How to build a powerful distributed computer:

The affordable way to get a computing mega-matrix

By PC Plus February 01, 2009 PC

You can never have enough processing power, especially if you enjoy working with 3D graphics or compiling your own software.

Pleasingly, with a very small outlay, it is easy to use any spare machines you may have to create a single homogeneous computing mega-matrix and calculation engine just by wiring them all together and running the right software.

Even modest hardware can make a significant contribution to your net computing power, and if you've already got the hardware then you've got nothing to lose.

PC hardware is now so cheap that buying a couple of extra machines and wiring them into the same computing pool could make a very cost-effective expansion. This is what we are going to build, and we're going to use Ubuntu Linux to do it. Linux can take cluster computing tasks like these in its stride, and you don't need to fork out for a licence for every machine.

Before embarking on this endeavour, we need to make one thing clear. The combined processing power of a cluster of computers can only be used for certain applications.

You won't be able to boost your frame rate on *Crysis* or *Far Cry 2*, for example, and you won't be able to run your everyday applications across the cluster unless they're designed to do so. Neither are we building a *Beowulf* cluster, where you'd need to use specialised libraries and programming languages to take advantage of the parallel processing potential within the cluster.

We're going to work with distributed computing, which involves splitting up a task across several machines in a local cluster. As a result, our applications can be far more down to earth and practical. You will be able to dramatically reduce rendering times with Blender, for example, or the compilation time of major apps like the GNU/Linux kernel.

You'll also be able to parallelise any number of tasks and use each machine separately from your master if you wish. As with multi-core processors, there's some inefficiency when scaling jobs across a cluster of processors, but it will almost always be much faster than without the processor working.

In theory, you can use any old PC. The minimum requirement is that it must be able to run Linux; so that narrows the choice down to almost any PC from the last 10 years. But in reality, the cluster works best if the machines that you're linking together are relatively close in specification, especially when you start to take running costs into consideration.

A 1GHz Athlon machine, for example, could cost you over £50 a year in electricity costs. You'd be much better off spending this money on a processor upgrade for a more efficient machine. A similar platform for each computer also makes configuration considerably easier. For our cluster, we used four identical powerful machines.

You only need powerful machines if you're making a living from something computer-based – 3D animation, for instance – where you can weigh the extra cost against increased performance. We're also going to assume that you have a main machine you can use as the master. This will be the eyes and ears of the cluster, and it's from here that you'll be able to set up jobs and control the other machines.

Hardware compatibility

Linux has come a long way in terms of hardware compatibility, but you don't want to be troubleshooting three different network adaptors if you can help it. And it's the network adaptors that are likely to be the weakest link.

Cluster computing is dependent on each machine having access to the same data, and that means that data needs to be shuffled between each of the machines on the network cluster continually.

Due to the heavy data requirement, wireless is too slow for almost any task, so you're going to need a physical connection between each machine. The faster your networking hardware, the better.

To start with, you should be fine with your standard Ethernet ports as long as they're capable of megabit speeds (100Mbps). If you need greater capacity, gigabit Ethernet cards are also cheap. Just stick them into a free PCI or PCI-X slot, and then disable the slower port if you can.

You'll also need a way of connecting the machines together. A gigabit switch can be bought for around £20 to £30, and this will let you connect each machine together at the maximum speed capacity.

You may be able to use the Ethernet ports on an internet or wireless router, but most will only be able to handle megabit speeds. It might be a good idea to start out using your standard Ethernet ports and your internet or wireless router and upgrade to a dedicated switch and ports later on.

As for general PC requirements, that depends on how you plan to use your cluster. Memory is more important than processing speed because it's likely that you'll be using your combined computing power for large, memory-intensive jobs. That means lots of images and textures in the case of 3D rendering, or lots of libraries, objects and source files in the case of distributed compilation.

Each node in a cluster will normally operate independently of the rest. If one machine has a lower specification, it shouldn't be too much of a bottleneck, but this depends on how you use your cluster.

If you're rendering a 300-frame animation and the slow machine happens to render only a handful of frames, then it has still made some contribution without holding any of the other machines back. But if the same machine is used to help render a single complex frame, it's likely that you'll need to wait for the slower machine to complete its work after the faster machines have finished.

Re-using hardware

If you're building these machines from scratch, you can cut corners. Other than for installing the OS, you won't need a keyboard, mouse, optical drive or screen.

You can even avoid using an optical drive for installation by using a tool called UNetbootin. This generates a Ubuntu USB stick installation from a standard CD image. However, its success depends on the capabilities of the system BIOS within each machine.
BIOS settings are normally accessed by pressing a certain key when you boot your machine. This key is most commonly [F2], but it could also be [F1], [Delete] or [Escape]. Look for any on-screen messages at boot for a clue. From the BIOS you should be able to discern whether your system can boot off external USB devices.

You should also disable 'halt on all errors' if the option exists. This means that your machine will boot even when there are errors or problems detected – which is what you need if the machine complains that no keyboard, mouse or screen has been detected.

Internal storage needs are minimal. You need around 10GB for the operating system, and it's going to be difficult to find a hard drive that small these days if you haven't got one or two handy.

For storing your project data, we'd recommend using either a large drive (or array) on the master machine, or use an external NAS device connected to the same switch as the other machines in the cluster. Linux can mount remote drives onto the local file system so that apps treat the remote storage as if it were local.

Each machine needs to be connected to the switch that we mentioned earlier using a standard Ethernet cable. You also need to connect the switch to your Internet or wireless router. A router is needed to assign a network address to the machines in the cluster; this should happen automatically thanks to the DHCP server running on the router.

If you don't want to use a router (or connect your cluster to the Internet), use your master machine as a DHCP server. This will create IP addresses for each machine on the switch.

Installing the software

With the hard stuff out the way, it's time to install the software. We used the latest version of the Ubuntu Linux distribution, Intrepid Ibex.

Ubuntu is our first choice because there is such a wide variety of packages available, and these can be installed through the default package manager without any further configuration.

This makes installing apps like Blender across your cluster as easy as typing a single command on each machine, and it also means that you can install support utilities such as the Dr Queue rendering queue manager just as easily.

We opted for the Desktop rather than the Server version. This was because we still need the desktops on each node for setting up our various tests, and while we could still do the same with the Server edition (which by default has no desktop), configuration would be easier with a desktop.

Installing Ubuntu is easy but a little tedious, as you need to go through the following routine for each machine. After creating a burned CD from the ISO image of the distro, boot each machine in turn with this disc in the optical drive. If it doesn't boot, check the boot

order in your BIOS. Next, choose the 'Install Ubuntu' option from the Boot menu and answer the resulting questions.

On the Partition Options page, use the entire disk for the installation (unless you're sharing the machine with a Windows OS), and create the same user account on each machine. This makes setting up the shared storage device easier. You should also give each computer within the cluster its own name (on the Who Are You page).

Installation can take up to 45 minutes, depending on the speed of each node. When finished, your machine will restart and you'll need to log in to your new desktop. It's likely that you will also have to install a few security updates; a small yellow balloon message will open if these are necessary. The next step is to install the control software.

New applications and software can be installed through the Synaptic package manager. This can be found in the 'System | Administration' menu. First, you need to search for a package called 'openssh-server'.

In Synaptic, click on the package to enable it, followed by 'Apply' to download and install it. You need to do the same for a package called 'tightvncserver'. Both of these tools are used for remote administration, so you won't need them if you plan to keep a keyboard, mouse and screen attached to your various nodes.

To connect to each machine from a master device running Windows on the network or one of the Linux machines, you need to use an SSH client. The most popular application for Windows is the freely available Putty, while Linux users can simply type 'ssh -l username IP Address' into the command-line.

To find the IP address of each node, either use your router's web interface for connected devices (if it has one) or right-click on the Network icon on the Ubuntu desktop and select 'Connection Information'. If you're using the command line, type 'ipconfig'.

From an SSH connection, you can launch a remote desktop session by typing 'vncserver :1' followed by a password. When the remote session is active, you can connect to a desktop session on each machine using a VNC client such as TightVNC Viewer for Windows or Vinagre for Ubuntu.

When specifying the address to connect to, make sure that you follow the IP numbers with ':1', as this is needed in order to specify which screen session to connect to.

Testing the cluster

Now that everything is running as it should, let's test the combined power of the cluster.

The easiest method that we've found (and the one that provides immediate results) is to use Blender, a professional-quality open-source 3D animation system that's particularly mathematically intensive. It's designed to work with more than one instance running on different machines.

You can easily run a job on your server before farming it out to the entire cluster. All you need to do is open an instance of Blender on each machine, get them to read the same '.blend' animation file, switch to the Output page and make sure that every machine is writing its output to the same directory.

You then need to enable both the 'Touch' and 'No Overwrite' options. When rendering is started, each Blender instance will grab the next available frame that hasn't been created by another. It's an effective and pretty quick way of spreading the load.

There are two downsides to this option. Firstly, you'll need to have a screen connected to each node because Blender won't run over VNC. Secondly, Blender will only work with animations. Single-frame images can't be distributed. Alternative solutions are more complicated, but happily, both of these problems can be overcome by installing a render farm queue manager.

The best tool that we found was Distriblend. This suite of free Java tools automatically distributes Blender jobs between the various nodes of your cluster. You use a client to add Blender jobs to a distributor that then sends the jobs on to each node. We used Distriblend for our distributed rendering benchmarks, and we were able to almost quadruple our processing speed across our four machines.

The render time for a 120-frame, 640x480 resolution character animation example from Blender Nation was cut from two minutes to less than 30 seconds. On a more complex fluid dynamics job ('Vector blurred fluid' by Matt Ebb), we saved ourselves over two hours by distributing the 50 frames of animation across our cluster. And that's just the beginning.

If you love 3D graphics, this soon feels like the only way to render. The beauty of it is that your main machine remains unburdened while your cluster does all the hard work.