# A Punnet of Berries Documentation

TeamPi

September 01, 2013

# CONTENTS

# CHAPTER ONE

# INTRODUCTION

## 1.1 Document Intent

This design document serves as a reference, outlining all information pertaining to the Punnet of Berries project. It is intended to provide an overview of design considerations and technical specifications regarding the final implementation. Descriptions of system design will be shown to allow for the understanding of what is to be constructed and how it will be built. This document should not be considered an installation or configuration guide; its primary goal is to detail the design, development, and functionality of the cluster.

## 1.2 Document Overview

Outlined in this document is the framework on which the Punnet and its accompanying software were developed. Each iteration in the development process is illustrated in detail in each section, covering system design, architecture, and implementation. Within this are accompanying graphical documentation and requirement specifications to aid in the understanding of system architecture.

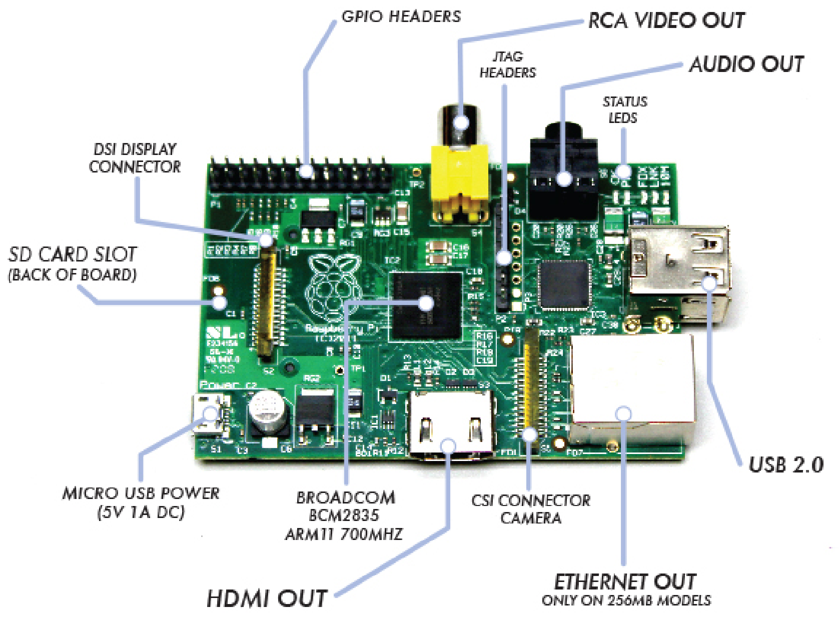
## 1.3 Scope

The breadth of the Punnet of Berries project has been intentionally delimited in order to make it feasible in a relatively short period of time. Given the allocated time span in which development has been defined, it is important to delineate any milestones that need to be achieved in the early alpha phase of design. This will be covered in *Goals and Objectives*. This design document is focused on core architectural design and critical parts of the system. As the project incorporates many pre-existing systems in conjunction with custom software, a large focus will be attributed to the interaction of these components.

## 1.4 The Punnet of Berries

A Beowulf cluster is a collection of (typically) inexpensive computers, networked together by a local area network, usually an Ethernet, and running some parallel processing software. This concept puts the power of supercomputing into the hands of schools and small research groups.

Developed to educate and inspire the next generation of programmers, the Raspberry Pi is a single-board, credit card sized device. It is capable of running Linux, and other light-weight operating systems, with its ARM processors.


GPIO HEADERS
RCA VIDEO OUT
JTAG HEADERS
AUDIO OUT
STATUS LEDS
DSI DISPLAY CONNECTOR
SD CARD SLOT (BACK OF BOARD)
USB 2.0
MICRO USB POWER (5V 1A DC)
BROADCOM BCM2835 ARM11 700MHZ
CSI CONNECTOR CAMERA
HDMI OUT
ETHERNET OUT ONLY ON 256MB MODELS

The Raspberry Pi is a powerful, yet, low-cost "mini-computer". The **Punnet of Berries** project aims to create a Raspberry Pi *Beowulf cluster*.

CHAPTER

TWO

# SYSTEM OVERVIEW

In order for a compute cluster to function properly and fairly, its resources must be managed and monitored by a batch system scheduler. Batch systems enforce limits on the number of jobs running at one time and the resources available to them. Compute cluster users request resources by submitting jobs to the batch system, which then determines the scheduling that best utilises the resources available.

The Punnet of Berries compute cluster will be managed by a custom batch system, the *Berry Batch*.

## 2.1 Specifications

### 2.1.1 Hardware

| **Category** | **Qty** | **Description** | **Supplier** | **Unit Cost** | **Total Cost** |
|---|---|---|---|---|---|
| Computer | 6 | Raspberry Pi Model B 512MB | element14 | $41.80 | $250.80 |
| Storage | 6 | Assorted brands 8GB SD card | MULTIPLE | $12.00 | $72.00 |
| Enclosure/Case | 6 | Raspberry Pi Case (Clear) | element14 | $9.79 | $58.74 |
| Power Supply | 6 | Power Supply USB 5V/1.2A | element14 | $22.00 | $132.00 |
| Ethernet Cables | 6 | 1.5m Cat-5e Patch Cable | MULTIPLE | $3.00 | $18.00 |
| Switch | 1 | 8-Port 10/100 Mbps Ethernet Switch | UNKNOWN | $34.00 | $34.00 |

### 2.1.2 Software

The operating system of choice for the Punnet of Berries super computer is a ported version of Arch Linux for ARM processors. This linux variant is very lightweight and only requires approximately 130MB of disk space, packing in only the bare essentials required for a functional operating system.

In addition to the operating system, the following applications will be installed:

- GCC
- Open MPI
- Python 2
- Python 3

CHAPTER

THREE

# DESIGN CONSIDERATIONS

## 3.1 Goals and Objectives

The Punnet of Berries project aims to create a financially viable Beowulf cluster, which is energy efficient and demonstrates the advantages of parallel computing. This will be achieved by utilising inexpensive hardware and open source technologies. In order to function properly, the system's resources will need to be managed and allocated by a batch scheduler. The Punnet of Berries will be accompanied by a custom batch system, the *Berry Batch*.

**Priority Objectives**

- Establish open source tools that will be utilized.
- Generate system topology and determine appropriate interfaces.
- Develop system and application design.
- Determine most appropriate software development methodology.

**Functional Requirements**

- The user must be able to specify the number of cores to undertake any given job.
- The Batch master must be able to support as many slave nodes as possible.

*These requirements will be expanded on throughout development.*

**Non-functional Requirements**

- Scalability
- Testability
- Reliability
- Documentation
- Open Source

## 3.2 Constraints

### 3.2.1 Hardware: Raspberry Pi

The Raspberry Pi is a comparatively low end single-board computer. As such, the design process must accommodate for these hardware specifications:

- SoC (System on a Chip) Broadcom BCM2835

- 700 MHz ARM core
- VideoCore IV GPU
- 512 MB of SDRAM.

Aside from an SD card, the Raspberry Pi does not feature any sort of non-volatile storage. Efficiency is a concern when constrained to a low capacity memory card. There is also an interface bottleneck, as both the USB and Ethernet share the same bandwidth.

An additional concern is of power distribution in regards to scalability; this would likely be resolved with the utilization of custom power supplies, but nonetheless it is something to consider.

### 3.2.2 Software: Parallelism and Concurrency

An area of concern is the difficulty of developing software which executes computations in a parallel manner, by distributing them across multiple cores.

### 3.2.3 Software: Architectural Considerations and Code Compilation

The Raspberry Pi hosts a single-core ARM processor, operating at 700MHz (with overclocking available). Due to this, any software developed will be in languages that are high-level, architecturally dependent and need to be recompiled on the Pi itself.

## 3.3 Methodology

### 3.3.1 Collaboration Tools

The collaboration tools utilised for the development of the Punnet of Berries project reflect the Open Source nature of the project.

**Version Control** All code and documentation relating to the Punnet of Berries project will be hosted in repositories on GitHub under the *rmit-teamPi* organisation.

**Project Management** The project and bug-tracking will be managed using Redmine, an open source web-based project management tool. The Redmine account will be hoseted for free on HostedRedmine.

### 3.3.2 Development

Given the collaborative effort of the development process, it is prudent to define the best practices and design principles to abide by. All protocols must adhere to a certain standard to promote scalability and ease of use.

The design approach is one that highlights the employment of parallel processing, concurrency, and synchronization. Fundamental operating system concepts will be explored and integrated, exposing the Raspberry Pi as a viable platform for cluster computing as well as comprehension of underlying technologies.

Design Principles:

- **Scalability** A computer cluster is, by nature, designed to be scalable. This inherent design characteristic must be adhered to, avoiding any sort of strict node limitations.
- **Usability** As this project is largely for educational purposes, the cluster should be easily constructed with high cohesion. The final application interface should also be readily accessible.

- **Flexibility** The software should be customizable, allowing users to specialize; secondary choices should be made available to enhance the application.

CHAPTER

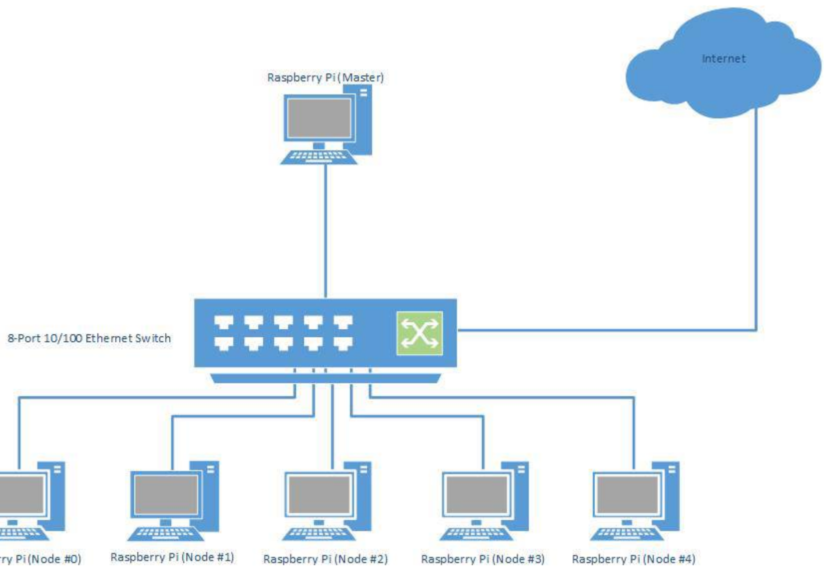
FOUR

# ARCHITECTURE

## 4.1 System Design

*A chain is only as strong as its weakest link.*

In a distributed model super computer, this statement cannot be more true. No matter how quickly each individual node can process its allocated tasks, if the communications link between the master system and each of the slave nodes becomes a bottleneck, then the entire system will suffer in performance.



In order to produce an efficient, flexible and reliable system, the selection of a networking topology which can fulfil those three specific characteristics is crucial. By selecting the star network topology as the basis for the Punnet of Berries super computer, the key requirements of efficiency, flexibility and reliability are met.

### 4.1.1 Efficiency

By design, star networks utilise a central hub such as a networking switch to handle to flow of packets in a network. This allows data packets to only travel to those nodes which they are intended for, reducing the overall traffic in the network.

### 4.1.2 Flexibility

By utilising a central hub for communications, a star network can easily be expanded to support additional nodes without affecting the entire network. Furthermore nodes can be removed at any time, if servicing is required.

### 4.1.3 Redundancy

By utilising a central hub for communications, the likelihood of total system failure is greatly reduced. Whilst failure of the central hub would mean that the entire system is brought down, failure of an individual node wouldn't have any affect on the overall system.

## 4.2 Program Design

The Berry Batch will consist of a centralised manager daemon and worker daemons.

Each node in the Punnet of Berries cluster will run a Berry Batch worker daemon. These workers exist to execute the jobs submitted to the system. The workers will:

1. Wait to be assigned a job.
2. Execute the job.
3. Returns the job's exit status.
4. Back to 1.

A job finished execution, with one of the following status:

1. **Successful** The job finished execution within the requested time frame and resources and without encountering any errors.
2. **Error** The job encountered an error during execution.
3. **Walltime Exceeded** The job exceeded the requested walltime and was terminated by Berry Batch.
4. **Resources Exceeded** The job required more system resources than requested and was terminated by Berry Batch.

The Berry Batch manager will monitor the system's resources and jobs. Users will interact directly with the Berry Batch manager to manage their job requests. A user is able to:

- Submit jobs.
- View all queued jobs.
- View the status of all of a particular user's jobs.
- View the status of a particular job.
- Cancel their own jobs.

### 4.2.1 Submitting Jobs

Users will write job scripts, which will be passed to the Berry Batch manager. A job script will consist of header and execution sections. The header section will specify the job details for queueing by Berry Batch. This could include a name for the job, the memory required, an estimated walltime and the number of nodes to run on. The execution section will contain the actual commands to be executed as a part of the job. While the job script's specific format is yet to be decided, an example is:

```
## Header Start ##
Name="JobHello"
Walltime=00:50:00
Nodes=2
Memory=256mb
## Header End ##

mpirun -np 2 helloWorld
```

Once a job has been submitted, the Berry Batch manager will determine which priority queue the job should wait in based on the resources and walltime requested in the header section. The queues could be:

- Short
- Medium
- Long
- Special (requires permission from the Punnet of Berries administration to run)

The *special* queue has first priority, followed by the *long* queue, and so on. If the resources are not available for any job in the *special* queue, the manager will look in the *long* queue for a suitable job, and so on. Each queue will operate in a *First in First Out* fashion.

### 4.2.2 Job Execution

Once a job has been submitted, it will move through the following states:

1. Queued
2. Running
3. Complete
4. Cancelled

When a job has completed, a Berry Batch job summary file will be written. This summary will contain details of the job's execution, such as the resources and walltime that were requested as well as what was actually used. Any standard output generated during execution will also be included in the summary file.

CHAPTER

# FIVE

# TESTING

## 5.1 Cluster Performance Tests

In order to test the performance and functionality of the 'Punnet of Berries' cluster, the system will be tested using a 'HPL (High-Performance Linpack Benchmark)'. This benchmark provides guesstimates of how many GFLOPS (**G** iga **FL** oating-point **O** perations **P** er **S** econd) of processing performance the Punnet of Berries cluster is able to achieve.

Once the cluster itself has been benchmarked, the 'HPL' will then be run on a single Raspberry Pi as well as seperate system containing a modern x86/x64 based processor. This will provide a meaningful comparison with regards to the performance gain a cluster of six Raspberry Pi's has over various other systems.

## 5.2 Batch System Tests

To prove that the Berry Batch is functioning correctly, it will undergo a range of tests. These include, but are not limited to, the following:

| **Test Cases** | **Expected Result** |
|---|---|
| Submit 1 short job. | Confirmation. Job put in short queue. |
| Submit 1 medium job. | Confirmation. Job put in medium queue. |
| Submit 1 long job. | Confirmation. Job put in long queue. |
| Submit 1 special job. | Job set to wait for permission. |
| One user submits a short job followed by one medium job. | First job put in short queue. Second job put in medium queue. |
| One user submit a short job. Second user submits a short job. | First user's job put in short queue. Second user's job put in short queue after the first. |
| One user submit a short job. Second user submits a medium job. | First user's job put in short queue. Second user's job put in medium queue. |
| One user submit a short job. Second user submits a medium job. Third user submits a medium job. | First user's job put in short queue. Second user's job put in medium queue. Third user's job put in long queue. |
| User views all current jobs. | List of jobs currently in the system. |
| User views a particular user's jobs. | List of user's jobs currently in the system. |
| User views a specific job. | Details of the given job. |
| First user views all current jobs. Second user views a specific jobs. | First user sees list of jobs currently in the system. Second user sees the details of the given job. |
| User cancels a queued job. | Confirmation of job cancellation. |
| User cancels a running job. | Confirmation of job cancellation. |
| User cancels a completed job. | Warning of invalid state. |
| User cancels an already cancelled job. | Warning of invalid state. |
| One user cancels another user's job. | Permission denied. |

CHAPTER

SIX

# ROLES AND RESPONSIBILITIES

The members of TeamPi are:

- **Alyssa Biasi:**

  **Primary Role:** Documentation, Testing and Management

  **Secondary Role(s):** Application Development, Operating System Configuration

- **Adrian Zielonka**

  **Primary Role:** Application Development

  **Secondary Role(s):** Operating System Configuration, Documentation

- **Zach Ryan**

  **Primary Role:** Hardware and Operating System Configuration

  **Secondary Role(s):** Application Development, Documentation

CHAPTER

SEVEN

# MILESTONE 01

## 7.1 Reflection

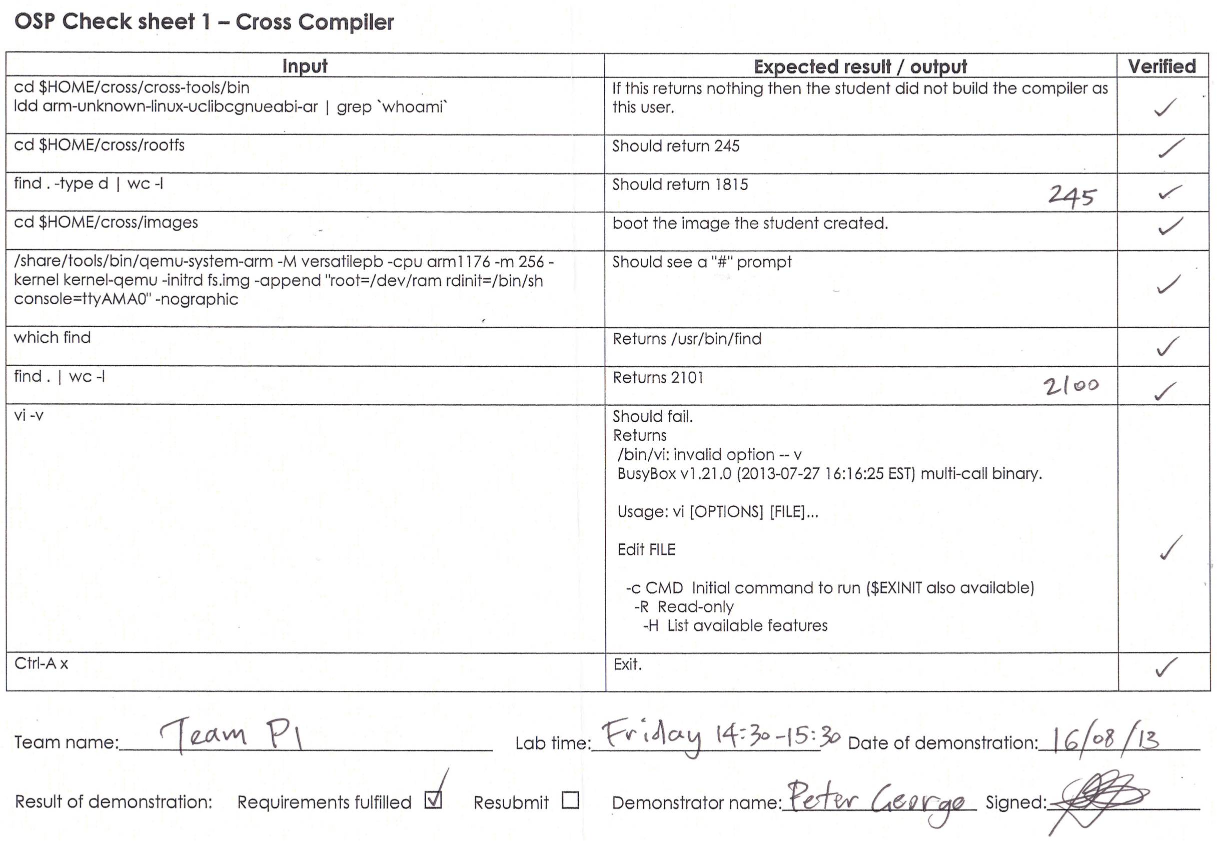
Milestone 1 involved following a cross-compiler recipe and becoming familiar working in a command line Unix environment.

The image was build by one TeamPi member, Alyssa Biasi. Alyssa already has considerable experience working on the command line. After spending some time getting to know the recommended tool, `tmux`, she followed the instructions given in the recipe and performed the sanity check. No problems were encountered during the build.

**uClibc vs. glibc:** Clibc was developed without consideration for other architectures and thus is a C library targeted directly for embedded Linux. glibc was intended for higher output performance and has additional features not required on a Raspberry Pi. The omitted features allow uClibc to function with a smaller amount of memory.

**Cros-Compliers:** A cross-compiler is critical in the creation of executables that are able to function on architectures that differ from the platform on which the code was developed.

## 7.2 Result



### OSP Check sheet 1 – Cross Compiler

| Input | Expected result / output | Verified |
|---|---|---|
| cd $HOME/cross/cross-tools/bin<br>ldd arm-unknown-linux-uclibcgnueabi-ar \| grep \`whoami\` | If this returns nothing then the student did not build the compiler as this user. | ✓ |
| cd $HOME/cross/rootfs | Should return 245 | ✓ |
| find . -type d \| wc -l | Should return 1815 245 | ✓ |
| cd $HOME/cross/images | boot the image the student created. | ✓ |
| /share/tools/bin/qemu-system-arm -M versatilepb -cpu arm1176 -m 256 -kernel kernel-qemu -initrd fs.img -append "root=/dev/ram rdinit=/bin/sh console=ttyAMA0" -nographic | Should see a "#" prompt | ✓ |
| which find | Returns /usr/bin/find | ✓ |
| find . \| wc -l | Returns 2101 2100 | ✓ |
| vi -v | Should fail.<br>Returns<br>/bin/vi: invalid option -- v<br>BusyBox v1.21.0 (2013-07-27 16:16:25 EST) multi-call binary.<br><br>Usage: vi [OPTIONS] [FILE]...<br><br>Edit FILE<br><br>-c CMD Initial command to run ($EXINIT also available)<br>-R Read-only<br>-H List available features | ✓ |
| Ctrl-A x | Exit. | ✓ |

Team name: Team P1 Lab time: Friday 14:30-15:30 Date of demonstration: 16/08/13

Result of demonstration: Requirements fulfilled [✓] Resubmit ☐ Demonstrator name: Peter George Signed: [illegible]