Overview of Distributed Computing

signin.ritlug.com
(pray it works!)

RITLUG

Summary

- Data crunching (supercomputers)
- Rendering (render farms, Hollywood, Pixar, etc.)
- High availability (failover of things like web apps)

Data crucnching is like the simulations NASA supercomputers do.
https://www.youtube.com/watch?v=3RqF8m65r8g
Article on Blender's render farm & Big Buck Bunny - interesting read:
http://bbb3d.renderfarming.net/explore.html
The Blender render farm uses BURP, which is based on BOINC (next slide)
High availability is more like a mainframe sometimes.
https://www.youtube.com/watch?v=ximv-PwAKnc

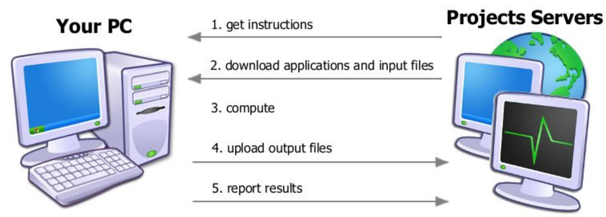
Data Crunching

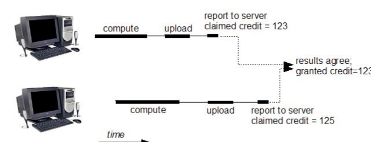
- What supercomputers for research usually do
- Can include simulations
- Examples:
  - BOINC
    - SETI@HOME (Search for ExtraTerrestrial Intelligence)
    - Einstein@home (LIGO, the gravitational waves thing)
    - LHC@home (CERN's supercollider)
  - Folding@Home (Protein folding, mostly medical research use)
  - You can participate in these examples too!

https://boinc.berkeley.edu/
http://folding.stanford.edu/

Data Crunching: Process

1. Computer joins the cluster/project
    a. Note: “Cluster” means “can be viewed as a single entity”
2. Computer gets sent a workload
3. Computer does the work
4. Computer sends back the result
5. The controlling server may (cross-)validate the result
6. Public projects frequently have a points system to reward contributors
    a. Points usually based around computational complexity

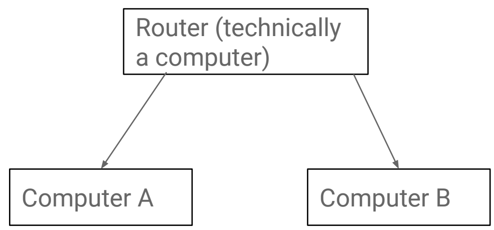




Graphics from BOINC’s page on how it works
Cross-validation needed b/c of bad actors & different hardware can get different results (for example CPUs & GPUs do math slightly differently)

Real Time Computing: Networks

This qualifies as distributed computing:

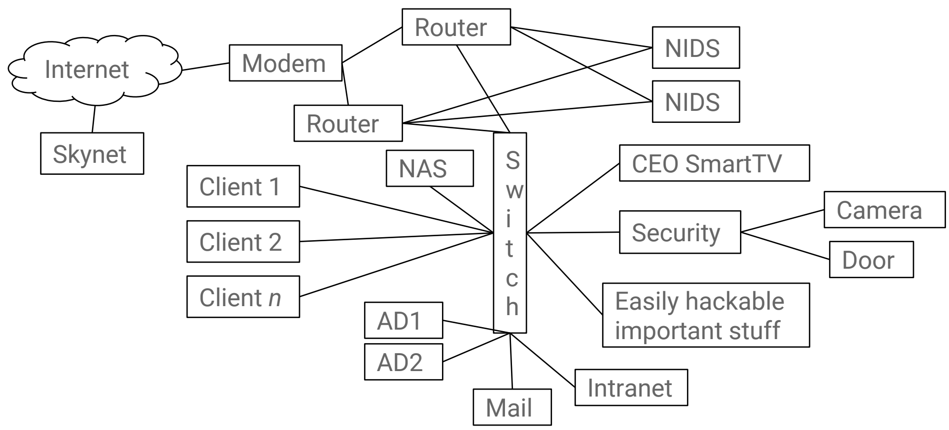


Multiple computers, networked (2 is still multiple)

...or this:

You don’t actually need a router/similar for networking, you can make it static

...but this is more interesting:



So, how did we land here and what did it take?

How? Process pt. 1: Manual

Take (a few) computer(s) and configure them by hand

Pros:

- Good for learning

Cons:

- Bad for any real use in most circumstances

Quick, move on to fun stuff

How? Process pt. 2a: Semi-automatic

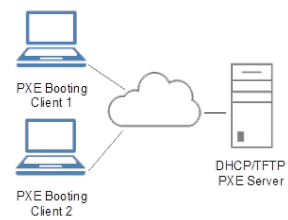
Set it up on one computer & clone ("imaging" - see also: Ghost(script), PXE)

Pros:

- Uniform, deployable

Cons:

- Can't manage w/o re-deploying



Quick, move on to fun stuff
PXE is over the network booting, which can be used to image the machine (since the disk is not being used by the OS, unlike with a normal boot)
https://en.wikipedia.org/wiki/Preboot_Execution_Environment

How? Process pt. 2b: Semi-automatic

Download & run a script (see also: setup.sh, Ansible, Chef, Puppet, Windows AD (GPO))

Pros:

- Uniform, easy to re-deploy

Cons:

- Still manual work

Quick, move on to fun stuff

We use Ansible in the TigerOS infrastructure

How? Process pt. 3: Automatic

Put management stuff in image

Pros:

- Manageable & redeployable
- Good for computer labs

Cons:

- Not most efficient use of resources

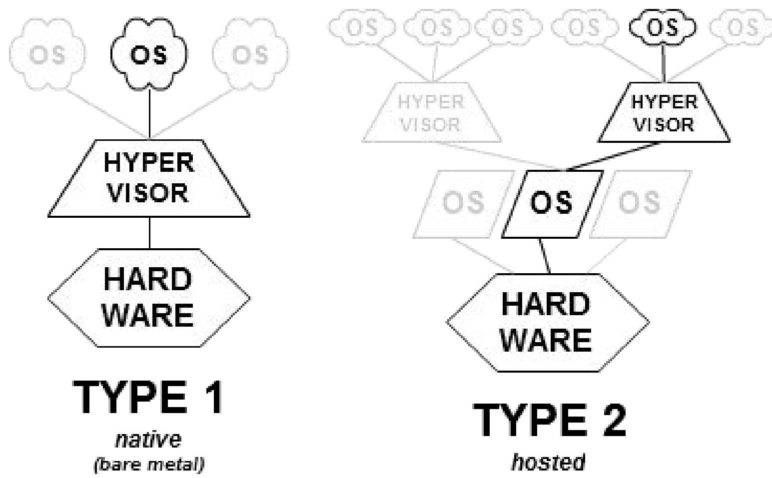
Quick, move on to fun stuff

RIT’s labs are set up like this.

How? Process pt. 4: Virtualization

Run virtual computers ("virtual machines" aka VMs) on the same hardware

4.0 Virtualization Types



As you can see, on a bare metal HV the HV is the Host OS (or heavily integrated at a very low level (inc. kernel)), whereas the hosted/software HV is an emulator (somewhat, there is hardware-level integration)

4.1: Virtualization Features

Thin Clients

Run the computer in a VM & use a lower powered computer to access it

Overprovisioning

Allocate more resources than the host computer has to account for under-usage of resources

Abstraction

Resources can be easily reassigned or moved w/ minimal (if any) effect on the things that depend on it.

Live moving of VMs

A VM can be transferred statefully from one host to another & just have the system look at the new place

Abstraction: (see also: OpenStack, pfSense virtual IPs (Open vSwitch has similar?)).
Thin clients allow a work-from-anywhere model.
Overprovisioning is to account for VMs not fully utilizing resources they are allocated (for example, allocate 150% of host's ability and get 100% usage by having VMs use 66% on average).

Virtualization: Summary

Pros:

- Lots, and we can use everything from the “How #3: Automatic” slide (not that that was as interesting)

Cons:

- Still could use resources more efficiently

How? Process pt. 5: Containers

Why run *x* copies of [insert OS here]?

Let’s not!

5.1: VMs vs. Containers

VMs

- Have to run multiple copies of [insert OS here] and the base libraries/packages it uses
- Have to maintain all of the systems
- Can over-reserve resources
  - It can block off all the RAM it's assigned even if it's not using it all for example

Containers

- Run Docker and abstract the basic OS-level resources the containers need to run
- Simply make a new version and swap out the old instance
  - Orchestrators can do this
  - Can be done since containers are ultimately stateless except when explicitly not
- Automatically assigns resources

Questions?

Attempt to demo RancherOS