The **Computer** Nonsense Guide

Or the influence of chaos on reason! 18.11.2017

Contents

Abstract

- In the name of nonsense, we refuse the Aristotelian formal logic.

The Computer Nonsense Guide describe both the languages and the operating system of Nonsense Worlds, Inc. Artificial Intelligence Laboratory completely documented by this guide.

The software environment and operating-system-like parts contain many things which are still in a state of flux. This work confines itself primarily to the stabler parts of the system, and does not address the window system, user interface and application programming interface at all.

The Computer Nonsense Guide is product of the efforts of many people too numerous to list here and of the unique environment of the Nonsense Worlds, Inc. Artificial Intelligence Laboratory.

Nonsense Worlds runs as an open-source multidisciplinary laboratory that conducts open research in distributed systems, artificial intelligence and high-performance computing.

Our primary base is within the HPC community. However, given the fact that mostly every mobile device is a supercomputer these days our users come from a wide variety of industries.

Our Mission: Driven by nonsense we focus on multi-dimensional research providing tools inside a simple tiling window environment for play, work and science.

Our Goal: Provide a distributed AI toolkit and workspace environment for machines of all ages.

We build our O/S on Debian plus all computer nonsense tools like additional semi-autonomous assistant, custom tiling window user interface and heavy focus on Lua, Erlang and Python.

Other native ZMTP implementations are libzmq (C++), NetMQ (C#), JeroMQ (Java), libzmtp (C).

You can easily talk to any other ZeroMQ socket or a more standard MPI stack on your network.

This integration enables zillion of BEAM processes to communicate with Torch, PyTorch, OpenResty, LÖVE, Tornado, Turbo or your favorite Singularity instance and focus around mutualism and balance inside distributed systems of C, Python, Lua and BEAM processes.

We make a stable workspace that anyone can use today, at nonsense these things work together into one unified environment with native support for Luerl and LFE.

This guide describes research done at the AI Laboratory of Nonsense Worlds, Inc.

Metamorphoses

My mind is bent to tell of bodies changed into new forms. Ye humans, for you yourselves have wrought the changes, breathe on these my undertakings, and bring down my song in unbroken strains from the world's very beginning even unto the present time.

Before the sea was, and the lands, and the sky that hangs over all, the face of nature showed alike in her whole round, which state have men called chaos: a rough, unordered mass of things, nothing at all save lifeless bulk and warring seeds of ill-matched elements heaped in one.

No sun as yet shone forth upon the world, nor did the waxing moon renew her slender horns; not yet did the earth hand poised by her own weight in the circumambient air, nor had the ocean stretched her arms along the far reaches of the lands. And, though there was both land and see and air, no one could tread that land, or swim that see; and the air was dark. No form of things remained the same; all objects were at odds, for within one body cold things strive with hot, and moist with dry, soft things with hard, things having weight with weightless things.

Nature composed this strife; for she rent asunder land from sky, and see from land, and separated the ethereal heavens from the dense atmosphere. When this she had released these elements and freed them from the blind heap of things, she set them each in its own place and bound them fast in harmony.

The fiery weightless element that forms heaven's vault leaped up and made place for itself upon the topmost height.

Next came the air in lightness and in place. The earth was heavier that these, and, drawing with it the grosser elements, sank to the bottom by its own weight. The streaming water took the last place of all, and held the solid land confined in its embrace.

When nature had thus arranged in order and resolved that chaotic mass, and reduced it, thus resolved, to cosmic parts, she first molded the earth into the form of a mighty ball so that it might be of like form on every side.

Then she bade the waters to spread abroad, to rise in waves beneath the rushing winds, and fling themselves around the shores of the encircled earth.

Springs, too, and huge, stagnant pools and lakes she made, and hemmed down-flowing rivers within their shelving banks, whose waters, each far remote from each, are partly swallowed by the earth itself, and partly flow down to the sea: and being thus received into the expanse of a freer flood, beat now on shores instead of banks.

Then did she bid plains to stretch out, valleys to sit down, woods to be clothed in leafage, and the rock-ribbed mountains to arise. And as the celestial vault is cut by two zones of the right and two of the left, and there is a fifth zone between hotter than these, so did the providence of nature mark off the enclosed mass with the same number of zones, and the same tracts were stamped upon the earth.

The central zone of these may not be dwelt in by reason of the heat; deep snow covers two, two she placed between and gave them temperate climate, mingling heat with cold.

The air hung over all, which is as much heavier than fire as the weight of water is lighter than the weight of earth. There did nature bid the mists and clouds to take their place, and thunder, that should shake the hearts of men, and winds which with the thunderbolts make chilling cold.

To these also the world's creator did not allot the air that they might hold it everywhere. Even as it is, they can scarce be prevented, though they control their blasts, each in his separate tract, from tearing the world to pieces. So fiercely do these brothers strive together.

Above these all she placed the liquid, weightless ether, which has naught of earthy dregs.

Scarce had she thus parted off all things within their determined bounds, when the stars, which had long been

lying hid crushed down beneath the darkness, began to gleam throughout the sky. And, that no region might be without its own forms of animate life, the stars and divine forms occupied the floor of heaven, the sea fell to the shining fishes for their home, earth received the beasts, and the mobile air the birds.

A living creature of finer stuff than these, more capable of lofty thought, one who could have dominion over all the rest, was lacking yet.

Then she was born.

So, then, the earth, which had but lately been a rough and formless thing, was changed and clothed itself with forms before unknown.

Getting started

First double check that you have Erlang, LuaJIT + luarocks and singularity >= 2.4 installed.

Then run this command:

luarocks install cube-cli

For help using cube-cli, including a list of commands, run:

$ cube-cli --help

Congratulations, you are jacked up and good to go!

Hello Luerl

- Luerl is an implementation of standard Lua 5.2 written in Erlang/OTP.

Lua is a powerful, efficient, lightweight, embeddable scripting language common in games, IoT devices, AI bots, machine learning and scientific computing research.

It supports procedural, object-oriented, functional, data-driven, reactive, organizational programming and data description.

Being an extension language, Lua has no notion of a "main" program: it works as a library embedded in a host. The host program can invoke functions to execute a piece of Lua code, can write and read Lua variables, and call Erlang functions by Lua code.

Through the use of Erlang functions, Luerl can be augmented to cope with a wide range of different domains, creating a customized language sharing a syntactical framework.

Luerl is a library, written in clean Erlang/OTP. For more information, click on the get started tutorial. You may also browse the examples and learn from the luerl_demo source code.

Don't forget to check the About and Community pages.

Hello LFE

- Good news, bad news and how to win big.

LFE is a proper Lisp based on the features and limitations of the BEAM Virtual Machine, attuned to vanilla Erlang and OTP it coexists seemlesly with the rest of the ecosystem.

It's a language out of a language out of Sweden that can be used to build web scale, asynchronous, non-blocking, sharded, event driven, message passing, NoSQL, reliable, highly available, high performance, real time, clusterable, bad ass, rock star, get the girls, get the boys, impress your mom, impress your cat, artificial intelligence applications.

You can blast it in the face with a shotgun and it keeps on coming.

Functional core

Erlang/OTP assumes functions with same name but different arities, each function has only a fixed number of arguments.

Pattern matching

BIG WIN™

Lisp-2+

LFE tried Lisp-1 but it didn't really work, Lisp-2 fits the VM better so LFE is Lisp-2, or rather Lisp-2+

The BIG Question

Apart from 42 the anwer to Life, the Universe, and Everything...

Q. Will LFE end the moaning and wining about Erlang syntax and how bad all the things?

A. NO!

Singularity Containers

Singularity enables SHA/OS users to have full control of their environment on whatever host they are on. This includes HPC, resource managers, file systems, GPUs and/or IoT devices, etc.

Containers are used to package entire scientific workflows, software and libraries, and datasets.

Singularity does this by enabling several keys:

- Encapsulation of the environment
- Containers are image based
- No user contextual changes or root escalation allowed
- No root owned daemon processes

Did you already invest in Docker? The Singularity software can import your Docker images without having Docker installed or being a superuser.

As the user, you are in control of the extent to which your container interacts with its host. There can be seamless integration, or little to no communication at all.

Container instances

Singularity has support for container instances, which means services!

Images instances can be started, stopped, and listed.

Along with instances comes **Network Namespace Isolation**

Documentation

Learn the build environment, including changing the cache and specifying credentials for Docker.

StarCraft: Brood War AI

StarCraft serve as an interesting domain for Artificial Intelligence (AI) research, since represent a well defined complex adversarial system which pose a number of interesting AI challenges in areas of planning, dealing with uncertainty, domain knowledge exploitation, task decomposition, spacial reasoning, and machine learning.

Unlike synchronous turn-based games like chess and go, StarCraft games are played in real-time, meaning the state will continue to progress even if the player takes no action, and so actions must be decided in fractions of a second, game frames can consist of issuing simultaneous actions to hundreds of units at any given time.

> M. Čertický, D. Churchill. The Current State of StarCraft AI Competitions and Bots. In Proceedings of the AIIDE 2017 Workshop on Artificial Intelligence for Strategy Games. 2017.

Installation

cube-cli install spqr

Meet the Marian AI bots

Go Read the Quick Start and our current list of Heuristics

Tsunami engine

luarocks install tsunami

Stream Mechanisms

When things wish to communicate, they must first establish communication. The stream mechanism provide a flexible way for processes to conduct an already-begun conversation with devices and with each other: an existing stream connection is named by a file descriptor, and the usual read, write, and I/O control request apply. Processing modules may be inserted dynamically into a stream connection, so network protocols, terminal processing, and device drivers are independent and separate cleanly.

However, these mechanisms, by themselves, do not provide a general way to create channels between them.

Simple extensions provide new ways of establishing communication. In our system, the traditional UNIX pipe is a cross-connected stream. A generalization of file-system mounting associates a stream with a named file. When the file is opened, operations on the file are operations on the stream. Open files may be passed from one process to another over a pipe.

These low-level mechanisms allow construction of flexible and general routines for connecting local and remote processes.

The work reported on streams describes convenient ways for programs to establish communication with unrelated processes, on the same or different machines.

Streams

A stream is a full-duplex connection between a process and a device or another process. It consists of several linearly connected processing modules, and is analogous to a Shell pipeline, except that data flows in both directions.

In essence, the stream I/O provides a framework for making file descriptors act in the standard way most programs already expect, while providing a richer underlying behavior, for handling network protocols, or processing the appropriate messages.

Trial and Error

Supervisors

On production, this usually means a fairly straight-forward combination of external process management, overload monitoring and proxying.

circusd

Circus is a Python program which can be used to monitor and control processes and sockets.

Circus can be driven via a command-line interface, a web interface or programmatically through its python API.

supervisord

Supervisor is a client/server system that allows its users to monitor and control a number of processes on UNIX-like operating systems.

monit

Monit is a small Open Source utility for managing and monitoring Unix systems. Monit conducts automatic maintenance and repair and can execute meaningful causal actions in error situations.

Coroutines

Processes

Organizations

An monkey, a building, an automobile, a drone: each is a concrete object and can be easily identified. One difficulty attending the study of organizations is that an organization is not as readily visible or describable.

Exactly what is an organization such as a business concern? It is a building? A collection of machinery? A legal document containing a statement of incorporation? It is hardly likely to be any of these by itself. Rather, to describe an organization requires the consideration of a number of properties it possesses, thus gradually making clear, or at least clearer, that it is.

The purposes of the organization, whether it is formal or informal, are accomplished by a collection of people whose efforts or to use a term to be employed throughout this work, behavior are so directed that they become coordinated and integrated in order to attain sub-goals and objectives.

The Lua API

Lua is an embeddable language implemented as a library that offers a clear API for applications inside a register-based virtual machine.

This ability to be used as a library to extend an application is what makes Lua an extension language.

At the same time, a program that uses Lua can register new functions in the Lua environment; such functions are implemented in Erlang (or another language) and can add facilities that cannot be written directly in Lua. This is what makes Lua an extensible language.

These two views of Lua (as extension language and as extensible language) correspond to two kinds of interaction between Erlang and Lua. In the first kind, Erlang has the control and Lua is the library. The Erlang code in this kind of interaction is what we call application code.

In the second kind, Lua has the control and Erlang is the library. Here, the Erlang code is called library code. Both application code and library code use the same API to communicate with Lua, the so called Luerl API.

Modules, Object Oriented programming and iterators need no extra features in the Lua API. They are all done with standard mechanisms for tables and first-class functions with lexical scope.

Exception handling and code load go the opposite way: primitives in the API are exported to Lua from the base system C, JIT, BEAM.

Lua implementations are based on the idea of closures, a closure represents the code of a function plus the environment where the function was defined.

Like with tables, Lua itself uses functions for several important constructs in the language.

The use of constructors based on functions helps to make the API simple and general.

There are no coroutines in Luerl it may seems counter intuitive coming from a more common Lua background.

In this ecosystem you always want to use processes instead, the BEAM Virtual Machine it's build for handling independent isolated processes that are very small and almost free at creation time and context switching. The main difference between processes and coroutines is that, in a multiprocessor machine a OTP release on the BEAM Virtual Machine runs several processes concurrently in parallel.

Coroutines, on the other hand, runs only one at the time on a single core and this running coroutine only suspends its execution when it explicitly requests to be suspended.

Pura LFE

An object is really a function that has no name and that gets its argument a message and then look at that message and decide what to do next.

In its pure essence message passing is a form of function calling or function calling is a form of message passing and objects are a form of function or functions are a form of objects.

Why Lisp?

The original idea was to produce a compiler, but in the 50's this was considered a major undertaking, and McCarthy and his team needed some experimenting in order to get good conventions for subroutine linking, stack handling and erasure.

They started by hand-compiling various functions into assembly language and writing subroutines to provide a LISP environment.

They decided on garbage collection in which storage is abandoned until the free storage list is exhausted, the storage accessible from program variables and the stack is marked, so the unmarked storage is made into a new free storage list.

At the time was also decided to use SAVE and UNSAVE routines that use a single contiguous public stack array to save the values of variables and subroutine return addresses in the implementation of recursive subroutines.

Another decision was to give up the prefix and tag parts of the message, this left us with a single type an 15 bit address, so that the language didn't require declarations.

These simplifications made Lisp into a way of describing computable functions much neater than the Turing machines or the general recursive definitions used in recursive function theory.

The fact that Turing machines constitute an awkward programming language doesn't much bother recursive function theorists, because they almost never have any reason to write particular recursive definitions since the theory concerns recursive functions in general.

Another way to show that Lisp was neater than Turing machines was to write a universal LISP function and show that it is briefer and more comprehensible than the description of a universal Turing Machine.

This refers to the Lisp function eval(e,a) which computes the value of a Lisp expression e, the second argument a being a list of assignments of values to variables, a is needed to make the recursion work.

LFE Flavours

Torch and PyTorch

Torch

Torch is a scientific computing framework with wide support for machine learning that puts GPUs first. It is easy to use and efficient, thanks to LuaLang and an underlying C/CUDA implementation.

A summary of core features:

- a powerful N-dimensional array
- linear algebra routines
- neural network, and energy-based models
- Fast and efficient GPU support
- Embeddable, with ports to iOS, Android and FPGA backends

Torch comes with a large ecosystem of community-driven packages in machine learning, computer vision, signal processing, parallel processing, image, video and audio among others, and builds on top of the Lua community.

PyTorch

PyTorch is a python package that provides two high-level features:

- Tensor computation (like numpy) with strong GPU acceleration
- Deep Neural Networks built on a tape-based autograd system

You can reuse your favorite python packages such as numpy, scipy and Cython to extend PyTorch when needed.

Usually one uses PyTorch either as:

- A replacement for numpy to use the power of GPUs.
- a deep learning research platform that provides maximum flexibility and speed

Tornado and Turbo

Tornado

Tornado is a Python web framework and asynchronous networking library, originally developed at FriendFeed. By using non-blocking network I/O, Tornado can scale to tens of thousands of open connections, making it ideal for applications that require a long-lived connection to each user.

Tornado can be roughly divided into four major components:

- A web framework (including RequestHandler which is subclassed to create web applications, and various supporting classes).
- Client- and server-side implementions of HTTP (HTTPServer and AsyncHTTPClient).
- An asynchronous networking library including the classes IOLoop and IOStream, which serve as the building blocks for the HTTP components and can also be used to implement other protocols.
- A coroutine library (tornado.gen) which allows asynchronous code to be written in a more straightforward way than chaining callbacks.

The Tornado web framework and HTTP server together offer a full-stack alternative to WSGI.

Turbo

Turbo.lua is a framework built for LuaJIT 2 to simplify the task of building fast and scalable network applications. It uses a event-driven, non-blocking, no thread design and minimal footprint to high-load applications while also providing excellent support for embedded uses.

It’s main features and design principles are:

- Simple and intuitive API (much like Tornado).
- Low-level operations is possible if the users wishes that.
- Implemented in straight Lua and LuaJIT FFI on Linux, so the user can study and modify inner workings without too much effort.
- Good documentation
- Event driven, asynchronous and threadless design
- Small footprint
- SSL support (requires OpenSSL)

Proxying

We include an example for proxing tornado and turbo behind nginx (openresty) as a load balancer.

Process Management

Traditionally, Tornado and Turbo apps are single-processes and require external management behind a process supervisor and nginx (openresty) for (proxying) load balance.

Why Erlang?

BEAM processes

Why ZeroMQ?

ZeroMQ (also known as ØMQ, 0MQ, or zmq) looks like an embeddable networking library but acts like a concurrency framework. It gives you sockets that carry atomic messages across various transports like inter-process and TCP.

Its asynchronous I/O model gives you scalable multicore applications, built as asynchronous message-processing tasks.

You can connect sockets N-to-N with patterns like fan-out, pub-sub, task distribution, request-reply and the new ZMTP 3.1 resource property.

Distributed Messaging thanks to the ZeroMQ Community.

- Carries messages across inproc, IPC, TCP, multicast.
- Smart patterns like pub-sub, push-pull, and request-reply.
- Resource Property (NEW in ZMTP 3.1!)
- High-speed asynchronous I/O engines, in a tiny library.
- Backed by a large and active open source community.

Go Read the guide and Learn the Basics

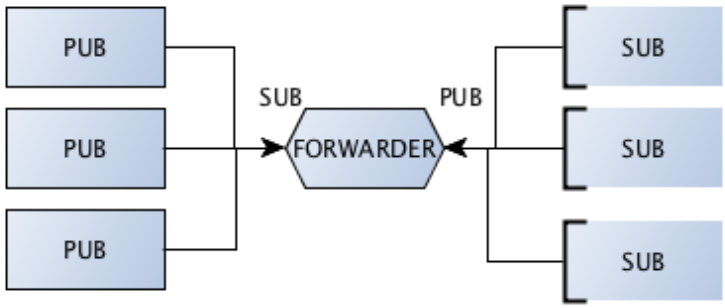
ZMQ devices

You must have noticed that you can bind a port to any of the ZeroMQ Socket types. In theory, most stable part of the network (server) will BIND on a specific port and have the more dynamic parts (client) CONNECT to that.

ZMQ provides certain basic proxy processes to build a more complex topology with basic device patterns this implementation focus on Forwarder and Streamer.

Forwarder

Forwarder device is like the pub-sub proxy server. It allows both publishers and subscribers to be moving parts and it self becomes the stable hub for interconnecting them.



This device collects messages from a set of publishers and forwards these to a set of subscribers.

Streamer

Streamer is a device for parallelized pipeline messaging. Acts as a broker that collects tasks from task feeders and supplies them to task workers.