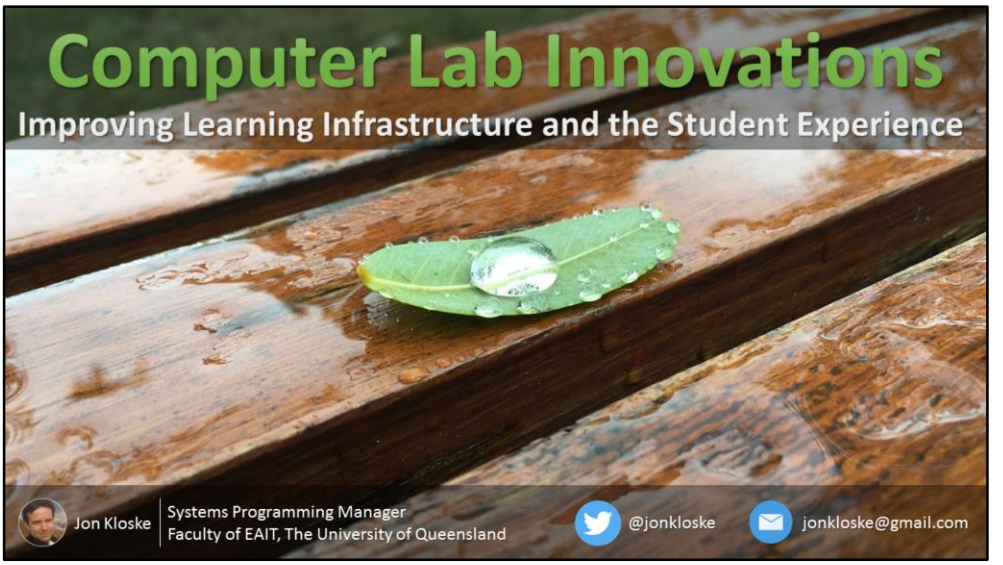


Computer Lab Innovations:

Improving Learning Infrastructure and the Student Experience

Abstract:

The Faculty of EAIT at UQ faces many problems providing flexible, highly available computer labs: computers are often the objects of study rather than merely support tools, large collections of specialist software with very restrictive licensing, and supporting BYOD users.

Over the past 15 years, we have developed an innovative set of tools, such *Pervade* and the *UQ Lab Status Monitor* (*UQLSM*) to help deploy and manage lab computers, and *Hypervade* and *Hadoop* to provide virtualised desktop and compute infrastructure.

The UQ Lab Status Monitor (UQLSM) provides realtime computer availability data and floorplans for students, enabling them to find a free lab computer more quickly and efficiently. We have received dozens of requests for the system by other universities and schools in Australia and in the USA.

Pervade and Hypervade are Linux based baremetal image deployment systems utilising bittorrent for scalability and in the case of Hypervade, KVM for virtualisation (allowing a second virtual copy of the physical lab machine to be made available through an RDP proxy for laptop and tablet users). This enables rapid deployment of our 150GB+ software image across 750+ PCs, and provides full access to the software for BYOD users without additional hardware costs.

Speaker Bio:

Jon Kloske is a software engineer and the Systems Programming Manager at The University of Queensland's Faculty of Engineering, Architecture and Information Technology. Jon has over 10 years experience working with and managing IT and software projects on diverse platforms in the higher education sector.

Introduction

In this presentation I'm going to take you through some of the systems I and others at UQ's Faculty of EAIT have been working on over the last 10 years or so. We have a focus on investing in development of systems to support ourselves and our users, because we've found that this significantly reduces our support load, and creates a "virtuous cycle" of further investment, compared to the other approach which is a pure support model where technical debt leads to much higher support costs.

Some of the core things we've learned from our approach that I'm hoping to convince you of today are:

- Data is the most important part of any system.
  It's more important than the choice of software you use to work with it, and flexible easy access to the data should be a fixed requirement. Finding yourself locked into a vendor severely limits your options, and your data will usually be locked away, unable to benefit you in unexpected ways down the track.

- Sensible innovation can be easier and less costly than you might imagine.
  It can be very scary trying to create something yourself, and the temptation is always there to look for a turnkey product. Our experience has largely been that if we're not completely happy with a product and it's not designed to be customizable or extendable, then it's better to look elsewhere or roll our own. Conversely, our experience with our own systems has been largely positive, and others share our view as several of them are in use outside our immediate area now.

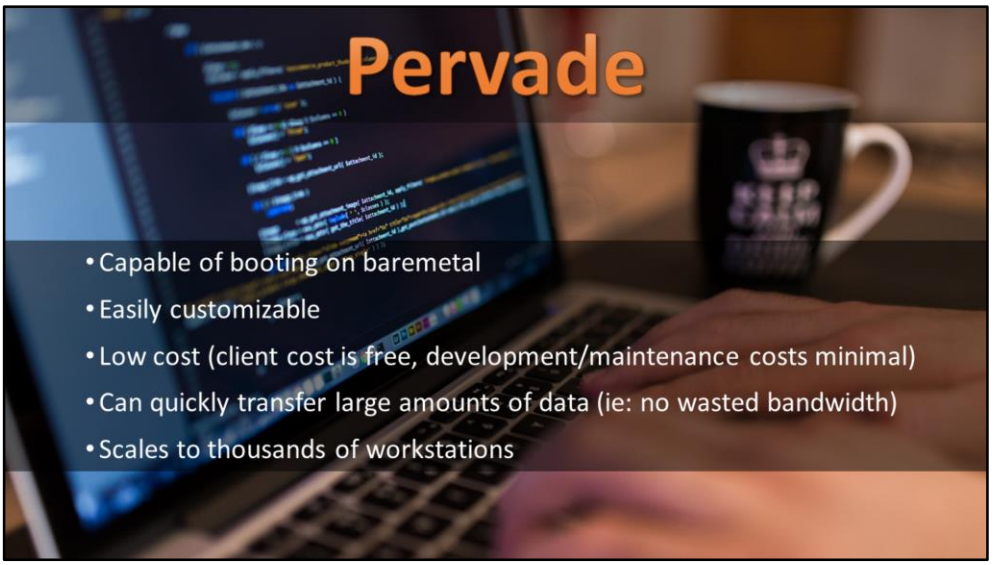
- Seeing beyond technology assumptions expands the options you have available to you.
  This is hard to explain but ultimately what I mean here is that if you look at what something actually does rather than what it's traditionally been used for, you will find lots more options for using when you need to glue things together, or "innovate around the edges" (because I'm obviously not advocating starting from scratch on everything every single time!)

So the first system I'll talk about I think highlights this approach in a fairly accessible way.

Pervade

We have a number of challenges on our coursework network when it comes to deploying or repairing operating system and software environments. For a start, we have over a hundred gigabytes of software we have to support on a network with around 750 individual machines, and many of our courses require that our users be given administrative access (we teach IT – they need to be able to break stuff to learn!)

Our existing product “Rembo” (though it became “IBM Tivoli Provisioning Manager for Operating System Deployment with Toolkit Addon”, and was then discontinued as a product) was really starting to reach the limit of what it could do, and we couldn’t find anything to replace it that met more than a few of our core criteria.



Pervade

What we needed from a deployment system was:

- Capable of booting on baremetal
- Easily customizable
- Low cost (price per workstation, though development costs were considered also)
- Can quickly transfer large amounts of data (ie: no wasted bandwidth)
- Scale to thousands of workstations

So we went looking for components (preferably free and open source) that we could glue together to achieve all these things. The solution to the first three of these criteria was pretty obvious to us.

Linux

Linux can PXE boot on baremetal, it's fully customizable, well supported and maintainable, it's widely used, and it's free. It also gives us a broad range of options down the track should we need to support additional functionality. The process we went through to set up the Linux based deployment system was:

1. Set up a TFTP server and pointed the clients at it via DHCP. Pretty basic.

2. Create a bootloader that loads a configuration file and PXELinux based GUI menu system from our deployment server via HTTP.
   Doing this meant we had full control over what configuration a particular machine was given, and allowed us to do one-time-boot options (ie: we could configure a machine to default to a full deployment, and then on the next reboot the default has been reset to simply boot to the hard drive).

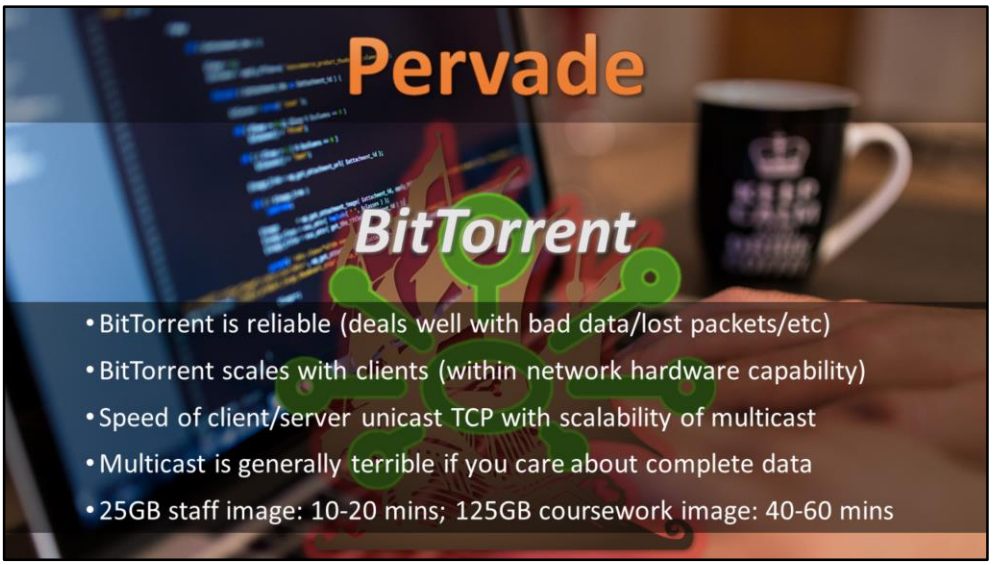
3. Write the snapshot and deployment scripts.

   If we're doing a deployment:
   - create an EXT2 cache partition at the end of the drive for us to store the downloaded image
   - create the partitions at the start of the drive windows is expecting
   - ntfsclone(8) –restore-image
   - mount the partition and inject driver packs, and any other items we need to
   - reboot into windows initial setup process

   If we're snapshotting an image:
   - ntfsclone(8) the partition via gzip(1) through an ssh(1) connection to the deployment server

The next problem we had to solve was how to get the OS and software image onto the cache partition. Doing our best to focus on the problem rather than traditional solutions, we went looking for something that at its core was designed specifically for spreading data as far and as fast as possible to as many clients as the network will allow.

It turns out, there's something that millions of dedicated volunteers have been thoroughly testing and helping to refine over the last fifteen years designed specifically to solve this exact problem.

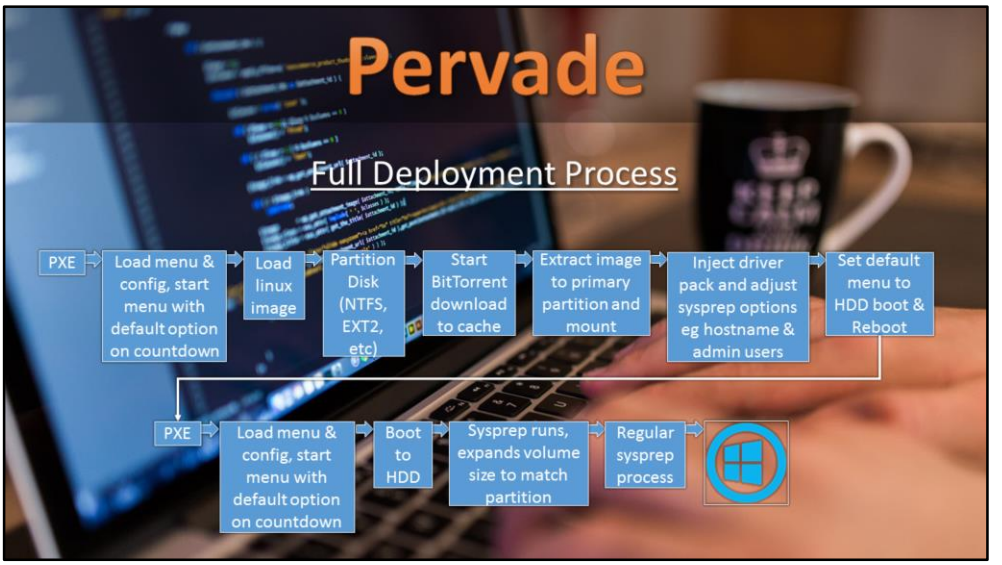


BitTorrent

BitTorrent is actually a really nice protocol for transferring large files around to multiple machines at once, and there's plenty of libraries and clients available for Linux. Getting it to work is pretty simple too – all we need is a tracker running on the deployment server that creates torrents from the uploaded snapshots and then seeds them. It's easy to control, and does a reasonable job of spreading pieces around so that even with a single seed, the swarm rapidly saturates available network capacity (typically the limit is the disks in the client machines, rather than the network speed). You get performance similar to a single unicast TCP client connected to a server, but with the scalability of multicast.

Our staff image is about 25GB and takes between 10 and 20 minutes to download and write to the primary partition, and our coursework image is around 125GB and takes between 40-60 minutes to download and write to the primary partition. The difference in imaging times doesn't heavily depend on the number of hosts imaging, but mostly comes down to which building the imaging occurs in; and typically the buildings with the older switches or more bizarre physical network topology are the slowest (as you'd expect).

We've found BitTorrent as a mass image deployment method to be very reliable and flexible – for example, if we had a slow link somewhere and we needed to improve the performance of imaging at the end of that link, it would be as simple as setting up a mirror seeding client, and configuring the remote hosts to use that instead.

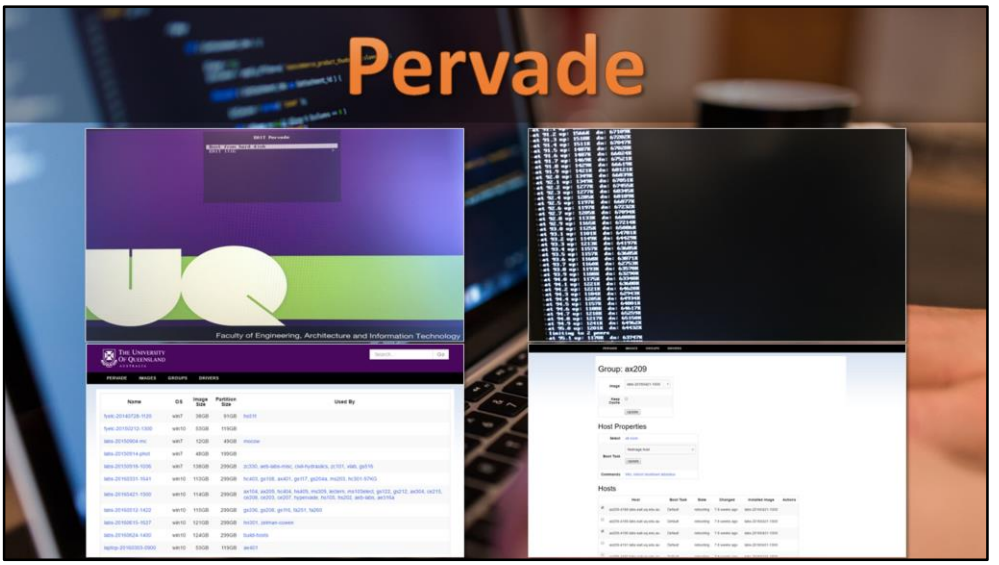


Process Overview

The full process for deploying a new client OS and software stack using pervade on baremetal is as follows:

1. Client boots to PXE
2. PXE loads the menu and config files for the client
3. The default menu option, set via the management interface to reimage the client, is selected by default with a countdown timer. It would be possible at this point to select another option if desired, though the more advanced options (such as imaging with a different image or running other tools are password protected.
4. Once the reimage process is selected or the timeout autoselects it, the full reimage process is started. This begins by downloading the Pervade Linux image.
5. Pervade creates the on-disk partitions – namely, an EXT2 cache partition is created at the end of the disk, and the NTFS (and other) partitions for windows are created from the remaining space at the front of the disk.
6. BitTorrent clients are started up and the image begins to download (and, in the case of multiple clients downloading the same image, upload to other clients).
7. Once the torrent has completed downloading, it's extracted into the primary partition, and mounted.
8. Pervade then injects the relevant driver pack for the hardware type, sets some sysprep options such as the hostname and admin users (based on other systems we run), and optionally performs any other steps we need it to.
9. Now that we have the image deployed, Pervade changes the default menu option for this host to "boot to HDD", and reboots the computer

1. Client PXE boots
2. As per the last boot, the menu and config files are loaded, but this time the default option is to boot to the HDD.
3. Windows first-run is started as per the regular sysprep process. Pervade has configured the sysprep config files to run a command early on to resize the NTFS volume to expand and fill the primary partition, so there's no wasted space due to geometry mismatches on the build host and the client machines.
4. The regular sysprep process completes, and windows then does its normal startup processes, and is then ready for clients to use.



Show and Tell

Boot Menu

This is what the boot menu looks like (it's set to default to HDD boot in this case). As you can see through the glare in my terrible photo (apologies), it's fully graphical, branded, and functional.

The EAIT ITIG menu is a password protected section that allows for more advanced options.

Torrent Download

This is the rather functional interface showing the progress of downloading of the image over bittorrent. We could probably put a nicer display on this page, but it's less user facing than the boot menu and we prefer full debug output in our environment.

Images

In terms of management and usage of the Pervade system, there's a basic webpage we've set up that lets us view or control things such as host groups, images, driver packs, and configure a host group (or part of one) to update to a new image. I'll quickly cover two of the screens and then wrap up the discussion of Pervade.

Shown here is the Images screen which shows a selection of our images, what OS they are, the size of the data, and the primary partition size. It also shows which hostgroups are assigned this image currently. This helps plan image rollouts and check compatibility before deploying new images.

You can click on an image name and get a more detailed readout set of information on it, and a few controls to do with deleting or replacing the image. Clicking a hostgroup is slightly more interesting though…

Group

Viewing the group page gives you a bunch of controls to set the image and send commands and tasks to the hosts. There's also a section which shows you the information on a host by host basis of what image is currently installed, the state, tasks, and age of the current config.

I won't go into too much detail on these items – they're mostly just changing database settings which determine what configs are sent to the clients when they request them from the deployment server, though some of the commands touch on the system I'll cover next.

Lab Status Monitor

For as long as I can remember at UQ there's been a desire to reduce the number of lab facilities we have to maintain and spend money on. It's a pretty common situation – everyone wants to reduce expenditure, and laptops, tablets, and remote users seem like a great way to do that.

Of course, we also face resistance from course staff who want to provide lots of software and facilities for their students, and students themselves who often complain they aren't given enough facilities to support their studies.

Our general feeling was there was plenty of availability outside of a few key hours of the day during a few key weeks (generally end of semester), but it's hard to argue from opinion, and so we wondered if there was any way we could gather information on actual computer usage. We'd played around with windows security logs, and samba share mapping logs, but it wasn't really accurate enough for us.

So back in 2005, while I was waiting for my computer to rebuild after yet another hard drive crash, I started poking around in Visual Studio to see if there was any way to monitor and report who was logged in. By the time my computer had come back up, I had a passable working service written in C# that could report reasonably accurate usage statistics to a webserver, and a basic script and database on that webserver to record the information.

Over the next few months of semester, we realised the data was so useful we put a really basic bar graph showing availability in our labs onto an LCD monitor we screwed onto the wall near the foyer of our main lab building. Two things became clear out of doing that:

1. Trivially, marketing and admin wanted to use the screen for display notices as well, since everyone now looked at it all the time, and so our (in retrospect unfortunately named) "PPDisplay" system was born, allowing users to upload power point slides, images, videos, webpage urls, etc, and
2. We needed to beef up the capabilities of our data gathering statusmonitor client and the associated availability display systems – it needed to report more information more accurately, use less resources, and be even more useful to the students. We had very positive feedback that it was improving their ability to find a computer, but it needed to be available in more places and show more detail.

statusmonitor.exe

To start with, I rewrote the .NET client, which used quite a bit of memory (back then in 2005, the 64MB of .NET assemblies in ram was about 20% of the total memory available) and made access to the Terminal Services API more challenging, in C (well, technically C++). It's a small 2MB binary that loads configuration from a config file, logs to a logfile, and reports back instantly when states change by hooking logon/logout/lock/shutdown etc events, and sending status updates back to the server via WinHTTP.

Over the years it's supported various changes in the windows APIs (console sessions moving from session 0, 1, 2, and then floating entirely in Windows 10) and methods (the early versions of clients even supported Windows versions prior to the introduction of the Terminal Services API by falling back to checking process lists), but more recent versions I've removed some of the older code (structural changes mean I have to keep retesting all versions, and it's getting harder and less useful to test old windows versions against changes).

The service installs itself by running it with --install and to remove it by using the --remove switch. When run as a service it redirects output to a log file, and as you can see from the screenshot here where I've run a test on the command line, it can be configured to give detailed debug information and perform config and reporting system tests.

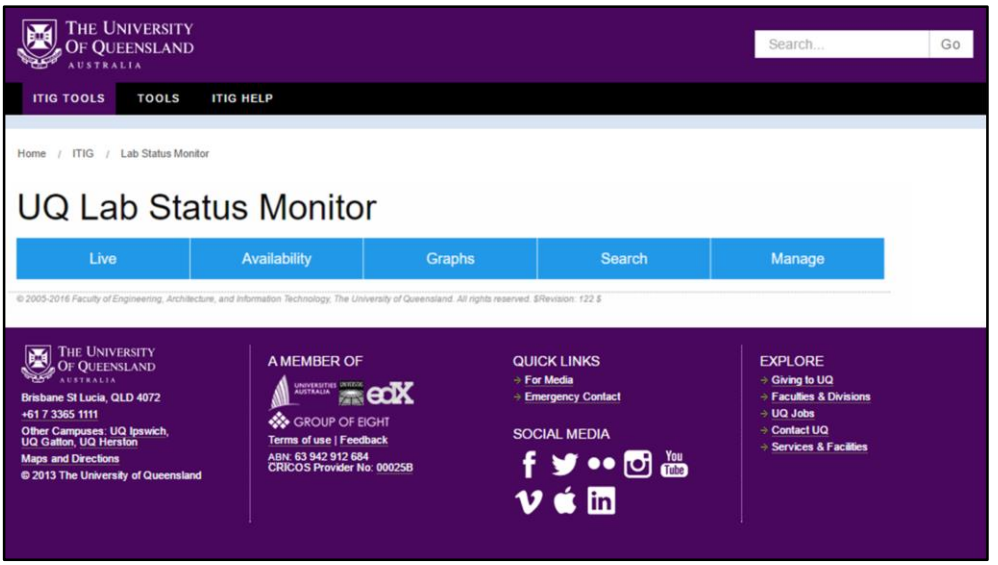
Information

Over the years, more and more information has been gathered as it was useful to know, and our full control of the client means it's very easy to extend the code to add the new information checks. Some of the information we're collecting includes:

- Status (Available, Occupied, Locked, etc)
- User logged in (if any)
- System uptime
- OS version
- Domain
- OS/software image name and version

We are actually looking at using it to collect information about usage-hours of software – ie, to determine whether something that was requested by course staff was actually being used, and if so, how much and where. This might make it easier for us to build smaller, faster images and just post deploy something via SCCM or another technology if it's large, rarely used, and generally used in the same place.

Of course, gathering all this information is good and all, but we also need a full system surrounding it to display it, manage it, and while we're at it, being able to control the clients remotely or message users would be nice too (eg, to save us having to physically visit a lab and log out a user who has clearly forgotten to log out).



Management

So this is the very boring front page for the Lab Status Monitor web interface. I'll briefly explain the five options here and then take you through a tour of them.

Live

This is the live UQLSM management interface. This is by far the most useful for lab managers. It gives you a live view of the status of the network as a pseudo bar chart with various controls over how it's displayed, and allows you to right click on an individual host to get full information and perform control actions which I'll go over soon.

Availability

This is the section of the system useful for students. We host this part on our student-accessible IT help site and display the floorplan components of it regularly on our fleet of display systems that are located near labs.

Graphs

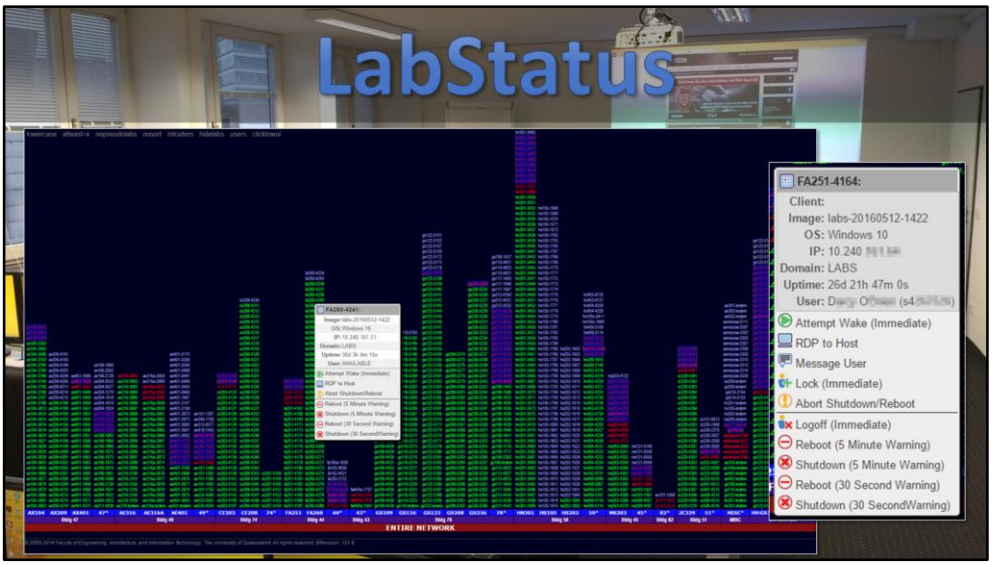
This is the part that's useful for capacity planning. You can show full usage graphs between arbitrary dates, as well as optionally overlaying "business hours" and room bookings. Using this we've been able to identify labs that were surplus to requirement and reduce our expenses.

Search

This allows you to search host or user history. Typically this is used during investigations into facilities or academic misconduct.

Manage

This allows you to define labs (groupings of computers), assign hosts to labs, clear out unnecessary status entries, define floorplan mappings, etc. It's really just a wrapper around some of the config and live status tables in the database.



Live

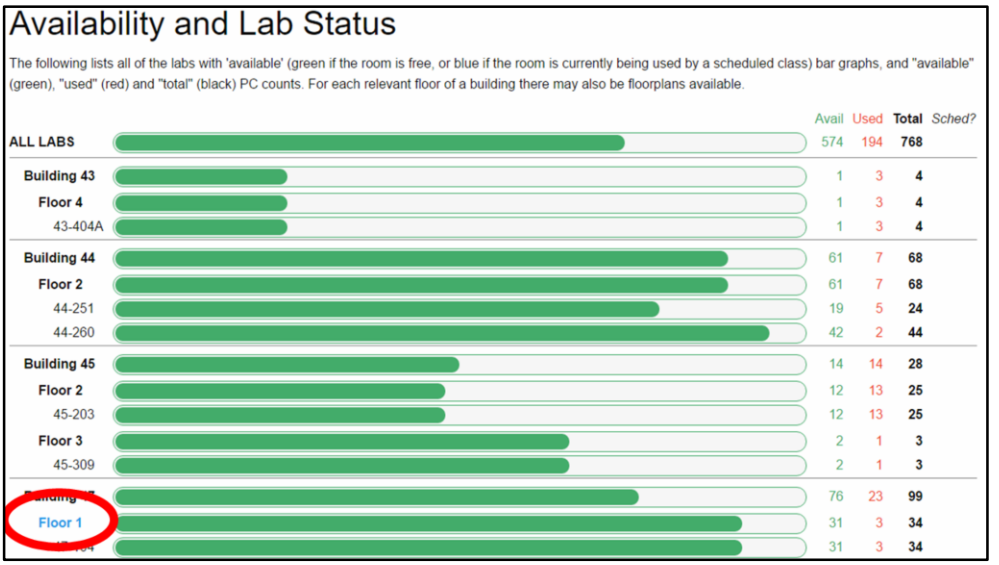
As you can see here, this is a really information-dense screen. It shows the total status of our ~750 computers grouped by lab and building, with hover text and context menus for full information and control, displayed as a pseudo bar chart. Note, the colour codes are the same as used elsewhere in the system, but to save space they aren't shown currently on this view.

Right clicking on a host brings up set of contextual options, depending on the state of the host. Quickly going over these options:

1. You can send a WOL packet to a host to try and wake it up if it's off or unknown
2. You can RDP to a host that's currently up (the RDP file can be optionally signed for instant connections)
3. You can send a short message to a user that's logged in
4. You can lock the screen for a logged in user
5. You can logoff a logged in user.
6. You can shutdown or reboot a host that's up, with a variety of delays
7. You can cancel a requested timed action

You're also able to right click on a lab, building, or the entire network, and similarly, you'll be able to send those commands to all the hosts in one hit. Yes, you can immediately log off everyone on the network with two clicks. Well, three now, since we added some confirmation dialogs :D

In the background, these commands are executed via a service running on one of our coursework lab network servers. There's another service we've written there that listens for these commands and executes them (currently via **psexec.exe** running as a suitably privileged user, however at some stage I'd like to update the statusmonitor client again to make a persistent TCP connection and allow commands to be sent via the back channel), but to save time I won't bother going into too much detail here. It's written fairly similarly to the statusmonitor client itself, but it contains some options for IP locking, strong sanity checking, and other stuff to try and prevent abuse. It's also obviously heavily firewalled off from the students and other systems.

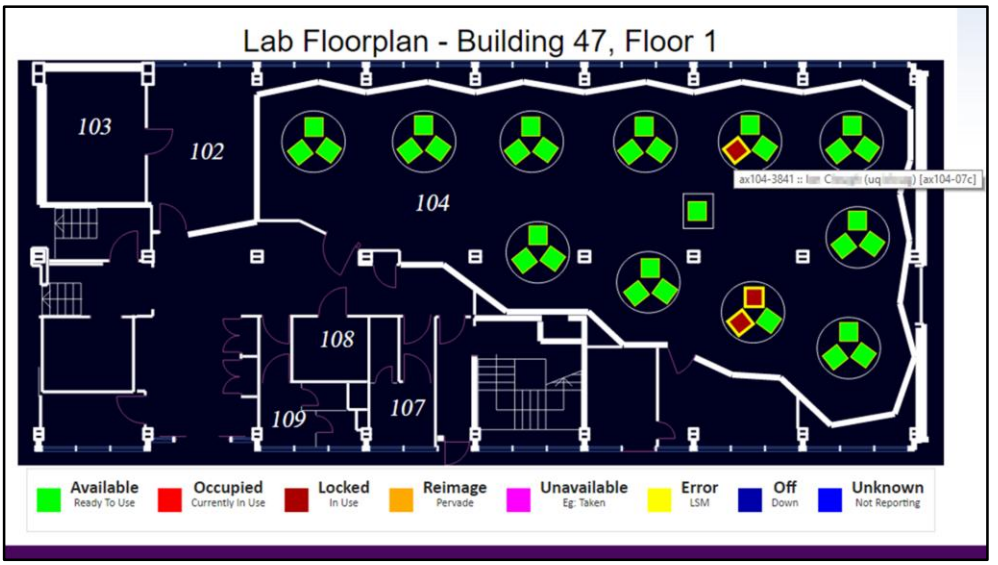


Availability

This is the view students see on our studenthelp website. They can very quickly see which buildings, floors, and labs have the most free computers, and if they're currently in a booked class session or not.

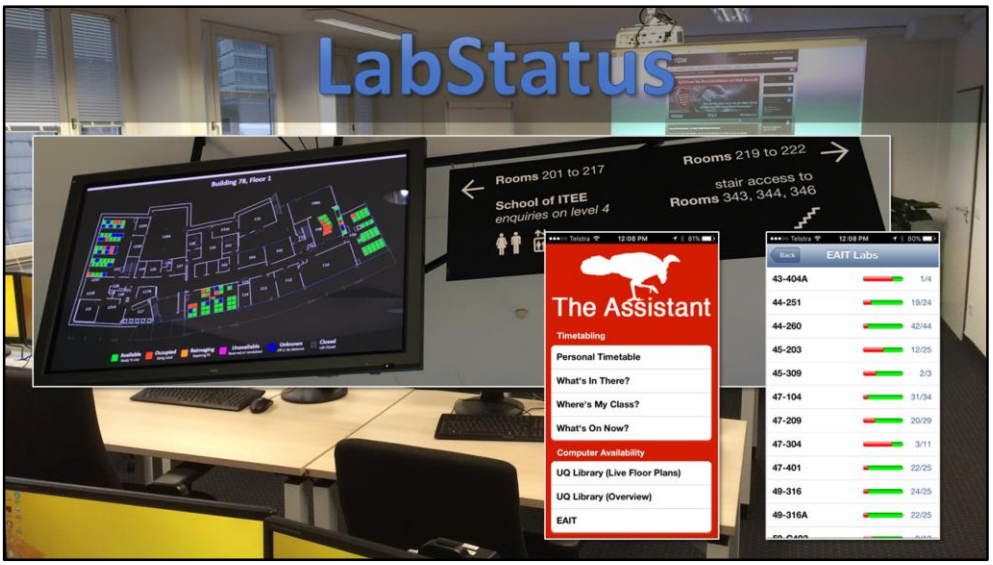
In this view I've also included a floorplan for one of the floors, as you can see here.

Clicking that will take you to the floorplan for that floor which will show you exactly where in the room the free computers are.



Floorplans

This is what our floorplans look like. You're currently seeing the "admin" view. Our student view significantly simplifies the states by grouping most of the states into "Occupied", and obviously doesn't allow you to hover on a PC and get titletext detailing the name and username of the logged in user like what you can see here!

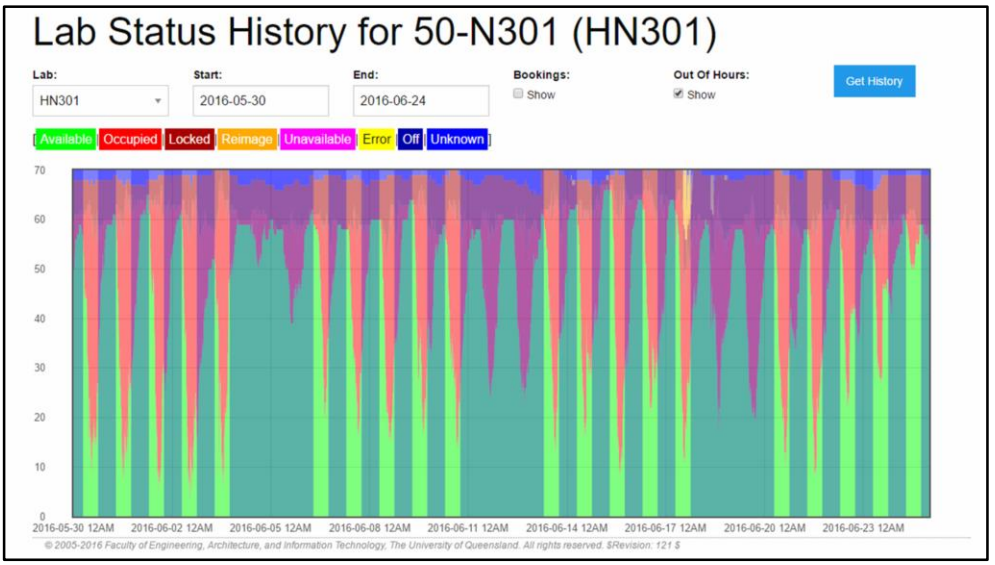


Student Experience

We display the student version of these floorplans in high rotation on our display systems near labs for example like the one shown here in the level 2 foyer of our main building (not far from where the original LCD bar chart screen was placed all that time ago), and once we started doing this, complaints about not being able to get a free computer almost entirely evaporated.

It's a pretty big win to be able to take what seem like incompatible requirements and satisfy them both. We can shrink expenditure on labs and recover space for other endeavours as well as ensure students have fewer to no problems getting a computer for use in their studies.

Of course, we also have this data now and in aggregate form it's publicly available, so we saw no reason we couldn't provide some basic data via an API to end users. For example, the developer of the (now no longer free, much to my chagrin) "UQ Assistant" app uses our APIs to provide availability data for EAIT labs to their users.

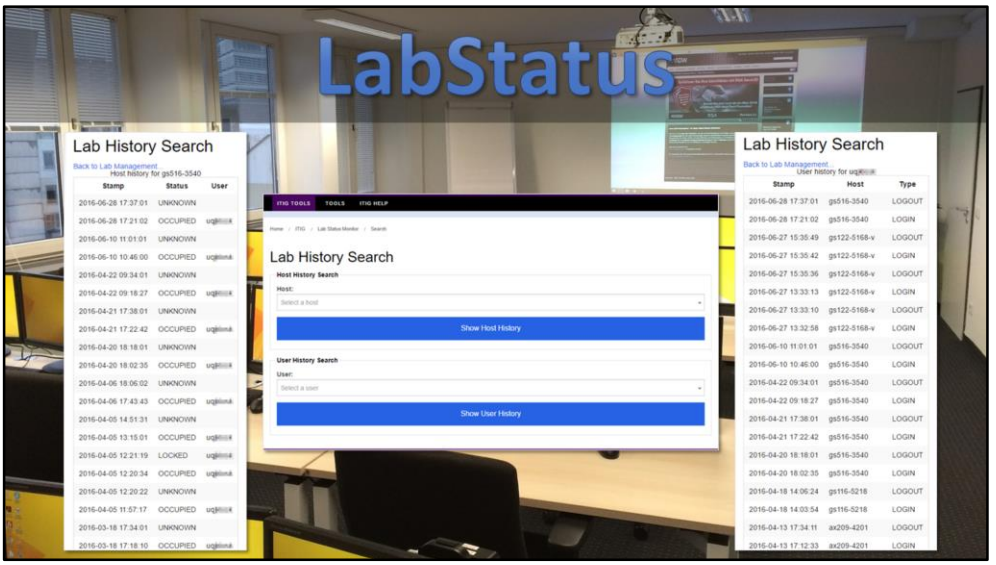


Graphs

As you can see here in this example, I've queried the system for the four weeks of data from our largest lab ending on the last week day of exams. At these large scales, individual peaks and troughs are smoothed somewhat, so this doesn't reflect the fact there were probably some sections of 100% usage during some of those peaks. But clearly, overall we still have plenty of capacity.

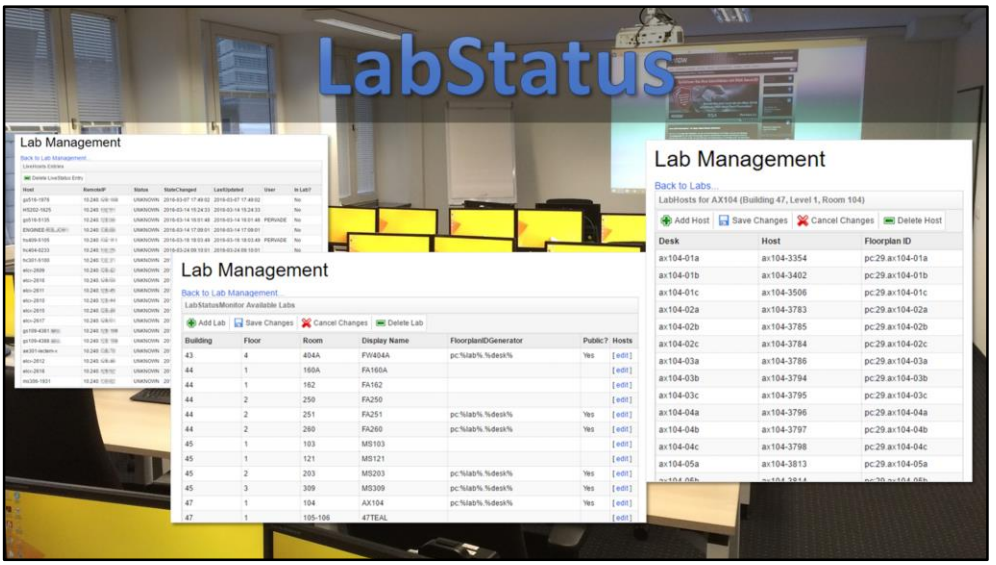
Or do we… we're showing late and night and weekends here as well, so how much of that ocean of green isn't reasonable to include? Well, we can overlay the "out of hours" blocks on top of the data to make it a little easier to visualise.

We can do a similar thing for lab bookings – we discovered some courses were booking rooms "just in case" and not actually using them using this. Thankfully that practice is declining now due in part to this kind of data display.



Search

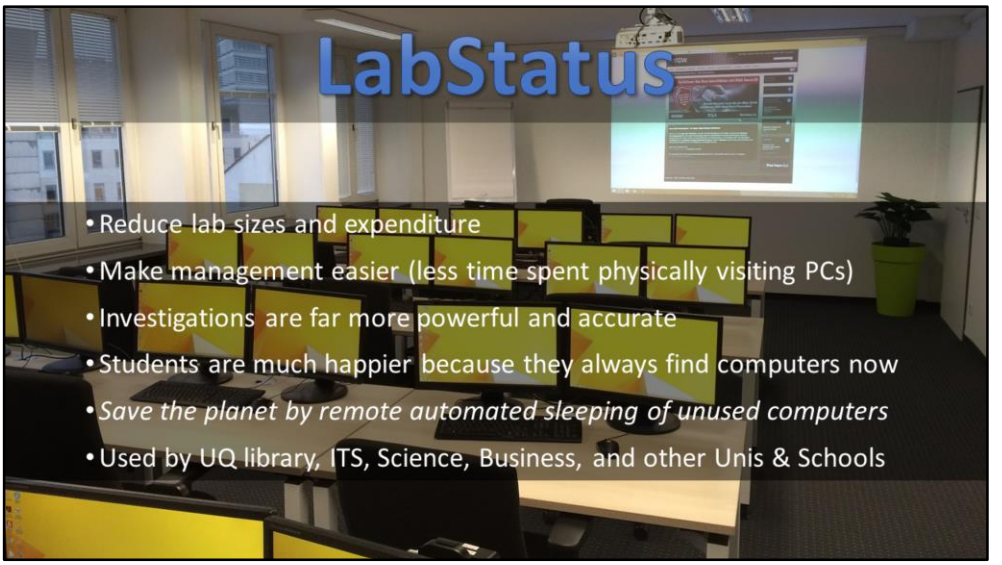
Used mostly for investigations (ie, who was logged onto this computer at this time, or when did this user log into a computer in this building last). You can search by user or hostname, and the results look like this.



Management

This section of the system is mostly just a wrapper around the config tables in the system. You can either clean up the livestatus table (all reports from hosts are recorded, even if they aren't assigned to labs, and when you replace all the hosts in a lab, the old entries are left here and orphaned), or manage labs, hosts, and mappings between them and floorplans.

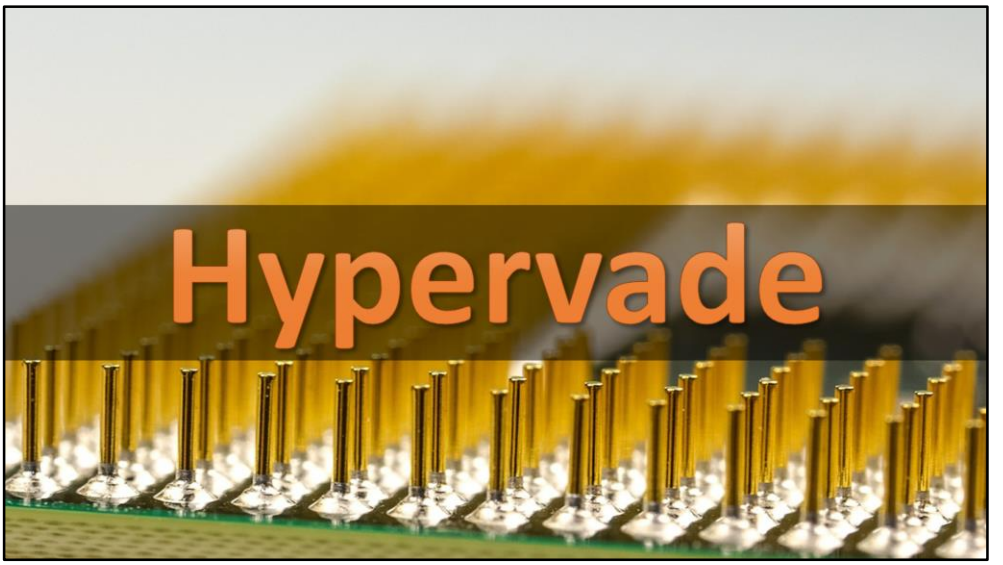
This just uses standard jQuery DataGrid components and is pretty basic.



Benefits

So concluding all the benefits of implementing the Lab Status Monitor, we've managed to:

1. Reduce lab sizes and expenditure which also frees up space which is often in demand
2. Make it far easier to manage computers (most of the time we don't need to visit physically to turn machines on, log out abandoned sessions; and we can message students to ask them to do something like "Close the door please" if they're right next to it and it's propped open to bypass security).
3. Investigations into academic misconduct or facilities abuse is much easier and more powerful now.
4. Students are much happier because they can pretty much always find a computer now.
5. As a side effect, we now have full scheduling and usage information for labs, so we can save the planet by shutting down unused computers. We have a project right now with the UQ Sustainability Office to do just that – we've already done studies on power usage and found very little difference between sleep and off, so we plan to update the lab control system to include a sleep command, and write a small monitoring system that will sleep a machine that's been available for more than 10 minutes outside a scheduled class, or outside business hours (depending on the config of a specific lab). Again, this is the flexibility and power you get when you have control over the system.
6. This system is already used in a very large number of areas around UQ, and has been for quite a while now. It's kind of fun to see how many UQ students know about the StatusMonitor. I'll find myself at a pub or a party and someone who has told me they studied at UQ asks me "what do you do", and I'll start off with "you know when you wanted to use a computer in the library, and you used those floorplans to see when one became free?" Invariably, their eyes will brighten and say something like "Oh yeah! Those things are so useful!". It's really nice to be able to smile and say "I wrote that."



Hypervade

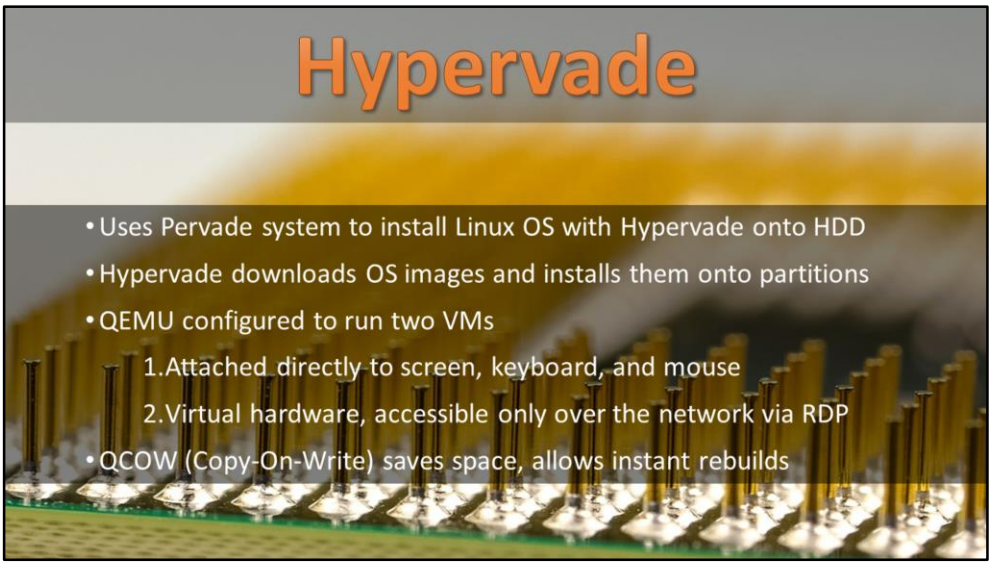
This is where everything starts to come together. We have a highly customizable and efficient way of getting software images deployed, we've optimised the number of computers in our labs, and our students are having less trouble getting access to computers.

But a lot of students like to work on their laptops or their desktops at home, and for licensing and technical reasons sometimes this is either not possible or would create a very large support burden for us.

Ideally, we'd like to provide the same environment (including easy access to our network storage) as the students have when they are working in the labs (there's nothing more frustrating for students than working on projects at home, then coming in to demonstrate them on campus only to find the environment is just different enough to prevent it from working).

One solution that immediately springs to mind of course is Virtual Desktop Infrastructure – run up a farm of computers that are remotely accessible and configured the same as the lab machines. But that can be expensive in terms of hardware, maintenance, and the standard data center resource management required for this. Especially given how idle our lab computers with 8 cores, 16GB of ram, and enormous HDDs can be, even when someone is logged into them.

Happily, the whole IT industry has been working on ways to make computers think they're actually multiple computers for quite some time now, and this solution is excellent because it gives us all the benefits of adding extra dedicated remotely accessible machines with the exact same environment as is available in the coursework labs, but we get none of the drawbacks of extra hardware outlay and further reduced utilisation efficiency – in fact, because we effectively get two bites at the apple, our efficiency goes up even further.



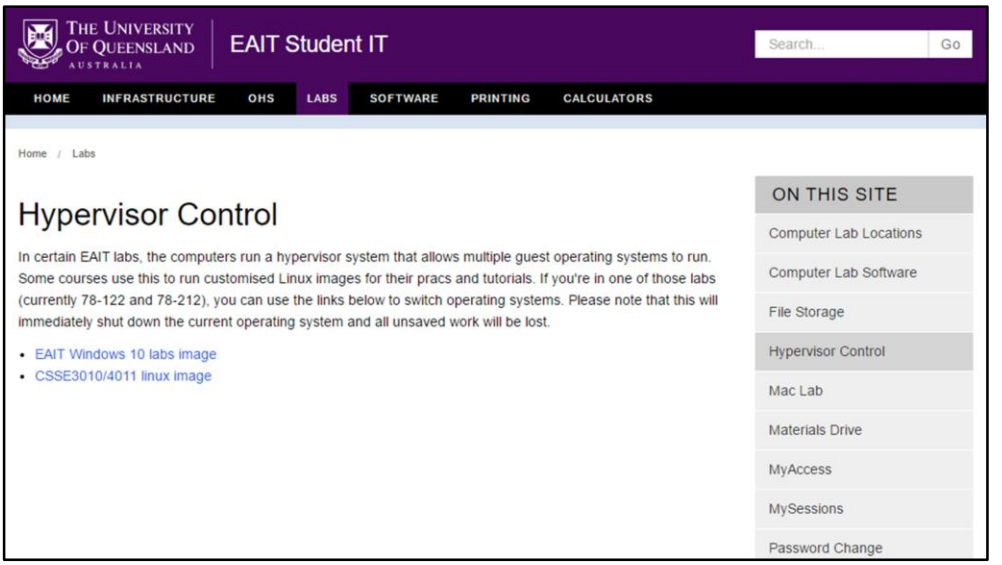
Hypervade

Hypervade starts out pretty similarly to Pervade – there's a set of PXE boot options, a management page to control the options and defaults, and controls to set systems rebooting or reimaging. The main difference though, is instead of directly imaging the machine with a copy of Windows, the Pervade Linux OS installs a Hypervade Linux OS onto the disk, and then boots to the hard drive and this Linux OS instead.

Once the Linux Hypervade OS is running, it downloads the image (or images – we can have multiple images if necessary; more on that later), creates an NTFS partition for it, and extracts it exactly as Pervade would normally do. However, instead of rebooting into this windows image, Hypervade configures and brings up two QEMU VMs – one attached directly to the screen, mouse, and keyboard, and the other connected to virtual hardware and accessible only over the network via RDP.

Hypervade uses QEMU QCOW (QEMU Copy On Write) files to ensure the guest OS partition is not modified, and this allows almost instant rebuilds, but also a lot of space efficiency to reduce the necessary disk space.

Each of the guest OSes can be managed as normal as though they were real machines, but we also have a fully customizable Linux host OS also running, which lets us do some clever things. For example, as the next slide shows, we can let the console guest OS talk to some well defined controls on the hypervisor.

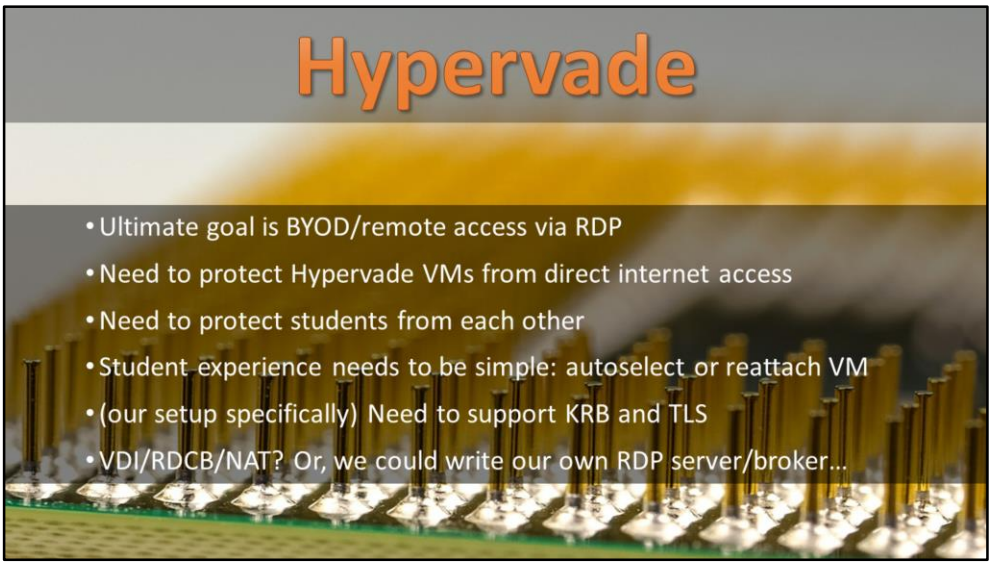


Hypervisor Control

Here you can see one of the more interesting capabilities this system allows us to perform: user initiated OS switching. By simply visiting a webpage and clicking on a link, the user can immediately terminate the current guest OS, and have the hypervisor reattach a new OS and start it up.

Here, a course required a Linux OS for teaching, and when the class is in session they are able to quickly switch from the default Windows image into Linux with a few clicks on a webpage. From inside Linux, if they wanted to switch back to windows, the same approach works – visit the webpage, and click the Windows 10 link instead.

There's many other possibilities here, and we've only just begun to explore them. For instance, we recently used the system to aid us in an investigation where we were looking for a software keylogger (boring conclusion: there wasn't one), but I think this is a great demonstration of the real power and flexibility of this approach, even without the BYOD/remote access component of it which I'll discuss next.



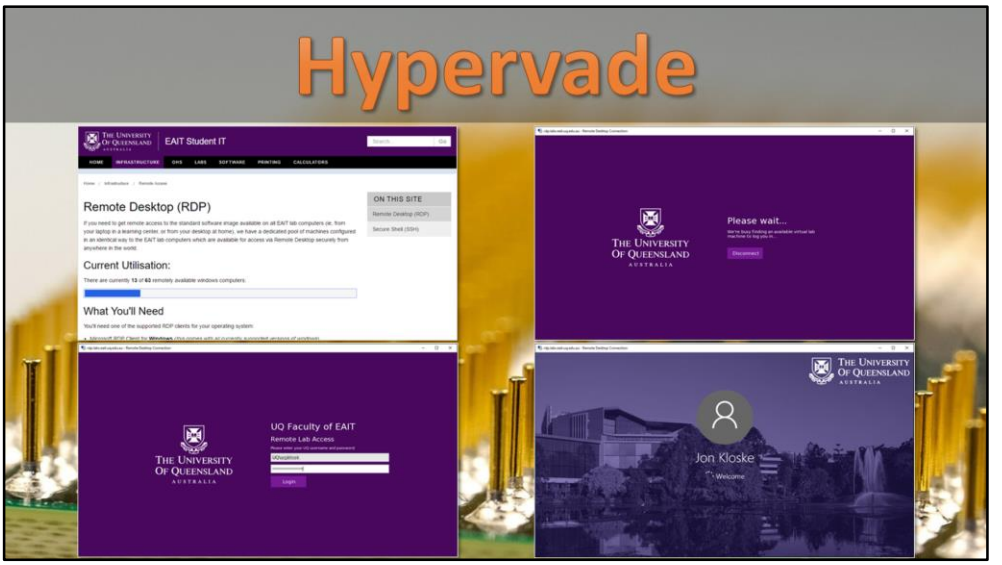
BYOD & Remote Access

Of course, the ultimate goal here is to support BYOD and Remote Access. We now have a number of virtual lab machines available for use externally, but it would quickly become a mess if we simply allowed them all to be accessible externally and let students self manage sessions. One immediate problem is students would be able to log each other out. Another is it would be a security risk in general to expose every single machine to the internet, even if it's just on 3389 RDP. But again, this is about the student experience, and having to hunt around on a webpage to find out which machine you were logged into so you could reconnect to your session sounds pretty boring.

Ideally we'd want a single RDP connection broker that students could connect to, authenticate, and have it automatically assign them a random free virtual computer, or if they already had a session, reattach them to that session. Microsoft has RDP connection brokers for their server line, but that would possibly require us to run server OSes and deal with all the complexity of that (and licensing, compatibility, and maintenance issues?) or play around with VDI. We could also do something with a firewall – reverse NAT with sticky states (though reattaching from elsewhere or dynamic IPs would be tricky to work with). These may be viable options, but again, we wanted a little more flexibility (and we wanted to solve a really annoying bug in Microsoft's RDP services that means we can't use Kerberos authentication with TLS).

So we did something really crazy. We read up on the Microsoft RDP protocol documents, and wrote our own RDP broker in Erlang. We found all sorts of interesting deviations from the spec, and plenty of fun gotchas, but we now have an RDP broker that we can tie into our other systems to ensure intelligent behaviour for our users, and satisfy all our other requirements around security and user experience.

How does this all look to the end user? I'll show some screenshots of the system in action now.



Show and Tell

The process for end users who want to access the Hypervade virtual lab infrastructure is the same whether they are on campus using the eduroam wireless network, or at home connecting from their desktop computers.

Help Guides and Availability

Firstly, we have a webpage accessible to current students and staff that explains how to remotely connect using RDP clients for their specific OSes as well as listing how many free virtual computers there are (so far we've never had 100% utilisation, though as you can see from the screenshot, when I snapped that we had only 13 free VMs).

RDP Broker Login

Once a student has followed the instructions and opened an RDP session to our RDP broker, they're presented with a custom login window where they can enter their username and password to log in.

RDP Session Management

They're then presented with a wait screen while the system finds a free computer. If no computers are available, they'll be placed in a queue until one becomes free. I've never seen the system spend more than a fraction of a second on the wait screen – as far as I know, other than during initial testing, we never had a situation where there were no free computers.

Attached to Desktop

Finally, you see the Windows 10 login screen for the system you're connected to, and are logged into the computer and can start using it.

Hadoop

We had some research groups come to us in around 2005 asking for suggestions for running some very large but embarrassingly parallel computations for optimising some set of problems (I think it had to do with robot machine learning or something). We had a large number of computers on our labs network that weren't being used out of hours, so we set them up with psexec.exe and some suitable credentials and they went and manually ran this stuff. They ran these simulations for 12 hours a night, 7 days a week, for about a month, across approximately three 30 seat labs. In the end, they managed about around 25 CPU-years worth of work and got a few papers published with the heatmap data from the results.

We had a few other people running some other large compute type jobs like this (and, incidentally, one project testing bittorrent performance over the LAN which was what initially made us think of using it as an imaging system) so we decided to look into ways to make this a little more robust than manual ad-hoc-ery. Hadoop seemed like a good place to start.

Design

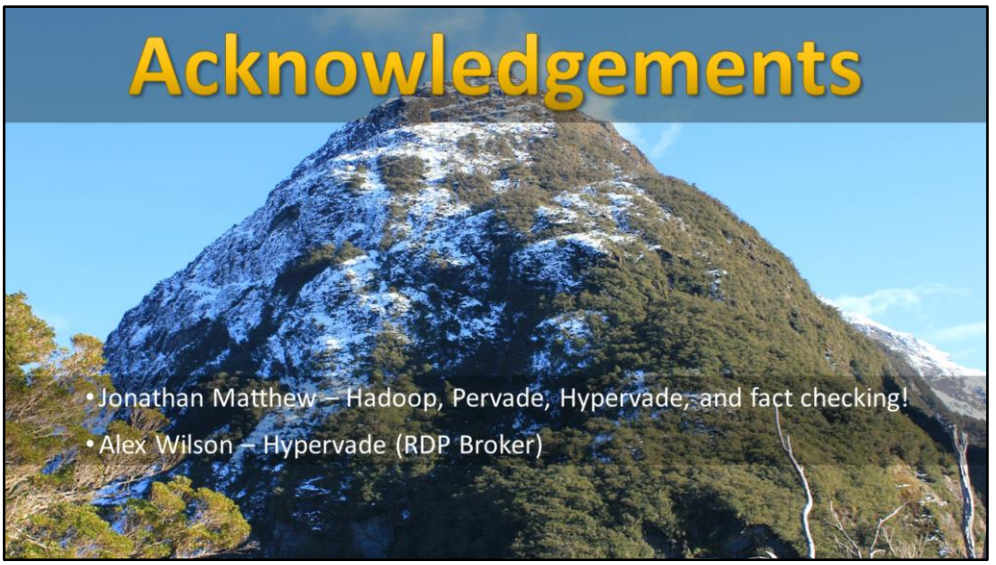
We have a coursework Unix server which we used as the head node for the cluster. On each of the participating lab machines we created a user with very few privileges and setup the Hadoop nodes to run under this user account. From there, it was pretty basic – users could launch jobs from the Unix server and have the work performed.

We ran into a few problems with Hadoop – because these systems were being used, end users could mess with settings or the environment. As a result, sometimes jobs would run differently depending on which client it ended up on, or worse, fail because someone had deleted something or renamed something it needed. We also found that if we ran fairly big jobs in labs with students currently using, they'd get annoyed and kill the tasks ☺

One thing Hypervade allowed us to do was to run the jobs inside the hypervisor Linux instance which gave us a little more control over the environment.

We're working on a number of changes at the moment to how we do compute clusters – we run a significant Smart Data Center (SDC) cluster for coursework or user web or application systems allowing us to scale really well and isolate systems better from each other. We also have recently set up a Manta Storage Service that we're going to re-use with a bunch of other specialist compute hardware together to provide the backend (rather than run it on our lab computers).

The main reason we're moving away from our original Hadoop cluster on the lab computers is that whilst the lab computers are sitting idle, we've measured their power usage at very little, whereas during computation they tend to use a lot more power (an order of magnitude higher). This puts additional load on building environmentals, and also increases the wear on our computers. We get much better economics on dedicated compute cluster hardware, and the benefit of having it all sitting behind a cluster is that we can keep switching these designs in and out, scaling up or down, adding or removing capacity, and it's largely transparent to the users.



Acknowledgements

I'd like to thank Jonathan Matthew for his assistance in fact checking this presentation. I obviously didn't write all the systems presented today, and so I appreciate the time he took to fill in a few of the gaps in my understanding. Any errors are of course my own.

Hadoop, Pervade, and Hypervade are largely Jonathan's babies as well. There's input from other people (Alex Wilson, David Gwynne, etc) but Jono is primarily responsible for them.

The Hypervade RDP Broker is Alex Wilson's work, and the Lab Status Monitor is my own work.

Questions

Ask away. I'm also happy to answer questions by email at jonkloske@gmail.com or via @jonkloske on twitter if we run out of time here, or if you'd just prefer to ask privately.