Sine Nomine Associates
October 2000

LINUX for S/390: Technical Notes on a Large-Scale Implementation

Contents

Executive Summary

This white paper presents technical commentary on deploying a large-scale enterprise and service provider solutions for UNIX-based Internet and ISP services in a US telecommunications provider. The technical commentary describes the implementation and testing of the much-bandied "41,400" image test and details some of the technical lessons learned and opportunities for future research in this area.

Introduction

This paper provides some technical notes on the design and deployment of the 41,400 LINUX® image test performed as a demonstration to illustrate the viability of Linux for S/390® as a large-scale ISP hosting and utility platform for a major US telecommunications company. The test was generated in January, 2000 as a verification of the appropriate use of an S/390 platform to deliver ISP infrastructure services as part of a completed managed router service offering.

The test was designed and executed in several phases (labeled Test Plan Alpha, Baker, and Charlie). The phases were constructed of identical components and varied only in scale (number of virtual Linux images generated in the configuration and loading parameters used to artificially impair performance of individual images). To best illustrate this, we concentrate in this paper on the largest of the three tests, Test Plan Charlie, which demonstrated the capability of one physical S/390 processor to support many virtual servers, each performing useful work.

To understand the background of the test, a basic knowledge of ISP and Internet data center (IDC) functions may be helpful to illustrate some of the requirements. In most common ISP and IDC deployments, the requirements are quite similar; commonly characterized as a three part set:

A) *A basic set of applications drawn from the open source community that deliver utility services such as WWW service, DNS, Usenet News, etc, Regardless of hardware platform, most ISPs and IDCs use the same application source to deliver these basic Internet utility functions (the UC Berkeley*

bind code to implement DNS, University of Toronto's INN for Usenet News, etc). These tools are considered standard usage for any ISP, and most UNIX-savvy systems administrators are familiar with configuration and setup of these tools. As the industry supports additional applications (such as time-sharing access to commercial applications common in the application service provider (ASP), the demand for commercial applications will increase, However, these basic services remain as a prerequisite for the correct operation of the commercial applications.

B) *A primarily UNIX-based environment due to the common requirement for remote management and lights out operation that is difficult (if not impossible) to provide in other environments.*

While some ISPs provide Windows-based services and applications, the cost of operation is much higher due to the design expectation of a Windows application that a human has access to from console connected to the server to support a GUI-based configuration tool. In the ISP environment, this is not practical due to the rack real-estate required, thus a remote management tool such as Carbon Copy or PC:Anywhere is required at a significant cost per server. The UNIX environment generally assumes only a serial terminal based console, or an X Window system-based GUI which directly supports remote network-based access for configuration purposes.

C) *A primarily TCP/IP based transmission environment.*

As the ISPs are normally focused on providing access to the global Internet, the primary network protocol is TCP/IP. Most ISPs and IDCs support other LAN-based protocols such as Novell's IPX or the NetBIOS protocols commonly used with OS/2® or Windows NT® for use with collocated servers, however the primary focus remains on TCP/IP based applications.

Beyond these characteristics, the design of ISP utility applications tends to be in a model that supports a horizontal scalability, ie, scalability is derived through deploying multiple servers running single copies of the application sharing a common data source, rather than the traditional multiuser system model of a small number of systems with diverse workloads.

Business-oriented ISPs are also generally required to deliver contractual service level agreements based on application and network performance. As neither Windows or most UNIX-based solutions provide the type of user-level resource controls common in the traditional S/390 environments (thus providing the ability to limit resources used by a specific system Userid), this requirement is normally met by using dedicated server hardware for each application or customer environment, i.e., each application or customer receives a separate system for each instance of the application.

Technical Notes on Test Plan Charlie

Based on these common characteristics, the idea of the test was to demonstrate that the Linux for S/390 port could deliver an acceptable solution that made use of pre-existing skills and staff, provided a significant level of scalability and reliability at lower cost than a discrete system solution, and that it would be perceived by actual customers as a reliable and stable service based on modern technology.

Test Structure

Problem Definition

The constraints of the test required that the test case demonstrate a real-life ISP application operating end-to-end and that the test demonstrate the following elements:

- *Application portability and configuration*
- *Scalability of solution in terms of number of customers and number of systems*
- *System management capabilities (resource constraints, differentiated levels of service, common console management)*
- *Overall technical feasibility and system requirements for deployment.*

To demonstrate these elements, we selected a simple WWW hosting application serving a static HTML page from a common location.

The test plan structure was configured to demonstrate four elements of standard ISP operational environments:

- *Basic IP connectivity and routing functionality.*
- *Shared data access and remote NFS functions.*
- *WWW server delivery of a simple static HTML page containing the words "Hello World".*
- *Error, logfile, and system management functions for a substantial number of servers*

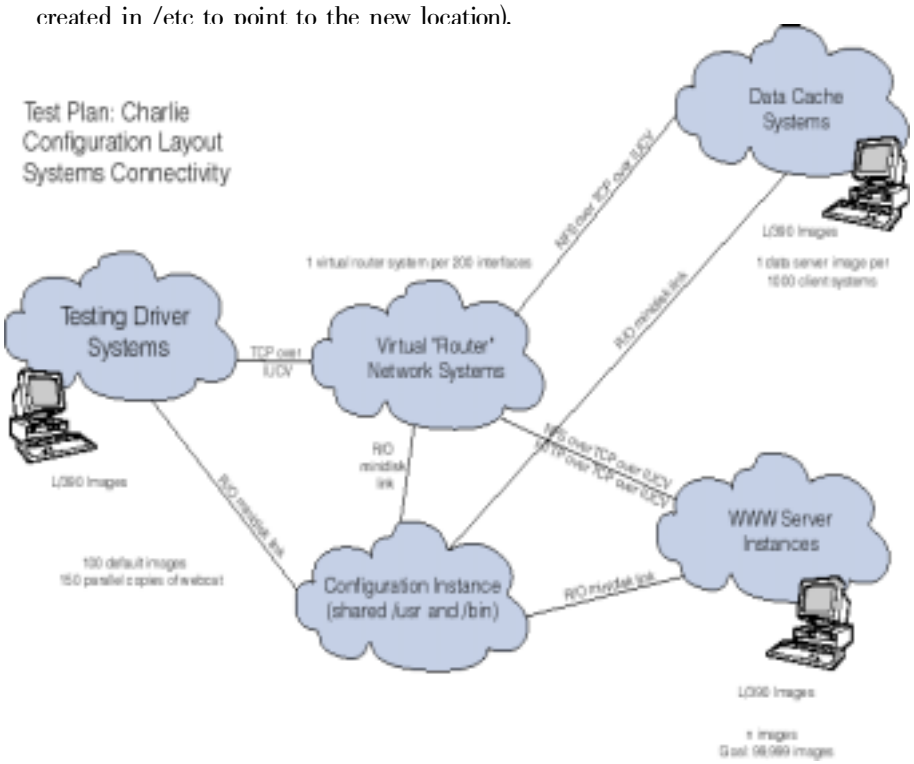
The configuration of the test appeared as in the following diagram. Five types of Linux server instances were created:

- *Virtual routers (Linux instances running Cornell gated for routing support)*
- *Data caches (Linux instances providing NFS file server functions)*
- *WWW servers (Linux instances running Apache to provide WWW access)*
- *Testing drivers (Linux instances running webcat to simulate users accessing WWW servers)*
- *Utility systems (Linux instances providing DNS, etc)*

After downloading the monolithic Marist distribution files, we split each major Linux filesystem tree (/usr, /bin, /lib, /etc, /var, /home) onto separate VM minidisks and configured a master Linux system for maintaining the shared components. We duplicated the master system Userid for each application server type (DNS, WWW server, NFS file server, testing driver system, virtual router), sharing the common elements (/usr, /bin, /lib, /etc) between the instances as R/O minidisks blocked at 4K to allow Linux to take advantage of CP in-core disk caching. This allowed all instances to share the same core code, but be individually customized for different tasks. /var was omitted from the masters as this filesystem is normally local to a particular instance, and /home was created as a mount point for NFS-based data.

Each application server type master was customized to a particular task. If configuration files were required to customize a particular service resided in the R/O portions of the code, the master image was modified to move the re-

quired configuration files to /var and replaced the entry in the R/O files with a symbolic link to /var/share/<original-location of the file> (ie, /etc/nsswitch.conf was moved to /var/share/etc/nsswitch.conf with a symbolic link created in /etc to point to the new location).



Components

The environment available for testing was a 2 CPU, 128M LPAR available on an existing 9672 at the customer site (originally used for Y2K testing). The customer also made a newly purchased (but not yet in production use) EMC DASD storage subsystem available to the LPAR as a dedicated resource with 16 ESCON® connections.

The customer already had VM/ESA® 2.4 available at the site and had installed a copy of VM and the DIRMAINT directory maintenance utility to provide userid creation and management services. The Marist College distribution of version 2.2.13 of Linux for S/390 was used as the base for the Linux virtual machine configurations.

Interconnects

Control and monitoring connections were provided by two dedicated 3270 console terminals available at the site. Interconnections between the Linux instances were originally based on the Linux TCP over IUCV driver, but were later changed to TCP over vCTCA to provide failover capability due to a limitation in the IUCV driver that allowed network link connection to only one other virtual machine per Linux instance (the CTC driver supports multiple CTC devices coupled to multiple virtual machines). For the test, all the virtual systems communicated internally using the 10.x.x.x private IP address space.

For each instance, a Perl script was created to run once during initial IPL (in/etc/rc.local) to configure the server to use a 192.128.x.x address, mount a CMS minidisk containing a configuration file with network and system parameters using CMS NFS server support, read and configure the necessary Linux files, and force a reboot with the real address and connectivity information.

To drive the test, a short Perl script called Webcat was developed. Webcat implemented a simple HTTP client that retrieved the contents of a specified URL, discarded the data, and recorded the wall time required to retrieve the data. Each testing system was configured to run 150 copies of webcat in parallel, repeat the 150 copies every 10 seconds, and initiate loading as soon as the system was available.

The remaining software development generated a set of REXX and CMS Pipeline tools to quickly generate a new server image Userid and its associated configuration file (one exec per server type). These tools were used later in the production system as the basis of a CGI script to support automated provisioning.

Scale/Phasing

The test was conducted in three phases: Able (750 servers), Baker (2750 and 10,000 servers), and Charlie (as many as the configuration would support). The customer staff performed the installation and configuration of the DNS, WWW server and NFS server applications according to the test plan documents to verify that the applications were source compatible and could be configured

correctly by users with generic UNIX experience, but without specific S/390™ experience.

Test plans Able and Baker required approximately 3-5 hours to execute the creation and initialization of the virtual servers and to bring them into the test configuration under general CMS PROP (programmable operator) control. During the test, VM monitor and seeks data was collected to determine the performance characteristics of a multiple server environment.

Following the testing of smaller numbers, Test Plan Charlie (continue creating servers until the system runs out of resources) was started at 5 pm on Friday evening, and had reached 41,400 servers at approximately midnight. At that point, the system refused to create additional virtual machines due to a lack of system resources, but did not crash. The system was paging heavily and using 100% of all available LPAR CPU resources at this point, but response was acceptable (within 2-3 seconds) for a random sampling of test virtual machines.

Transition to Production

The customer has accepted the solution for production use and has purchased an additional G5 10-way system to deploy the solution. Approximately 15 to 30 virtual systems per week are being created as new customers are added to the service. The virtual system deployment is driven by a VM-based WWW server executing a REXX script that automates the creation of each new customer environment by passing parameters to the tools written during the tests to generate a new system.

Use of the solution in production has allowed the telco provider to offer substantially lower costs to end customers, and the service has proved to be reliable and stable as they add more customer systems. An additional offering for mail queuing in case of customer server failure is in progress, adding sendmail and mailbox access via POP3 and IMAP servers to the list of large scale production applications.

Solution Commentary

Lessons Learned

Viability of High-Density Server Farms in Virtual Servers

Demonstration of a test farm on this scale provides technical evidence that a large-scale Internet service is possible and practical using a virtual server environment. By employing the system management and resource control tools that are integrated into the VM hypervisor, we were able to construct a WWW hosting facility capable of rapid growth at low cost to the provider, with equal performance to a comparable discrete system facility.

Direct Portability of Application Source to Linux for S/390

Open source application source code transferred to the Linux for S/390 environment was simply recompiled and operated correctly without code changes. In most cases, the process involved updating the compilation control file (makefile) to recognize a new operating system platform target and the following commands: ./configure make World make install

The RedHat Linux-based automated package install tool RPM is also completely functional in this environment, and can be used to speed deployment of applications by providing pre-compiled binary install kits for multiple architectures.

Rapid Deployment Capability of Virtual Servers

Virtual servers can be easily generated from a template in the VM environment. The LIKE functionality of DIRMAINT can be used to create a user similar to another userid already existing, and can be scripted using standard REXX tools.

Hypervisor Instrumentation and Resource Controls Well Suited to Application QoS Controls and Capacity Planning

The VM monitor data and CP resource management controls provide a significant level of insight into the operation and management of Linux systems in a large scale environment. The monitor data is detailed enough to track specific

I/O devices and could easily be used for billing purposes or as feedback into a more generalized resource monitoring system for controlling resource utilization and preventing runaway traffic utilization.

Areas for Future Research

Scheduler Handshaking in Virtual Environment

An area ripe for exploration would be integration with the underlying VM hypervisor to provide more sophisticated scheduling algorithms and/or a cooperative scheduling paradigm to reduce the idle timer problem described below. If the Linux kernel were modified to take advantage of the VM/ESA PAGEX support, it could substantially increase the efficiency of dispatching multiple virtual Linux systems by allowing an idle Linux system to give up a time slice if blocked on I/O or other interrupt processing.

Time-slice Management

The current Linux kernel design assumes that Linux is the only operating system active on the physical machine. The Linux kernel uses a simple method to handle waiting for work when idle: the kernel sets a timer interrupt to awaken it periodically. Upon receipt of the interrupt, the kernel scans a queue for events to process. If no events are found, the kernel resets the timer interrupt and quiesces again. This process consumes several tens of thousands of machine instructions per instance – essentially wasting the cycles if no work is to be performed.

In the default Linux distribution, the timer interval for idle scans is set at 100 Hz. Normally, this would not be a problem on a relatively capable machine, however in a virtual server environment, these constant wakeup interrupts consume valuable CPU resources that cannot be delivered to other virtual servers. To successfully deploy large numbers of virtual servers, it is necessary to adjust this interval to reduce the number of wasted cycles used on idle interrupt processing.

Note that adjusting this timer also affects the interactive performance of the Linux system; the idle timer is also used to compute the timeslices given other

processes in the Linux scheduler and reducing the number of timer interrupts lengthens the timeslice per process at the expense of system responsiveness. This is generally not a problem for server-only applications, but will significantly impact interactive use.

Storage and Filesystem Management

Use of an automounter tool would simplify configuration and management of filesystem pools, as would some effort committed to software RAID support at the virtual server level. Currently, filesystems larger than a physical volume are not available for end customer use, requiring some consolidation of data or multiple mounts to create large files. This area is under investigation by several people in the open source community, and should be included in future kernel releases.

IBM Corporation
Integrated Marketing Communications,
Server Group
Route 100
Somers, NY 10589

Produced in the United States of America
10-00

G326-4011-00