

Lazy Linux: 11 secrets for lazy cluster admins

Visit the inner sanctum of lazy Linux admins and discover how to reduce effort, regardless of number of nodes

Skill Level: Intermediate

[Vallard Benincosa \(vallard@us.ibm.com\)](mailto:vallard@us.ibm.com)

Linux Cluster IT Specialist
IBM

[Egan Ford \(egan@us.ibm.com\)](mailto:egan@us.ibm.com)

Linux Cluster Executive IT Specialist
IBM

22 Oct 2008

Cluster means different things to different people. In the context of this article, cluster is best defined as *scale-out* -- scale-out clusters generally have a lot of the same type of components like Web farms, render farms, and high performance computing (HPC) systems. Administrators will tell you that with scale-out clusters any change, no matter how small, must be repeated up to hundreds of thousands of times; the laziest of admins have mastered techniques of scale-out management so that regardless of the number of nodes, the effort is the same. In this article, the authors peer into the minds of the laziest Linux® admins on Earth and divulge their secrets.

Since their first appearance in 1998 in the list of the Top 500 fastest computers in the world, Linux clusters have risen from an obscure science experiment to the position of today's dominant force in supercomputing technology. In fact, the number of Linux clusters in the Top 500 list has grown from 1 system in 1998 (1 cluster, 1 Linux OS system) to four-fifths of the list in 2008 (400 clusters, 458 Linux OS systems).

Managing Linux clusters requires a unique set of skills that are not usually found among the single-system or small-networked-systems IT administrators -- it requires an in-depth knowledge of networking, operating systems, and pretty much all

subsystems in the architecture.

But this is not all: It requires a different attitude. It requires laziness above all else. It requires the admin to do what Scrooge McDuck told his nephews in Duckburg: "Work smarter, not harder."

In this article we discuss some of the best secrets of the laziest Linux cluster admins. While they can hardly be called secrets, for some reason people either do not understand or they underestimate the power of these ideas. To clear this matter up, we'll present the secrets here with an explanation of their importance.

The secrets are

1. Don't reinvent the wheel.
2. Use open source.
3. Automate everything.
4. Design for scale - plan to be lazy from the start.
5. Design for hardware manageability.
6. Use good cluster management software - don't dig wells with teaspoons.
7. Use open source monitoring solutions.
8. Control your users with a queuing system.
9. Verify you get what you pay for - benchmark it!
10. Manage cluster communication.
11. Look for ways to become lazier.

1. Don't reinvent the wheel

The lazy Linux cluster admin is not in the business of wheel-making; he focuses on building upon the work of others. There is no sense in wasting time building an application when a free supported solution already exists.

One of the rarest things in the world is an original idea or an original problem -- especially in the Linux cluster world. You will rarely come across something that has not been struggled over and solved back in 2004. This shouldn't make you feel unimportant or unoriginal; rather you should feel confident that there is really no

problem that can not be solved (speaking technically, not politically or socially). So accept the fact that most problems and their solutions have been recognized, diagnosed, and solved.

To waste less time, the efficient admin spends more time:

- Researching solutions that exist and adapting them to his own needs. Isaac Newton quoted Bernard of Chartres when he noted that in his work he stood "on the shoulders of giants." Think what would have been lost if he did not first try to understand Euclid's Elements first and then build off of others.
- Contributing or customizing open source projects rather than reinventing something that has already been done - knowing full well that if he writes something, it will more than likely fizzle out when he leaves to a new job because no one else will understand it.

We don't mean to stifle your creativity -- quite the opposite. Using work already made by others allows you to build the next layer that will make your environment far superior and efficient than those in other organizations.

2. Use open source

The most successful Linux cluster admins that we have worked with have a vast knowledge of current open source projects. They are the frequent contributors to mailing lists and their names are the ones associated with current projects when you search on the Internet. They usually troll <http://sourceforge.net> and <http://freshmeat.net> for new projects that interest them.

Open source tools have properties that allow them to live longer than the person championing them, especially the popular ones. There is a very good reason that tools like Ganglia, Nagios, and TORQUE are still in use even though they have been around a long time. They are good -- they save administration in software costs and licensing schemes.

One other aspect of the laziest cluster admins is that they are quite passionate about open source and use it in their own personal pursuits. This may be their own Web servers at home or applications that they run on their own Linux notebook. From Pidgin to Firefox, you'll find that the laziest Linux admins run Linux in some other aspect of their lives distinct from the clusters they manage at work.

3. Automate everything

Using scripts on the command line and other quick writes are a big part of the Linux

admin's tool chest. Scripting (as long as it doesn't reinvent anything) provides two useful results:

- The first and most obvious is that it saves typing and provides a repeatable pattern.
- The second is that it is self documenting and can be referred back to later.

We quite commonly see skilled admins with directories for scripts they have written on their machines. These scripts do everything from checking the firmware versions on the nodes to mapping GUIDs in an InfiniBand cluster.

One example where scripting is quite appropriate is that of generating an operating system image, whether it be stateless or stateful. If an admin has a "golden image" that needs to be propagated to each compute node on the system, he should know what is in it. Having a script that creates that image is the best documentation available because it explains exactly what is done and it is repeatable. Without scripts to build images, image bloat occurs and that eats up more space and slows the system down.

Too often we run across organizations with a golden image that they have been nurturing since 2000. The biggest reason: They don't know how to rebuild it. The second and probably the best reason: Because their application has been tested and "certified" on this image. *Certified* is one of those terms you run across that is as nebulous as the definition of *cloud computing* (which by the way is not a patented nor trademarked term).

The secret of why you should want to automate things is this: It takes more brain power to get out of work than it does to actually do work. The lazy Linux cluster admin does not accept work that turns his brain into fluff. If you have to ssh into every machine in the cluster and run a command, you are not being lazy enough. All commands to nodes should be done in one fell swoop using parallel commands or procedures. If your hardware vendor does not have Linux tools to automate BIOS updates or subsystem flashing, you should factor that into your acquisition cost.

Tips 8 and 10 in our last article on "[Lazy Linux: 10 essential tricks for admins](#)" documented several command-line scripting techniques we use often. There are many other ways to do it, some of which may be more efficient, but those tips just give you an idea of what can be done.

4. Design for scale and plan to be lazy from the start

Secret Number 3 (automate, automate, automate) is a great goal, but it is just a step on the path to *complete idleness*. For the laziest of admins, complete idleness can

only be achieved with autonomous scale-out management. The first step in this quest is a system immune to operations that do not scale.

Very large scale-out clusters are plagued with bottlenecks. That is, most scale-out admins use TFTP to network boot or install large sets of machines. As any experienced scale-out-oriented admin can tell you, TFTP is unreliable and does not scale. Without proper remote hardware control, a massive TFTP failure could require the lazy admin to actually get out of his chair (bed) and walk (bum a ride) to the data center (not home) to reset each and every machine (busywork)! Even with proper remote hardware control, the lazy admin will have to stop playing *WoW* long enough to periodically issue the commands (again, busywork) to reset the nodes at a smaller scale to get the system up.

With a little upfront planning management, bottlenecks (like the following) can be avoided.

Bottleneck #1: Provisioning services

DHCP, TFTP, HTTP, NFS, and DNS are the most common services used to provision clusters. Each of them has a threshold -- TFTP is the worst one when it comes to scaling. Fortunately all of them can be easily replicated to assist with scale.

Tip: Isolating DHCP and TFTP to a different NIC will dramatically increase scalability. For instance, we have measured TFTP scaling at 40:1 if sharing the NIC with other provisioning services; 80:1 if not sharing services or stateless booting.

Bottleneck #2: The network

The network it is often the most overlooked part of any design. We are referring to the GigE network used for management, not specialized high performance networks used for application traffic. Although in many cases there is only one network that must be shared for data and management; this can compound any scaling issues.

Be careful when designing hierarchal networks that you do not oversubscribe too much. If an 80:1 node-to-service-node ratio is required, make sure that ratio is maintained or exceeded throughout the fabric.

Bottleneck #3: Don't bite off more than you can chew

When we design large scale-out clusters, we take a cluster-of-clusters approach. Each sub-cluster or scalable-unit (SU) is a building block that scales within itself for all cluster operations, (for example, install, network boot, BIOS flashing, monitoring, etc.). Each SU has a one or more (depending on the size of the SU) service nodes to provide the services necessary to control, monitor, and provision all the nodes in the SU. To further aid in scalable management, each SU has its own broadcast domain (SU-to-SU and SU-to-World communications are routed -- check for

bottlenecks).

The central management node and service nodes have a private physical or virtual network so that aggregation of information from the service nodes and data pushed to the service nodes do not compete with other cluster traffic. We refer to this network, the management node, and the service nodes as the *hierarchical management cloud* or HMC. Its setup and operation is solely in the domain of the admins.

This cluster-of-clusters approach will allow the lazy admin to design systems that can scale beyond any budget and allow the same admin central control without fear that massive operations will fail.

5. Design for hardware manageability

We are surprised at the number of admins who don't think in terms of "lights out" when designing their clusters. Efficient administrators operate lights-out clusters, meaning their clusters sit in dark rooms away from humans and ideally it would be weeks or months before they actually need to see the physical machines they work on daily. In some cases, they'll never see the machines because they are managing them from the other side of the world. Of course, the laziest don't even know where the data center is -- it is just a set of host names or IP addresses.

Data centers are loud and sometimes cold (and who knows, maybe dangerous); the lazy admin should avoid them at all costs. Who knows the as-yet undiscovered health hazards of being in rooms with lots of machines. As power/cooling/staffing costs rise, the trends are towards moving data centers to places that are less expensive to operated. With this in mind, having absolute remote control should be regarded as essential to managing a Linux cluster now and for the foreseeable future.

Hardware vendors have largely given into customer desires for standards to remotely manage systems. IPMI 2.0 has become the current standard for most managed Linux clusters managed. IPMI offers a way to remotely power-cycle a machine, as well as have remote console viewable to see the machine boot from BIOS. At one customer site, we were able to troubleshoot a machine that was 60 miles away from us in the comfort of the customer's office. (The customer was one of those lazy Linux admins whose office was only lit by the neon signs on his wall. His office-turned-bachelor-pad was also equipped with two fridges laden with energy drinks and sugar-coated snacks. Needless to say, we didn't want to leave the office.)

IPMI is powerful -- we could change BIOS settings, reboot the nodes, and watch them boot up and see the screen dump without ever seeing the machine - it should be set up on all clusters. You should always demand at the least:

- Remote powering of the machines and
- Remote console or better viewing of the machines to see boot up problems that may arise.

With IPMI, we see little need in the Linux cluster space for other boxes that merely provide us a glorified interface to run IPMI other than perhaps a management node. Instead, we recommend standard open source tools like ipmitool that come packaged with most Linux distributions already. We find that our laziest Linux cluster admins will live and die by the command line.

What is still open to debate is the dependability of IPMI for remote console. We recognize that there are times when a real out-of-band terminal server can be of value. Terminal servers such as the Cyclades ACS48 remain a reasonable investment and provide out-of-band access and reliability that IPMI does not quite deliver.

In addition, IPMI 1.5 was not the most reliable IOHO (and it was an industry-wide issue). IPMI 2.0 does a much better job and many vendors add fancy Web pages around it to make it seem like it's out-of-band enough. There are arguments to include and not include terminal servers and just use IPMI native on the machine. Most of our customer's trains of thought chug along like this: *Every lazy Linux admin knows that he spends a great deal of time troubleshooting 5 percent of the nodes when 95 percent of them are doing just fine. In a case like this, perhaps it is better to just buy 5 percent more nodes instead of infrastructure and have spares. This then means the budget is spent more on compute power than on infrastructure.*

The argument that counters this one is that if one terminal server can save a plane trip across the country to troubleshoot a node, then the expense is worth it. We let the lazy Linux admin make the call -- after all, it's him that has to get in the plane. We've seen strong opinions on both sides.

6. Good cluster management software keeps you from digging wells with teaspoons

Cluster tools have come a long way since we first started installing Linux clusters in 1999. Back then, there were not many cluster management tools available and as such, most admins created home-grown suites of tools that deployed, monitored, and managed their clusters.

The laziest admins have either adopted open source tools or made the tools they developed back in 1999 available to the community. Seldom does anyone have an environment so unique that open source tools can not fill this gap. Most often, those who champion their own tools usually are alone and when they leave an organization, their tools disappear. However, we do recognize that there are many

sites where customized tools work just fine.

If you are not satisfied with your home-grown tools or are looking for something better, consider looking at several open source tools. Among the most prevalent for managing clusters are OSCAR (System Imager), ROCKS, Perceus, and our personal favorite xCAT 2. All of which are open source.

Perhaps the most popular open source cluster deployment/control solution today is ROCKS. ROCKS was created and maintained by UCSD and they have done a good job of making clusters user friendly. Our only gripe is its lack of flexibility, primarily at the OS level. ROCKS is based on Red Hat distributions which is fine for many people, but not those who use SUSE or wish to use images they have created based on RH 6.2 distributions. In addition, ROCKS is not a cloning solution which we find many IT organizations using.

Perceus is another such solution that differs from ROCKS in that it is a stateless installation. For the purposes of this article, we define stateless computing as running your operating system in memory instead of keeping it on disk. A disk is not required but can be used for local scratch or for swap.

What we like about xCAT other than our vested interest (full disclosure; we contribute code and actively develop xCAT) is that it has more flexibility, scales more, and has more power than any of the other tools. (And it has the most handsome and intelligent contributors.) The fastest supercomputer on earth, the LANL RoadRunner system (the Cell/B.E.[™]/Opteron[™] hybrid that is the first one-petaflops system *and* first one-petaflops Linux cluster) is managed by xCAT.

xCAT allows imaging, kickstart, autoyast, iscsi, and stateless for nearly every enterprise Linux distribution available. In addition, it has command-line tools that abstract IPMI commands from remote power to console setup and uses them in the same framework. xCAT has been actively developed since October 31, 1999 and was open sourced by IBM in 2007.

But to be fair, we'll mention the drawbacks to xCAT as well:

- It can be difficult to set up; with so much flexibility there are a lot of configurable parameters. We have set up a wiki to make this easier and there is lots of support via mailing lists and IRC channels.
- xCAT 2 is a complete rewrite of the venerable xCAT 1.3 and as such, there are features that need to be ironed out.
- Like many open source tools, the documentation could be improved.

There are many other cluster management solutions and we could easily have the *Emacs vs. vi* debate on the topic. But we'll just end with this: Give xCAT a try if you are looking for ideas or something better.

7. Use open source monitoring solutions

Last year at SC'07 we met with several top labs in the US to discuss the difficult problem of monitoring a Linux cluster. That led to a few calls in early 2008 to discuss this problem. Monitoring is difficult for several reasons:

- As the cluster gets larger, the data collected is larger and performance can be felt on the machine receiving monitored information. For example, if there are 2,000 machines and each one is sending metrics to an infrastructure node, the node can be brought to its knees and leave you wondering if it's responding.
- Collecting data with agents on compute nodes can suck memory and CPU cycles from the user's jobs. Many shops want no agents on the nodes because it lowers performance of applications. Striking the balance requires trade-offs: Is it worth the CPU cycles taken to get the data?
- We have not seen a scaleable tool that ties user jobs with machine performance to profile jobs in a way that is usable and visually appealing.
- There are no tools that do it all exactly as you would like it to be done. This is what is perhaps most striking about this space: The tools used are incomplete and miss doing at least one something everyone would want. Most admins use a combination of one or two management products. When we see products that promise to monitor your cluster that surpass all other solutions we are skeptical: Every cluster is different and no one size fits all. Customized monitoring seems to be the only way out of it.

So given the complexity, here's how some of the laziest administrators we know are solving the problem.

The most common solution we've noticed at large cluster shops (including top universities and government labs) was to use Nagios for alerting and Ganglia for monitoring. Between these two very customizable tools an admin can get great insight into the multitudes of things happening on the cluster. Ganglia has proven to scale extremely well.

But there are other points of view as well. At USC, Garrick Staples wrote pbstop as a plug-in to TORQUE to visually see what each job is doing and where it is running. He says this is all the monitoring that he needs and doesn't use anything else.

The most popular open source monitoring tools we have seen used by the scale-out cluster community are:

- Nagios.

- Ganglia.
- Cacti.
- Zenoss.
- CluMon.

We can say that many of these tools in turn make grand use of RRDtool in their implementation. CluMon also uses Performance Co-Pilot (PCP) underneath which is also quite popular. xCAT will have support for Ganglia and PCP in a future release.

To recap, the lazy Linux cluster admin knows

- Monitoring doesn't have a one-size-fits-all solution. Monitoring means different things to different people. Some people are interested only in machine alerting, others in performance metrics, others in job profiling, and others just want to know if certain services or applications are running.
- Customization will be different at different sites and on different clusters within the same sites.
- The trade-offs between what is being monitored, what it is used for, and what resources it requires needs to be weighed carefully.

8. Control/manage users with a queuing scheduler

The lazy admin knows that users are the root of all problems. Preventing users from having root powers or permissions therefore is extremely important. You can even go further than that: You should do everything you can to keep users off your machine.

Queuing systems provide this functionality: A user submits a job and the queuing system decides which nodes it will run on. Unless users are running a job, they should stay off of the machine.

Today's popular queuing systems include some pay for products such as: LSF and PBS Pro. These products are used by many commercial customers, as well as government labs and universities. For many systems, plain open source solutions like TORQUE and SLURM work just fine.

We do a trick with TORQUE coupled with the Maui scheduler to keep our users off the cluster unless they are running a job. In Linux, this is done by first setting the `/etc/security/access.conf` file so that only root can login and no one else is allowed. For example, on each node if you run the command:

```
echo "-:ALL EXCEPT root:ALL" >>/etc/security/access.conf
```

then only root will be able to log into this machine. Next, you create a TORQUE prologue script that allows the user to login that runs something like this:

```
perl -pi -e "s/:ALL$/ $USER:ALL/" /etc/security/access.conf
```

(Hint: The `$USER` variable is the second variable that is passed to the prologue script by TORQUE when the script runs.) Since the root user runs the prologue script, the user will be allowed to log into the cluster. When the job is completed, an epilogue script is run that removes the user from `/etc/security/access.conf` so that he can no longer log into the node `perl -pi -e "s/ $USER\b//g" /etc/security/access.conf`. This prevents users from clobbering each other.

We have seen "performance" problems on clusters that have nothing to do with the machine; the real problem is that multiple users run on the same machine and the jobs they run each require 100 percent of the CPU.

It's no secret that user management is imperative. But what is often overlooked in simple troubleshooting is that users themselves are creating the problem. We strongly suggest that users be kept off the system unless they enter in through a controlled environment like a resource scheduler. In addition, we urge that the cluster network itself (the gigabit management or user network) be separate from the rest of the corporate or campus WAN with only certain user nodes providing front-in access.

9. Benchmark! Find performance problems before they find you

The last thing you want to deal with is a torch-bearing mob threatening to burn down your village because performance is weak and results are incorrect. So keep in mind that all too often, hardware diagnostics are the only litmus tests when determining cluster worthiness, but hardware diagnostics may paint an incomplete picture.

Hardware diagnostics are usually pass/fail with a vendor-defined threshold -- your threshold may be higher or lower. If a hardware diagnostic test fails, then you do have a problem; however no failures does not mean there are no problems.

Here are some problems we've encountered that had a measurable impact on performance on with systems that passed vendor diagnostics:

- Double-bit memory errors.
- Single-bit memory errors.

- SRAM parity errors.
- PCI bus errors.
- Numerical errors.
- Cache errors.
- Disk errors.
- Performance inconsistencies.

Often there are problems that have nothing to do with hardware, but with software instead. Applications, libraries, compilers, firmware, and any part of the operating system can be the source many problems undetected by hardware diagnostics. Hardware diagnostics often do not run in the same runtime environment as the applications and do not stress the subsystems the same way as applications -- then problems created by software will be missed.

Clearly you need to run some type of relevant workload with your operating environment to verify that your cluster actually can do good work. This can be accomplished by running a few industry-accepted benchmarks. The purpose of benchmarking is not to get the best results but to get consistent, repeatable, accurate results that are also the best results.

How do you know if the results are the best results? A cluster can be broken down into the following major subsystems:

- Memory.
- CPU.
- Disk.
- Network.

Your hardware vendor should have benchmark data stating the expected Memory, CPU (FLOPS), Disk, and Network performance.

How do you know if the results are consistent?

Statistics. Each benchmark is run one or more times per node (or set of nodes for multi-node tests) and then the best representative of each node (or set of nodes) is grouped together and analyzed as a single population. The results are not as interesting as the shape of the distribution of the results. Empirical evidence for all the benchmarks in this article suggests that they should all form a *normal distribution*. A normal distribution is the classic bell curve that appears so frequently in statistics. It is the sum of smaller, independent (may be unobservable), identically-distributed variables or random events.

Benchmarks also have many small independent (may be unobservable) identically-distributed variables that may affect performance, such as:

- Small competing processes.
- Context switching.
- Hardware interrupts.
- Software interrupts.
- Memory management.
- Process/thread scheduling.
- Cosmic rays.

These variable may be unavoidable, but they are a part of the source of a normal distribution.

Benchmarks may also have *non*-identically-distributed observable variables that may affect performance:

- Large competing processes.
- Memory configuration.
- BIOS version and settings.
- Processor speed.
- Operating system.
- Kernel type (like NUMA vs SMP vs UNI) and version.
- Bad memory (such as excessive ECCs).
- Chipset revisions.
- Hyperthreading or SMT.
- Non-uniform competing processes (such as `httpd` running on some nodes but not others).
- Shared library versions.

These variables are avoidable. Avoidable inconsistencies may lead to multimodal or non-normal distributions and may have a measurable impact on application performance.

With a goal of *consistent, repeatable, accurate results*, it is best to start with as few variables as possible. Start with single node benchmarks like STREAM. If all

machines have similar STREAM results, then memory can be ruled out as a factor with other benchmark anomalies. Next, work your way up to processor and disk benchmarks, then two-node (parallel) benchmarks, then multi-node (parallel) benchmarks. After each more complicated benchmark, run a check for consistent, repeatable, accurate results before continuing.

In the outline following is a path we recommend (the benchmarks report the performance of the components in **bold**).

- Single-node (serial) benchmarks:
 1. STREAM (**memory MBps**)
 2. NPB Serial (**uni-processor FLOPS** and memory)
 3. NPB OpenMP (**multi-processor FLOPS** and memory)
 4. HPL MPI Shared Memory (**multi-processor FLOPS** and memory)
 5. IOzone (**disk MBps**, memory, and processor)
- Parallel benchmarks (for MPI systems only):
 1. Ping-Pong (**interconnect microsec** and **MBps**)
 2. NAS Parallel (**multi-node FLOPS**, memory, and interconnect)
 3. HPL MPI Distributed Memory (**multi-node FLOPS**, memory, and interconnect)

Wait, that sounds like a lot of work!

It is, but it is also necessary if you plan to be lazy later on. Fortunately we have the tools and documentation to make this easy. A few days upfront planning can save weeks of frustration later on. We'll talk about these tools and graphs in a future article; they will also publish as part of a future xCAT RPM that will greatly increase productivity.

10. Manage cluster admin communication

Setting up MediaWiki

To set it up, first designate a server to host the Wiki. We usually use the management server if none other is available:

```
ssh mgmt
```

Now make sure an http server, a mysql server, and php5 are installed. If you were using Red Hat 5.1 or its derivatives you would type:

```
yum install httpd mysql php5 mysql-server
Next, configure mysql:
service mysqld start
mysql_install_db
mysqladmin -u root password 'mypasswd'
Now install media wiki:
cd /tmp/
wget
http://download.wikimedia.org/mediawiki/1.13/mediawiki-1.13.0.tar.gz
tar zxvf mediawiki-1.13.0.tar.gz
mv mediawiki-1.13.0 /var/www/html/wiki
chmod a+x /var/www/html/wiki/config
Once you make it this far, navigate your Web browser to
http://localhost/wiki. You can just follow the menus to install the rest
of it.
Once you finish, it will instruct you to do this:
cd /var/www/html/wiki
mv config/LocalSettings.php .
```

Once you collect information about a system, the information should be stored some place useful so that the rest of the cluster staff can access it easily. We'd like to welcome you to the year 2000 -- documents in Word or Excel are neither cool nor is that the way to do this efficiently. The most productive practice for this we have seen is to set up an internal wiki. This is because the lazy admin gets tired of answering the same questions over and over again. Instead of having to look it up or run some command to give an answer, he simply says: "Check the wiki." And his task is done.

Every site should maintain a wiki that contains all the information about the cluster that is generally asked for like:

- A log of changes to system including when administrative action was performed on the cluster.
- Inventory of cluster: Firmware versions, models, serial numbers, number of nodes in cluster, processor type, memory.
- Links to open support tickets with the vendor and where to get updates.
- Documentation for common tasks using the management software.
- Information on how OS images are created.

In short: The wiki should have enough information so that if someone asks a question about the cluster the admin need only say "Check the Wiki." In turn, any time you give someone an answer to something that is not in the wiki, you should tell them they need to pay it forward and right down that knowledge you gave them in the wiki. In addition, it is good for the churn in personal that is the reality of the wild IT world.

Why's a wiki better than other forms of documentation?

- Wiki's are editable at a central place and access control can be given to people who need it. We've seen documents that reside in a share repository, but to view them, one must first navigate to this repository, then save to their hard drive and then open. Ineffective. One link is faster and less frustrating to get to.
- Writing information in a Word document or spreadsheet is static and is not nearly as easy to edit, resave, and send out revisions. Not to mention, we see too many iterations of the same document around and people don't realize if they have the latest version or not. Especially when more than one person is editing it. In effect: Living documents live longer on wikis.

Setting up a wiki is extremely easy. We use MediaWiki. It's free, easy to get, and easy to install and configure. (See sidebar.)

Wiki syntax is much easier than HTML and there are many useful links on the Web that show how to use it. There are also good extensions that you can get for highlighting code syntax in perl or bash if that is what you use.

We find little resistance in organizations when we propose a wiki and hope it makes you lazier by installing one.

11. Look for ways to be lazier

We frequently see people in the cluster business doing things the way they do it because that's the way they've always done it. We think this is a good way for a Linux cluster shop to lose talent and get the least work out of their cluster. Change is the name of the game and new ideas come along frequently.

Naturally, we don't expect anyone to be able to investigate every idea that comes their way, but being familiar with newer trends is something that sets apart the good admins from the mediocre ones. Having said that, in this fast moving space, no one can possibly know something about everything and very few know everything about something. But good cluster admins know some things pretty well, have tested even more things, and ask questions about things they haven't heard about.

So if someone starts talking about something you haven't heard, the lazy Linux admin will ask a question because he's too lazy to go off afterwards and use his favorite search engine to find information about it. The scariest Linux cluster admins are the types that never ask questions. The lazy Linux cluster admin is not afraid of saying he doesn't know something. He is confident in his skill that if he doesn't know it, then someone else in the room doesn't either.

Today, there are great things happening to the world of managing Linux clusters. The most interesting that we have seen are:

1. More of a shift to stateless computing. Makes it easier to administer images and easier to keep nodes in synch.
2. Looking at clusters from the data center perspective. This means taking into account power and cooling and the payoff in terms of three or more years as opposed to just acquisition cost.
3. Liquid cooling efficiencies as opposed to air cooling. Did you know that liquid cooling is 90 percent more efficient than air cooling?
4. Adaptive management. This is where the queuing system is able to provision nodes on demand. This is true cloud computing and we have demonstrated this with Moab and xCAT. This is the final step of the path to *complete idleness*.

Summary

If this article has done its job, then you should now have ideas for how you can do less work and get better control of your existing Linux cluster environment and plan for your next one. We are confident that the ideas and practices we have set forth in this article contribute to better cluster utilization, a more professional science around it, and leaner and more efficient cluster administrative staff.

Having less people and less to problems means less meetings, less work, and more time for *WoW*, herding goats, sleeping, or doing whatever your lazy pursuits may be.

Resources

Learn

- Tips 8 and 10 from the article "[Lazy Linux: 10 essential tricks for admins](#)" (developerWorks, July 2008) documents several command-line scripting techniques we like to use often.
- Some of the resources we mentioned in this article:
 - [Top500 Supercomputer sites list](#) has been listing the world's fastest computers since 1993.
 - I believe we mentioned the [Supercomputing 2007 conference](#). (There's at least one of these gems each year.)
- In the [developerWorks Linux zone](#), find more resources for Linux developers (including developers who are [new to Linux](#)), and scan our [most popular articles and tutorials](#).
- See all [Linux tips](#) and [Linux tutorials](#) on developerWorks.
- Stay current with [developerWorks technical events and Webcasts](#).

Get products and technologies

- Here's where to get some of the resources we mentioned in this article:
 - [Rocks](#) is an open-source Linux cluster distribution that enables end users to easily build computational clusters, grid endpoints, and visualization tiled-display walls. The group has one goal: Make clusters easy.
 - [OSCAR](#) (Open Source Cluster Application Resources) lets users, regardless of their experience level with a *NIX environment, install a Beowulf-type HPC cluster.
 - [Perceus](#) is the next generation of enterprise and cluster provisioning toolkit created by the developers of Warewulf.
 - [xCAT](#) (eXtreme Cluster Administration Toolkit) is a scalable distributed computing management and provisioning tool that provides a unified interface for hardware control, discovery, and OS diskful/diskfree deployment.
 - [Nagios](#) is a host and service monitor designed to inform you of network problems before your clients, users, or managers do -- it was designed for Linux but works fine under most *NIX variants.
 - [Ganglia](#) is a scalable distributed monitoring system for HPC clusters and grids that is based on a hierarchical design targeted at federations of clusters.

- [pbstop](#) is an ncurses monitoring and admin tool distributed inside perl-PBS (Portable Batch System) and is used by the admins of the largest clusters in the world.
 - [TORQUE](#) is an open source resource manager providing control over batch jobs and distributed compute nodes, a community effort based on the original *PBS project.
 - [Cacti](#) is a complete network graphing solution designed to harness the power of RRDTool's data storage and graphing functionality; it includes a fast poller, advanced graph templating, multiple data acquisition methods, and user management features out of the box.
 - [Zenoss](#) delivers commercial open source IT management solutions.
 - [CluMon](#) is a cluster monitoring system developed at the National Center for Supercomputing Applications to keep track of its Linux super-clusters; it is a tunable system that can be made to work for almost any set of Linux machines.
 - [RRDtool](#) is an open source, high performance data logging and graphing system for time series data.
 - [Performance Co-Pilot](#) provides a framework and services to support system-level performance monitoring and performance management; SGI provided an open source version in 2000.
 - [SLURM](#), the "highly scalable resource manager" (and a popular worm-based soft drink on the cartoon *Futurama*), is an open-source resource manager designed for Linux clusters of all sizes that provides three key functions -- it allocates exclusive and/or non-exclusive access to resources; it provides a framework for starting, executing, and monitoring work on a set of allocated nodes; and it arbitrates contention for resources by managing a queue of pending work.
 - [Moab Workload Manager](#) is a policy-based job scheduler and event engine that enables utility-based computing for clusters.
- [Order the SEK for Linux](#), a two-DVD set containing the latest IBM trial software for Linux from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.
 - With [IBM trial software](#), available for download directly from developerWorks, build your next development project on Linux.

Discuss

- Get involved in the [developerWorks community](#) through blogs, forums, podcasts, and spaces.

About the authors

Vallard Benincosa

Vallard Benincosa has been building Linux clusters since 2001. According to T. Adams, [Vallard doesn't really know anything](#). According to Dirck, [Vallard is overpaid](#). According to his wife, Vallard needs to spend more time at home and is underpaid. According to Vallard, he's just likes to surf, play guitar, and have fun with his two kids.

Egan Ford

Egan Ford started building Web and HPC Linux clusters in 1999 and was the chief architect for IBM's first large HPC cluster (Los Lobos at the University of New Mexico). Since then Egan has led in the design and implementation of some of IBM's largest systems including AIST, LANL Roadrunner, and the US National Science Foundation Teragrid (teragrid.org). Egan is the creator of IBM's first cluster management solution (xCAT) and co-author of two Linux HPC Redbooks.