

Technical Overview

Neronet

*Toolbox for managing the training
neural networks*

CSE-C2610
Software Project
Aalto University

December 6, 2015

Outline

Overview

Components

Remarks

Technical
Overview

Neronet

Overview

Components

Remarks

Outline

Overview

Components

Remarks

Technical
Overview

Neronet

Overview

Components

Remarks

Neronet is a framework designed to facilitate the specification, submission, monitoring, control, analysis and management of many different computational experiments.

The Neronet family consists of three members:

1. **Neroman** – a user run tool that acts as the *Neronet frontend* and user interface to provide access to Neronet's features and functionality. It is the researchers workhorse.
2. **Neromum** – a **Nerokid** manager daemon that helps the family stay organized while the kids are playing in distant clusters, possibly in collaboration with a **Warden** that is predesignated to supervise and control the playing grounds.
3. **Nerokid** – a tiny creature that looks after your computational toy and reports to mum in case it breaks.

Introduction

In more technical terms:

- ▶ **Nerokid** is a program designed to be launched by a job scheduler or **Neromum** directly in a cluster node to monitor, analyse and control a single experiment job and its environment in tandem with a **Neromum**
- ▶ **Neromum** is a job scheduler, manager and communications agent designed to work by herself or with a standard cluster job scheduler such as SLURM, OGE or Jobman (a **Warden**)
- ▶ **Neroman** is a tool that acts as the *Neronet frontend* and user interface to provide Neronet's functionality by dispatching **Neromums** in specified clusters or nodes with a bunch of jobs to do. The **Neromums** in turn dispatch the jobs to the **Nerokids** and then continue as communication intermediaries.

Overview

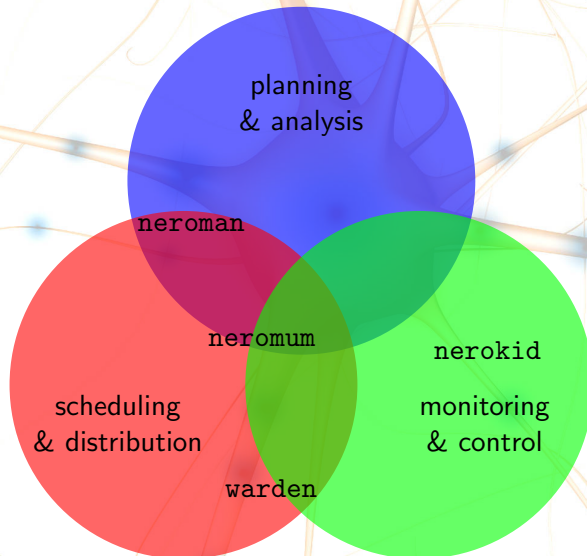
Components

Remarks

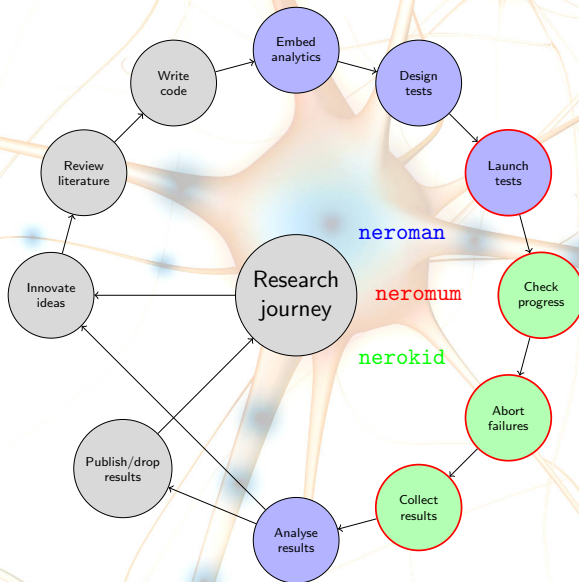
The following three slides provide a good introduction:

1. *Job management* – the associated problem domain spheres and how the system components are related to them
2. *Journey map* – the journey or work flow of a typical system user (a computational researcher) and the steps in which the Neronet tools are designed to be used
3. *Basic use case* – A sample basic system use case description

Job management



Journey map



- ▶ **User:** A computational researcher
- ▶ **Goal:** To test how well a new design works with several different configuration options and parameter values
- ▶ **Preconditions:** SLURM cluster and Neronet setup, code and analytics developed and test inputs setup in Neronet compatible manner
- ▶ **Basic flow:**
 1. Specify a batch of **Nerokids** in the *config* YAML with parameters, inputs and other configurations
 2. Dispatch the jobs to the SLURM setup with autogenerated sbatch scripts and arguments: `neroman --submit triton 124-186`
 3. Receive and check progress notifications from email
 4. Monitor the experiment to see near realtime updates of analytics variable updates: `neroman --submit triton 124-186`
 5. Receive final results data and updates. To a log data file.
 6. Analyse, reiterate and/or publish results
- ▶ **Post conditions:** Computational experiments have been conducted in a very straightforward, effective and researcher friendly fashion

Outline

Overview

Components

Remarks

Technical
Overview

Neronet

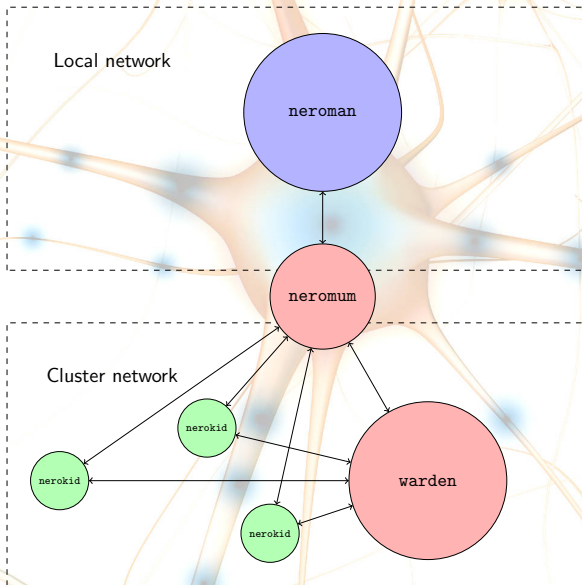
Overview

Components

Remarks

- ▶ All Nero components are lightweight Python programs run with just the researcher's privileges on any modern *nix
- ▶ SSH is used for communication between **Neroman** and **Neromums** (user's existing ssh keys, ssh configs and privileges are used)
- ▶ Sockets are used between **Neromums** and **Nerokids**
- ▶ **Neromum** communicates with any **Wardens** using their CLIs and/or APIs
- ▶ The system is ment to be easy to setup, lightweight and the usability good for several types of uses.

Communication



- ▶ A daemon administered and configured by the researcher herself
- ▶ Should be run on a system with two way SSH access to the cluster gateway or nodes where the **Neromums** are deployed
- ▶ Key functionality
 - ▶ Facilitate and standardize experiment specification
 - ▶ Batch submit experiment jobs to **Neromums**
 - ▶ Send email notifications with progress data
 - ▶ Facilitate monitoring and control of running jobs
 - ▶ Autocollect key job results into a researcher specifiable format (f. ex. Excel)
 - ▶ Facilitate experiment analysis and history management
 - ▶ Lightweight and extendable with custom functionality
 - ▶ Configurable via YAML files

- ▶ A daemon started and administered by a **Neroman**.
- ▶ Should be run on a system with two way SSH access to the **Neroman** and socket access to the nodes where **Nerokids** are deployed
- ▶ Collaborates with standard **Wardens**.¹
- ▶ Key functionality
 - ▶ Receive and execute commands from **Neroman**
 - ▶ Communicate with any **Wardens** by autogenerating job scripts (eg. sbatch) and their CLIs and/or APIs.
 - ▶ Collect information from **Nerokids**, process and cache them, send them to **Neroman** on request
 - ▶ Transmit commands from **Neroman** to the **Nerokids**

¹SLURM is currently used by Triton and CSC. OGE is still used worldwide. Jobman could be setup for the CS gpu cluster.

- ▶ A daemon started and administered by a **Neromum** directly or through a **Warden** on any modern *nix system (typically a cluster node) to start and monitor computational jobs
- ▶ Key functionality
 - ▶ Send information to **Neroman** via **Neromum** as configured
 - ▶ computing environment information
 - ▶ experiment job progress information read either through an API or by parsing output logs and data files (eg. CSV, JSON)
 - ▶ Interact with the **Warden** as specified (eg. autotermination based on poor experiment progress)
 - ▶ Lightweight and extendable with custom functionality
 - ▶ Configurable via YAML files and perhaps a Python API

Outline

Overview

Components

Remarks

Technical
Overview

Neronet

Overview

Components

Remarks

- ▶ A server (**Neroman**) per user approach is chosen because
 - ▶ easy minimalist setup (easy to try)
 - ▶ no need for special privileges
 - ▶ fully customizable by the user herself
 - ▶ low resources overhead per setup
 - ▶ total number of users relatively low as well
- ▶ SSH is used because
 - ▶ it is an existing standard among target system environments
 - ▶ user's existing SSH keys and configs provide an easy and effective way to provide secure networking
 - ▶ no need for network, port routing or privileges adjustments

- ▶ Python 3 is used because
 - ▶ it is already available in most modern *nix systems
 - ▶ it has good support for the known software requirements
 - ▶ it is already popular among computational researchers
 - ▶ many related research libraries use it (Scipy, Numpy, Theano, Lasange, Pylearn, Blocks)
- ▶ Sockets are used because
 - ▶ they provide fast and efficient realtime communication in tightly connected networks
 - ▶ they are lightweight for the cluster filesystem