
1) Choose to send a job and then provide path to ContractFile
2) Choose to receive job results and then provide a job's name
3) Choose to retrieve info
4) Quit

_
```

*Figure 6 Client retrieves job results*

### 1.4 Quit

After the order for exit has been issued, Client will disconnect from the server and return the screen text color to the color that was before Client connected to the server. Notify you of your disconnection and ask you to press any key before it closes.

## 2. Developer's guide

At the Department of Electronics, Microelectronics, Computer and Intelligent Systems (ZEMRIS) there is a cluster of computers connected through LAN with Windows OS. They are used by researchers for execution of jobs on a single node executed multiple times (for statistics) and parallel MPI jobs executed over multiple nodes. Cluster Manager organized deployment, execution and retrieval of data for users.

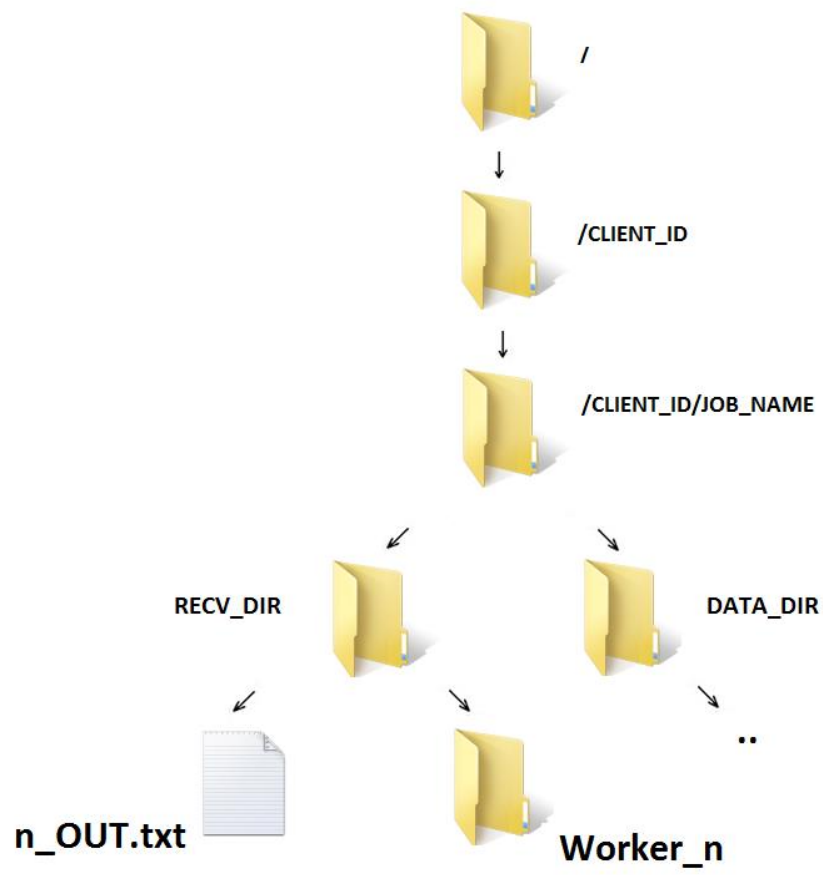
Cluster Manager components are Client, Server, Manager and Database. Client is used for interaction with the Server over network. Client organizes sending and receiving of job data and status representation for scheduled jobs from Server over network. More than one user should be able to run Client and connect to the Server at the same time.

Database is used to store persistent data about jobs and clients. In the database there are two tables `clients` and `jobs`.

Server handles connection, authentication and requests from Clients. When Client is scheduling a job Server stores data in the database and stores file in the data directory for that job, this can be observed in Figure 7 Filesystem. When Client requests to retrieve job data Server retrieves all the data from recv directory that can be seen in Figure 7 Filesystem and after successful transaction removes data from database. When Client makes query for information Server obtains data from database and forwards it.

Manager manages nodes in the cluster. Manager and Server communicate over database. Periodically Manager checks if there are new assignments that can be executed. Manager executes jobs that he can organize to be executed. When executing a job he copies all the data to the working node and orders job execution. Output of job is redirected to file. After the job is executed, manager supervises when the job is finished. When job is finished data is transferred from the working nodes to the recv directory in filesystem as seen in Figure 7 Filesystem.





*Figure 7 Filesystem*

### 3. Code Overview

Target OS is Windows. System was developed using Visual Studio 2015. Solution is composed from twenty five projects. In this chapter summary and overview of projects is provided. Inspiration for code was the determinism of C++. When constructor is called resources are acquired, when destructor is called resources are freed. Composition and delegation are preferred over inheritance.

Projects **Client**, **Server** and **Manager** are projects that produce executable file and are considered as entry points for those components. Other projects generate library files.

**Assert:** Exceptions are used. Motivation was simple. Function should not return error, it should throw and in that case the destruction of objects on the stack should take care of the cleanup. With this approach, block “ if(!condition) throw exception; “ (where condition is boolean data type and exception is object to be thrown) becomes repetitive through the code.

Project assert takes responsibility for assertion of condition and throwing the appropriate exception when certain condition is not met. With this approach intention is to remove the need for the use of the throw keyword outside of this project when exceptions arise.

Problems with current approach is additional generation of objects before checking the condition. In general using exception has its downsides (for example if constructor throws within function template that makes smart pointers, exception on top will be bad\_alloc, the real exception will be swallowed).

**Configuration:** Project Configuration was created to store configuration settings that should be available across Client, Manager and Server components during the compilation. Project contains hardcoded data, there should be no hardcoded data outside of this project.

**Console:** There is no GUI in the current version. Interface is terminal.

There are two RAII classes for terminal console manipulation InputDisabler and ColorManipulator.

**Database:** Fourth component beside Client, Server and Manager is Database. In a previous iteration of this system, MySql database was used. (It was a few years ago, early 2012. Choice of database was random). The dependency for code to connect to database was relatively low but it was rudimentary and written in C.

Project is made to encapsulate access to MySql database and make it safer and easier.

**Entity:** Project Entity acknowledges existence of sub components in existing architectural solution. To register a job on a Server user prepares a Contract – File of defined format. Contract – File defines job to be executed, files to be transferred and topology of transferred files. Beside defined format there are other constrains. Contained information has to be valid and Server uses a subset of information in original Contract – File. Information is post-processed before being transferred to Server.

**Exception:** To obtain information from errors during the runtime class exception in the project exception was formed. Project handles retrieving error information from operating system.

**Filesystem:** Performing operations on folders and files is crucial component of this system. Cluster Manager has defined usage of filesystem and how job directories should be organized.

**Locked:** Components are multithreaded. There is a need for synchronized structures.

**Memory:** Memory management is an important topic and deserves its own project. Entire solution was started by the end of 2012 following the then-new C++11 standard and the Visual C++ Compiler November 2012 CTP. At that time there was no `std::make_unique` and no RAII classes for dynamically allocated arrays.

Project is updated so that unnecessary memory management tools are removed and calls to them are replaced with standard implementations. But the idea of project remains. There should be no keywords new and delete outside of this project.

**Network:** Components Client and Server exchange messages over network.

Elementary functions send and receive and functions that operate on string sendMessage and receiveMessage. RAI class with reference counting named Device to initialize Winsock. RAI class Socket to contain file descriptor of socket.

**Utility:** Various helping functions are needed. Like functions that operate on string and extract or provide information from it or some general activity functions. But there are also functions that are very important but not developed enough to have their own project such as serialization and deserialization.

**Client\_Network:** Uses components from project Network to create Client side connection to the Server.

**Manager\_Entity:** Subcomponents regarding worker representation and different stages in job execution cycle.

**Manager\_Manager:** Main Manager component that encapsulates Manager's threads.

**Manager\_Persistent:** Manager has to be able to manipulate persistent data in database. Project provides interface for access to persistent data and its implementation.

**Manager\_Thread:** Manager has two main threads. Thread that executes jobs and thread that supervises executed jobs. Execution of jobs is done in separate threads. Manager creates additional thread per executed job.

**Server\_Entity:** There are entities that are necessary subcomponents of Server. Entities like Deterministic Finite Automaton and records of active Clients that are being connected to the Server.

**Server\_Locked:** Server is multithreaded. Server stores information about the connections that are active and records for active connections. Access to those containers must be synchronized.

**Server\_Network:** Uses components from project Network to create component that accepts connection with the Client.

**Server\_Persistent:** : Server has to be able to manipulate persistent data in database. Project provides interface for access to persistent data and it's implementation.

**Server\_Server:** Server has four layers. First layer named Listening Server is aware of responding and accepting Clients. Second layer named Logging Server is able to store data about it's clients. Third layer named Instructed Server expects transition function in order to behave as Deterministic Finite Automaton. Fourth layer named Detailed Server has fully defined behaviour but in order to run it is expecting information about persistent data source.

**Server\_Thread:** Server has two main threads. Thread that accepts Clients and thread that responds Clients.

## 4. Conclusion

Main motivation was to create a potentially useful project written in modern C++ open for contribution.

There are problems that were not solved or were created due to taking the simple approach. Because of simple procedure for serialization and deserialization, usage of character '#' in Contract – File is forbidden. When Server sends job data to Client, even if the error occurs during the transaction of the last file, server will consider job as being sent. Empty files can not be sent over network. If server is sending empty file it will fill it with default constant content. If client is sending empty file Client will notify user and abort action. Due to policy for irregular behavior from client not being decided, for every irregular behavior server will automatically disconnect Client.

**Keywords:** Cluster Manager, client – server architecture, master – slave architecture, Object oriented design, C++, networking, multithreading