Software Requirements Specification

for

Peer-to-Peer Distributed Computing Framework

Version 1.0 approved

Prepared by  
Project ID: 16-023   
  
Prashan Dharmapala IT13037116  
Sri Darshanun IT13002466  
Imtizam Ismail IT13029364  
K. Lumeshkantha IT13071608

Sri Lanka Institute of Information Technology

**11/4/2016**

**Table of Contents**

[List of Tables iv](#_Toc447871536)

[List of Figures iv](#_Toc447871537)

[1. Introduction 1](#_Toc447871538)

[1.1 Purpose 1](#_Toc447871539)

[1.2 Scope 1](#_Toc447871540)

[1.3 Definitions, Acronyms, and Abbreviations 1](#_Toc447871541)

[1.4 Overview 2](#_Toc447871542)

[2. Overall Descriptions 3](#_Toc447871543)

[2.1 Product perspective 3](#_Toc447871544)

[2.1.1 System interfaces 5](#_Toc447871545)

[2.1.2 User interfaces 5](#_Toc447871546)

[2.1.3 Hardware interfaces 6](#_Toc447871547)

[2.1.4 Software interfaces 6](#_Toc447871548)

[2.1.5 Communication interfaces 6](#_Toc447871549)

[2.1.6 Memory constraints 6](#_Toc447871550)

[2.1.7 Operations 7](#_Toc447871551)

[2.1.8 Site adaptation requirements 7](#_Toc447871552)

[2.2 Product functions 8](#_Toc447871553)

[2.2.1 Use case diagram 8](#_Toc447871554)

[2.2.2 Use case scenarios 9](#_Toc447871555)

[2.3 User characteristics 13](#_Toc447871556)

[2.4 Constraints 13](#_Toc447871557)

[2.4.1 Hardware limitations 13](#_Toc447871558)

[2.4.2 Time constraints 13](#_Toc447871559)

[2.4.3 Criticality of the application 13](#_Toc447871560)

[2.4.5 Parallel operation 14](#_Toc447871561)

[2.5 Assumptions and dependencies 14](#_Toc447871562)

[2.6 Apportioning of requirements 14](#_Toc447871563)

[3. Specific requirements 15](#_Toc447871564)

[3.1 External interface requirements 15](#_Toc447871565)

[3.2 Functions 15](#_Toc447871566)

[3.3 Performance requirements 16](#_Toc447871567)

[3.4 Logical Database Requirements 16](#_Toc447871568)

[3.5 Design constraints 16](#_Toc447871569)

[3.6 Software System Attributes 16](#_Toc447871570)

[3.6.1 Reliability 16](#_Toc447871571)

[3.6.2 Availability 17](#_Toc447871572)

[3.6.3 Security 17](#_Toc447871573)

[3.6.4 Maintainability 17](#_Toc447871574)

[3.6.5 Portability 17](#_Toc447871575)

[4. Supporting information 18](#_Toc447871576)

[4.1 References 18](#_Toc447871577)

# List of Tables

Table 1.3.1: Definitions, acronyms, and abbreviations 1

Table 2.1.1: Feature comparison 4

Table 2.2.1: Functional requirement for start network 9

Table 2.2.2: Functional requirement for connect network 10

Table 2.2.3: Functional requirement for add task 10

Table 2.2.4: Functional requirement for abort task 11

Table 2.2.5: Functional requirement for execute task 11

Table 2.2.6: Functional requirement for abort execution 11

Table 2.2.7: Functional requirement for pause execution 12

Table 2.2.8: Functional requirement for set computing preference 12

Table 2.2.9: Functional requirement for receive task 12

Table 2.2.10: Functional requirement for view results 13

# List of Figures

Figure 2.1.1: Main interface for P2P system 5

Figure 2.1.2: Status interface for P2P system 6

Figure 2.2.1: Use case diagram for P2P system 8

# Introduction

## Purpose

This document will discuss the main requirements, the features that are planned to be implemented to the system, the system operations and how the users interact with the application. A detailed description of the functional and non-functional requirements can be found. There will also be explained the system constraints, technical dependencies, interface and interactions with external applications. This will help developers to gain knowledge about the functionality of the system while designing and implementing it. It will also be of use during any maintenance or enhancement in future.

The intended audience for this document will be the project team and others who use this document for reviewing purposes. It can also be referred by the users of the system and as a rough operational manual.

## Scope

This document covers the requirements for the Peer-to-Peer Distributed Computing Framework. The purpose of this document is to pilot the project team in selecting a design that will be able to provide space for the full-scale application. This guides developers to select a design to identify software limitations and technical challenges.

The main outcome is to come up with a common code providing generic functionality that can be selectively specialized to a particular problem area. This will be the framework for the P2P system. It should be globally scalable and popular among the general population as a way to access high performance computing. The second task is to develop a user agent software to demonstrate the functionality of the framework.

Not all problems can be broken down into perfectly parallel tasks. This framework focuses on embarrassingly parallel workloads only. These may include rendering of computer graphics, discrete Fourier Transform, BLAST searches in bioinformatics, etc. This SRS does not specify the requirements for any of the hardware and software platforms in the System, although within this document assumptions are made as to the capabilities, including software, networking, availability, storage and performance.

## Definitions, Acronyms, and Abbreviations

|  |  |
| --- | --- |
| API | Application Program Interface |
| BOINC | Berkeley Open Infrastructure for Network Computing |
| GUI | Graphical User Interface |
| P2P | Peer-to-Peer |
| RC | Rivest Cipher algorithm series |

Table 1.3.1: Definitions, acronyms, and abbreviations

## Overview

In the following, this document discusses the overall description of the system, which includes interface properties, dependencies and functions of the product. Furthermore, it contains functional and non-functional requirements of the system. Description of limitations of the product, addressing the constraints and assumptions can be found. Use case diagram and object class diagram is also included for a better picture during the implementation of the system. These provide necessary support and allows the project team to make sure that goals and functionalities of the system are being met.

Chapter 1 of this document briefly describes the purpose for implementing such a system. The purpose of writing this document and the intended audience who refer to this document. And determines the scope of the system. It gives a description of the purpose by including relevant benefits and goals.

Chapter 2 outlines the product perspective. Compares the available features in other similar and popular products, and how they operate. Functions of the application are briefly described with use cases. Design and implementation constraints describes the issues that limit the options available to the developers. All other assumptions made in implementing the system and its dependencies are also discussed.

Chapter 3 gives the specific requirements in developing the framework and the user agent. This includes the various types of external interfaces, performance requirements, the design constraints and software system attributes. A more detailed use case specifies how each function in the system works and what kind of information it deals with. Factors required to guarantee the reliability, availability, security and maintainability is explained under software system attributes.

# Overall Descriptions

This section briefly explains the background information about specific requirements of the framework and the user agent and the underlying technologies. Not every requirement of this system is discussed in depth. Instead, it describes the features that affect the final product.

With the main goal of bringing together the world’s computers together to make large scale, distributed, high-performance computers at will, this system provides an application with three major components.

1. Peer-to-peer, server-less network with support for arbitrary peers and network management

The network layer is the basis for the computing framework. The workload allocation and execution modules of the system runs on top of the network layer. The peer-to-peer nature of the network ensures that there can be no central server for management and coordination of the network. The responsibility of network management is more or less equally split between all the peers in the network. The management of the network entails network bootstrapping, peer identification, peer discovery, peer joining and peer leaving.

1. Workload allocator module that breaks down a task into smaller independent units of work

In order to take advantage of the processing power of several computers to simultaneously solve a problem, the problem must be broken down into smaller units. The divide and conquer strategy can be employed to solve such a problem in parallel. The workload allocator module is responsible for breaking down the task into smaller units and distributing it among the peers. This module is a pluggable unit and the framework has an interface to accept the pluggable workload allocator unit. Therefore, for each and every kind of problem, a unique workload allocator unit can be developed.

1. Workload executor module that receives, executes the work and submits the results

The module receives workload from nodes. The framework decides whether to accept or reject new workload depending on user preferences and CPU load. Ideally, this module utilizes the idle processing power of the CPU. Once the work has been delivered to the node, it will be completed in its own time and the results will be sent back to the originator of the workload. This is also a pluggable module that the framework can accept using an interface. Similar to the workload allocator module, the executor module can be developed for each kind of problem.

1. Fault tolerance

The system is fault tolerant up to a certain extent. The validity of the calculations is of utmost importance and cannot be ignored. Therefore, the system uses redundant calculations to validate the results. Another aspect of fault tolerance applies when the executing peers stop do not complete work units. The incomplete work unit must be re-allocated.

## Product perspective

Computational time sharing is now becoming a trending topic in the computer world. There has been several attempts undertaken in order to harness the idle processing power of computers. Yet, there is only very few systems that exist.

BOINC is an open-source middleware system that supports volunteer and grid computing. It was originally developed to manage the SETI@home project. And is now the most widely used volunteer computing system. BOINC distributes a task among the peers to be processed. To do this, a workload is broken down into smaller modules and then allocating them to the relevant peers. The results from the peers are then integrated to produce the final result. This works on a centralized model, where the peers are the clients. BOINC uses a credit management system helps to ensure that users are returning results which are both scientifically and statistically accurate [3].

Distributed.net was the first system to achieve computational time sharing. Distributed.net was found as an effort to break the RC5-56 portion of the RSA Secret-Key Challenge, which would take very long periods of time on a single computer [4].

Folding@home also works similar to BOINC. It focusses on analyzing protein data from biological research centers. This is in an attempt to understand protein folding, misfolding, and related diseases, with a minor emphasis in protein structure prediction.

These existing systems are based on pure volunteer computing on the client perspective. This is because the above mentioned systems are centralized. The client does not receive any reward for providing computation time. This might be a lack of motivation for the participation in volunteer computing. The proposed system not only provides the advantage to scientific or business parties, but will also provide the capability for an average user to perform their own tasks. This is achieved by bringing in a decentralized mechanism which is often known as Peer-to-Peer computing.

As the number of peers increase, comes the consideration on scalability, reliability and fault tolerance. Such issues are identified and categorized into four components as discussed in section 2.

Table 2.1.1 shows some of the features in the existing products and the proposed system.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Features** | **Distributed.net** | **Bitcoin** | **Folding@home** | **BOINC** | **Our Solution** |
| **Decentralized** |  | **✓** |  |  | **✓** |
| **Computational Time Sharing** | **✓** | **✓** | **✓** | **✓** | **✓** |
| **Parallel Processing** | **✓** |  | **✓** | **✓** | **✓** |
| **Authentication** |  | **✓** |  | **✓** |  |
| **Reward System** |  | **✓** |  |  | **✓** |
| **Credit System** |  |  |  | **✓** | **✓** |
| **Fault Tolerance** | **✓** | **✓** |  | **✓** | **✓** |
| **Result Validation** | **✓** |  |  | **✓** | **✓** |
|  |  |  |  |  |  |

Table 2.1.1: Feature comparison

The proposed system will be decentralized. This will allow users not only to provide their computation time, but also to request for computation time from others to complete one of their own workload. A reward system will provide the users with points based on several factors, such as amount of CPU time sharing volunteered, frequency of accepting projects, etc. Both the P2P implementation and the reward system will motivate users to participate in P2P computing.

### System interfaces

1. CentOS 6.5

2. Python interpreter 2

### User interfaces

Cancel

Browse

Add Project

File

View

Option

Tool

Help

Current Projects

Submit

Cancel

Public Projects

My Projects

Figure 2.1.1: Main interface for P2P system

Figure 2.1.2: Status interface for P2P system

Messages

File

View

Option

Tool

Help

Statistics

Task

Project

Project Manager

### Hardware interfaces

There are no special hardware interfaces. General purpose computers with network connectivity is the basic requirement.

### Software interfaces

1. Twisted - Network programming interface for Python
2. Kademlia - DHT API for Python
3. PyBOINC - Wrapper and a set of predefined libraries that lets you package a Python program as a BOINC application

### Communication interfaces

The following interfaces are used to connect to the internet:

Ethernet adapter   
Wireless network adapter

### Memory constraints

The framework is expected to use around 50 MB of main memory and 100 MB of secondary storage.

The pluggable allocator and executor modules are expected to use up to a maximum of 10 MB of main memory.

### Operations

There are two modes of operation:

1. Workload allocator mode

The application creates tasks, breaks it down into smaller tasks and submits them to other peers to be executed.

1. Workload executor mode

The application accepts tasks from other peers, executes and sends the results back to the allocator

The application is meant to be run without user interaction in the long term. User interaction is needed in the workload allocator mode to select a specific task to be executed and user interaction is needed in the workload executor mode to set the preferences.

There is generally no data processing functions in the framework. However, data processing functions may exist in applications built to run tasks that involve data processing.

### Site adaptation requirements

The framework is built for Windows and Linux operating systems. It is specifically built for and tested on CentOS version 6 and above and can be adapted to other operating systems with little effort.

These packages are required:

1. Python 2
2. Twisted 15

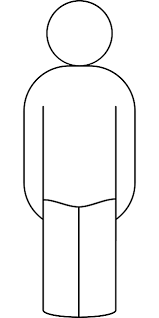
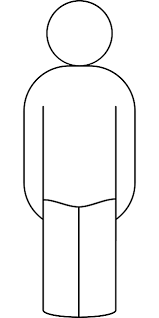
The framework is meant to be used by the general population for generic computational problems. The workload executor mode will run mostly on household sites on computers of volunteers. The Workload allocator mode may be run on mission critical environments where a lot of computational power is needed. This is not to mean that there will not be any workload allocators in the general population.

The application will be unilingual and support the English language, although the framework itself is independent of the language of the application and anyone can create the user interfaces of the application in any language they desire.

## Product functions

### Use case diagram

Figure 2.2.1: Use case diagram for P2P system



Allocator

Executor

Start network

Execute task

Find peers in the network

<<includes>>

Assign task

Abort task

Connect to network

Abort execution

View progress

Pause execution

View results

Set computing preference

### Use case scenarios

2.2.2.1 Peer-to-peer, server-less network with support for arbitrary peers and network management

|  |  |
| --- | --- |
| Use case 1 | Start network |
| Preconditions | User is connected to a LAN or a WAN |
| Successful End Condition | Connected to network |
| Actors | Allocator / Executor |
| Main Success Scenario | 1. Connect to LAN or WAN 2. Set network properties 3. Start and join the network as the bootstrap peer 4. Wait for other peers to join the network |
| Extensions | 4a. No peers send a join request for a considerable amount of time  4b. Display option to abort the network or wait longer  4c. If user selects the option to wait longer, attempt to receive join requests from peers for a longer time period before displaying the message again. |

Table 2.2.1: Functional requirement for start network

|  |  |
| --- | --- |
| Use case 2 | Connect to network |
| Preconditions | User knows at least one peer in an existing network |
| Successful End Condition | Network started |
| Actors | Allocator / Executor |
| Main Success Scenario | 1. Download (find) a partial list of peers already in the network  2. Connect to network |
| Extensions | 1a. The existing peers do not respond to join request  1b. Display option to select a different network or stop trying to connect to the network |

Table 2.2.2: Functional requirement for connect network

|  |  |
| --- | --- |
| Use case 3 | Add task |
| Preconditions | User is connected to a network  User has the allocator module of the task type |
| Successful End Condition | Task distribution begins |
| Actors | Allocator |
| Main Success Scenario | 1. Select the task type  2. Add the task data  3. Begin the task |
| Extensions |  |

Table 2.2.3: Functional requirement for add task

|  |  |
| --- | --- |
| Use case 4 | Abort task |
| Preconditions | There is at least one task submitted for execution |
| Successful End Condition | Task is aborted |
| Actors | Allocator |
| Main Success Scenario | 1. User selects the task to abort  2. Executors are notified  3. Task is removed from the network |

Table 2.2.4: Functional requirement for abort task

|  |  |
| --- | --- |
| Use case 5 | Execute task |
| Preconditions | Some work units of the task has been received from the network |
| Successful End Condition | Task execution begins |
| Actors | Executor |
| Main Success Scenario | 1. User selects the task to abort  2. Executors are notified  3. Task is removed from the network |

Table 2.2.5: Functional requirement for execute task

|  |  |
| --- | --- |
| Use case 6 | Abort Execution |
| Preconditions | There is at least one work unit executing at the moment |
| Successful End Condition | Task execution begins |
| Actors | Executor |
| Main Success Scenario | 1. User selects the work units to abort  2. Allocators are notified  3. Work unit reallocation begins |

Table 2.2.6: Functional requirement for abort execution

|  |  |
| --- | --- |
| Use case 7 | Pause Execution |
| Preconditions | There is at least one work unit executing at the moment |
| Successful End Condition | Task execution pauses |
| Actors | Executor |
| Main Success Scenario | 1. User selects the work units to pause  2. Task is paused until resumed |

Table 2.2.7: Functional requirement for pause execution

|  |  |
| --- | --- |
| Use case 8 | Set Computing Preferences |
| Preconditions | - |
| Successful End Condition | Computing preferences are set |
| Actors | Executor |
| Main Success Scenario | 1. User opens the computing preferences window  2. User sets the options  3. User confirms selection |

Table 2.2.8: Functional requirement for set computing preference

|  |  |
| --- | --- |
| Use case 9 | Receive Tasks |
| Preconditions | User is connected to the network |
| Successful End Condition | Task execution pauses |
| Actors | Executor |
| Main Success Scenario | 1. User selects the work units to pause  2. Task is paused until resumed |

Table 2.2.9: Functional requirement for receive task

|  |  |
| --- | --- |
| Use case 10 | View Results |
| Preconditions | The task has been completed |
| Successful End Condition | Task execution pauses |
| Actors | Allocator |
| Main Success Scenario | 1. User selects the work units to pause  2. Task is paused until resumed |

Table 2.2.10: Functional requirement for view results

## User characteristics

The framework has only one user mode and the application running on the framework has access to all the functionality of the framework. The application also has only one user mode since the one user should be able to access all of the functionality provided by the application.

The user can take the role of the allocator, the executor or both. These user roles can be thought of as modes of operation of the framework.

## Constraints

### Hardware limitations

The system is not practical to execute in the mobile platform since mobile devices have extremely limited battery life and running this application in mobile devices will surely deteriorate the battery level of the device in a very short time. This would defeat the purpose of the device since it will have to be plugged to a power source almost constantly to run the application successfully.

The system should ideally utilize the GPUs as well as CPUs. However due to limited scope, the implementation will not include GPU utilization.

### Time constraints

Time is a major constraint in the implementation of the system. A compromise between features and the time given to complete the system has been reached.

### Criticality of the application

If the framework is used to run mission critical application, then it is not advisable to run the system in volunteer computers, but use dedicated machines instead. The framework could be thought of as a quick way to bring together a number of machines for parallel computing in such a scenario.

2.4.4 Safety and security considerations

There is a concern that the system could be used to run illegal applications. A volunteer on the network could be unknowingly contributing their CPU time for illicit activities.

### Parallel operation

The framework acts as one giant grid of parallel processors allocating the workload among themselves. The limitation to this is that only the type of tasks that have parallel nature can be broken down into independent sub tasks and distributed throughout the grid.

## Assumptions and dependencies

It is assumed that when a task is allocated to volunteer computers on the network a deadline to complete the work is not observed.

## Apportioning of requirements

The primary specification of the Peer-to-Peer Distributed Computing Framework is described in chapter 1 and chapter 2 of this document. Chapter 3 referred to the detailed requirements specifications. The two levels of requirements are intended to be consistent. Inconsistencies are to be logged as defects. In the event that a requirement is stated within both primary and functional specifications, the application will be built from functional specification since it is more detailed.

Desirable requirements are the functions to be implemented in this release if possible, but aren’t dedicated as essential to be implemented by the developers. This can occur due to the time constraints and it is anticipated that they will be part of future release. Future versions of the system could incorporate a smarter interaction.

Utilizing the GPU for general purpose processing can speed up the computation by a considerable amount. The CUDA technology if integrated with the system can help realize this target, however time is a limiting factor and the requirement is define as merely a desired requirement.

# Specific requirements

## External interface requirements

There is a main user interface for the user agent software. The framework itself has no user interfaces. The user interface has two tabs. One for job allocation and the other for job execution. The options windows can be accessed from the main interface. The decision has been made not to develop two different interfaces for the allocator and executor since the job allocation and execution can happen simultaneously. The user interface will be limited to the types of controls that can be developed by PyQt GUI binding.

As for the hardware interfaces, it is handled by the underlying operating system, in this case CentOS. No low level hardware dependent modules will be written by the development team for this system.

## Functions

The system has two main parts. The framework and the application that runs on top of the framework. Therefore, it can be divided into three main modules.

1. The peer-to-peer network
2. The workload allocator
3. The workload executor

The three modules have the following functions:

1. The peer-to-peer network
2. Bootstrap the network
3. Join an existing network
4. Leave the network
5. Handle non-responsive peers
6. The workload allocator
7. Install new allocator modules
8. Breakdown parallel task
9. Distribute the workload
10. Aggregate the results
11. The workload executor
12. Install new executor modules
13. Receive work units
14. Execute the work
15. Submit the results

## Performance requirements

An internet connection with an average downlink speed of 400 kbps will allow an uninterrupted service.

The number of users may increase gradually and the system should be capable of holding new peer information. Equally, the system shall be able to respond to many peers simultaneously. Response time for a request should be less. The capacity refers to the maximum user load on the P2P network. It also depends on the network performance. In the case of increasing number of users or processing of data a real time system has to be considered to prevent work overload.

The recommended requirements for the personal computer are: a minimum of 1GB RAM, 800MHz processor and Centos 6 or above. Greater parameters will increase performance as a task may require more memory and CPU time.

## Logical Database Requirements

There are no database requirements.

## Design constraints

The interfaces of the application should be designed in a way that user experiences are smooth. System uses standard principles of designing and maintains consistency of the user interfaces by using identical pattern. Design of the system allows future modification to the system.

Consideration on the performance of the system to handle concurrency and allow uninterrupted service among the large number of users.

A system crash or a power failure may occur. Data corruption prevention and recovery methods is of great significance. Therefore, fault tolerance should be considered to implement a reliable system.

The coding conventions of Python is followed when developing the system.

The design standards provided by the UML specification is followed when creating the system diagrams.

## Software System Attributes

### Reliability

Reliability is the probability of failure-free software operation for a specified period of time in a specified environment [2].

The system should have the ability to perform its normal operations with minimum failures over a specified time. Failures may occur due to several reasons.

The application should be available to be reliable. The back end of the application should be online and returns results when requests are made by the peers.

### Availability

The system has to be available for users whenever required. The system should be especially safeguarded from DOS attacks. The biggest factor that affects availability is the number executors in the network. If the demand for CPU time is higher than the supply of CPU time then availability is highly degraded. The point system should help in keeping the supply and demand in balance.

### Security

There are security concerns in providing random users with access to the CPU of your computer. The system allows a program from an internet source to be executed on your computer and uses that program to operate on the data provided by random users in the network. The biggest security issue lies in the trustworthiness of the provider of the executor module. To establish trust, a repository of executor and allocator modules should be created, maintained and reviewed by an open source community. Users are strongly advised against installing modules developed by unknown publishers.

Another alternative to address this issue is to run the system in a sandbox within your computer. This could be implemented in the future versions of the product.

### Maintainability

The development process follows a modular design so that maintainability does not become a burden. The individual functions in the program are very compact and loosely coupled. Extensive code commenting is done in order to make it easy for anyone to read and understand the coding. Documentation provides the rest of the guidelines about the system.

The pluggable allocator and executor modules are developed according to the problem to be solved. A public interface is exposed by the framework to accept the pluggable units. These units must conform to the specification provided by the interface. These modules are expected to change frequently.

### Portability

The system is developed in Python. Python is a platform independent, interpreted language and therefore portability between operating systems does not require the system to be rebuilt. The relevant libraries should be replaced when the operating system is changed. Currently there are no plans to port the system into the mobile platform. In the future, the Android, iOS and Windows mobile platforms could be eligible for portability.

# Supporting information

## References

[1] Wikipedia: Software Requirements Specification [Online]. http://en.wikipedia.org/wiki/

[2] Software Reliability, Carnegie Mellon University, [HTML].Available: http://www.ece.cmu.edu/ koopman/des s99/sw reliability/ [Accessed: 24th March 2013

[3] BOINC: Berkely Open Infrastructure for Network Computing [Online]. http://boinc.berkeley.edu/

[4] Wired: Macho Computing at Root of RSA Contest Flap [Online]. http://archive.wired.com/science/discoveries/news/1997/03/2350