Requirements and Analysis Document for NNN

Table of Contents

Version: 0.1

Date: 2013-03-26

Author: Victor Nilsson

This version overrides all previous versions.

1 Introduction

A computer can not perform calculations any faster than its hardware allows, that is common sense. However, if we allowed computers to share the burden of a particular computation, it would be able to process the same amount of data in considerable less time. That was, at the very least, our line of reasoning when we decided on making a set of applications that would allow computers cooperate in order to solve those bigger problems more efficiently.

1.1 Purpose of application

The purpose of the application is to shorten the time needed for heavy calculations by distributing those calculations across multiple computers.

1.2 General characteristics of application

This application is, in fact, three separate applications. There is a *client* application, which sends requests to a computer running the *server* application that it wants some specific task to be done. Then there is the *worker* application, which is mostly like drone that does what the server tells it. Whenever the workers have completed a task, they will send the result back to the server which in turn returns it to the client who requested it in the first place. Task are defined in the client application, either by choosing from a pre-made set of calculations or by writing actual code in python.

1.3 Scope of application

The application will, in the end, probably not be much more than a prototype of a proper system for work distribution. One of the things that this system will lack is the ability to split a bigger computation into smaller workloads. More likely is that we will be able to provide the client with a tasklist, and divide those tasks among the available workers. We will, however, probably be able to run code snippets that the client specifies. You could

say that this application will be able to handle arbitrary operations, but in order to do this properly and efficiently we will probably have to establish proper methods for distributing every kind of task properly. The pre-made list of tasks will probably be simple, but heavy, calculations mostly in place for testing and demonstration purposes.

1.4 Objectives and success criteria of the project

Our objective with this project is easiest explained by providing a simple example. Let us assume that we have a list containing a couple of thousand numbers, each number having at least six digits. If our system can calculate the prime divisors of these numbers faster than an individual computer, then we have really succeeded. Even a slightly slower system is an accepted accomplishment however, as long as we can get the distribution to work properly and see that additional computers connected to the server results in less time spent calculating. A properly established connection between our three applications is the first goal for the project, the next being the ability to send tasks over the aforementioned network.

1.5 Definitions, acronyms and abbreviations

We will probably write most about the client, the server and the workers. The client is, in this case, is the application that will be used to define what calculations the system should work on, and will also be used to define the manner in which the work is split. The server is the middle hand of the client and its workers, and will be useful for managing the system in the case of multiple client connections. The worker is the computer hosting the worker application, and will perform the vast bulk of the actual computations. One can think of the complete system as a simple consultant firm, wherein the client is, well, the client, the server is the actual company that hands out work to its workers (consultants). We will use the term “task” whenever we reference to a calculation that the client wants to be done by the workers; in other words, the whole progress from client request to client reception of a result will be referred to as a task. We will also refer to a particular workers current part of a task as its “workload”.

While we refer to the applications by different names, we will also refer to the users of each application with separate names. The client user is simply the *user*, the server user is the *administrator*, and the person(s) who are providing the workers are the *volunteers.*

2 Requirements

?

2.1 Functional requirements

The most basic functions are the ability to define a task to be processed (the example previously made with finding the prime divisors, for instance), and then be able to follow this task through the whole progress for demonstration purposes. This is done by providing the client with a code snippet or a prewritten file containing code. While mentioning the functions that are mostly for demonstration purposes, a client should also be able to communicate with the workers, done mostly to show that the connection is up and running. This will also include showing the current status of the tasks at hand, for instance showing the number of tasks currently queued by the worker and, if applicable, how much of the current task that still needs to be processed.
The client should be able to stop a task in progress whenever he deems necessary, while a worker should be able to stop the part of the task given to it. In the case of a worker cancelling its task, the workload of said worker should be returned to the server in order to redistribute it to another computer. The worker should also be able to at any point deny any further work, and focusing on completing its current workload. And of course, the client will be able to choose (and in some ways customize) a number of preconstructed tasks to send to the workers.
For the more diabolical user, or rather as a way to test our error handling, the server should be able to shut down without doing anything more than provide an error message, in order to see what happens with the tasks.

2.2 Non-functional requirements

Possible NA (not applicable).

2.2.1 Usability

This application will probably require a basic understanding of python in order to use customized tasks. Otherwise the GUI should at least be clear and concise in its usability.

2.2.2 Reliability

NA

2.2.3 Performance

Connection should, above all, be stable. The transfer speed will hopefully not be a problem.

2.2.4 Supportability

We do, of course, aim to make the program as expandable as possible when it comes to adding new predefined tasks to the client.

2.2.5 Implementation

NA

2.2.6 Packaging and installation

NA

2.2.7 Legal

NA

2.3 Application models

2.3.1 Use case model

Abandon workload (Worker)
Abort current task (Client)
Establish connection
Kill (Worker)
Process customized tasks
Process predefined task
Show current status (Worker)
Show current status (Client)
Show current status (Server)
Show statistics (Worker)
Show statistics (Server)
Show statistics (Client)
Shut down (Worker)
Shut down (Server)

2.3.2 Use cases priority

High:

Establish connection
Process predefined task (Client)
Shut down (Server)
Start (Server)

Medium:

Abort current task (Client)
Kill (Worker)
Process customized tasks (Client)
Show current status (Worker)
Show current status (Client)
Show current status (Server)

Low:

Abandon workload (Worker)

Shut down (Worker)
Show statistics (Worker)
Show statistics (Server)
Show statistics (Client)

2.3.3 Domain model

UML, possible some text.

2.3.4 User interface

Text to motivate a picture.

2.4 References

APPENDIX

GUI

Domain model

Use case texts