Large-Scale Linux Systems Monitoring with OpenNMS

Southern California Linux Expo, 26 Feb 2011

Jeff Gehlbach
jeffg@opennms.org
Who Am I; Why Am I Here?

First Linux distro: Slackware 3.1, 1994
First kernel from source: 1.2.13, 1995
Other FLOSS management apps: MRTG, 1998; Cricket; Cacti; Nagios
Proprietary management platforms: HP OpenView, 1999; Netcool OMNIbus; Empire SystemEDGE; Concord eHealth
First SCaLE: 6x, 2008
Large-Scale

Highly subjective measure; hundreds +

Photo Credit: US National Oceanic and Atmospheric Administration
Linux

Major distributions of Linux 2.6
Also (with issues):
Mac OS X, *BSD, (Open)?Solaris 10+
Similar but different: AIX, HP-UX
Systems Monitoring

Really: Systems Management.

How to add them all to be managed?
Services up and responding quickly?
Something happened, how to know?
What's happening under the hood?
Something shiny to show the boss?
With OpenNMS®

World's First Enterprise-Grade Network Management Platform Developed Under the Open Source Model

Started in 1999 by ex-OpenView hackers
Maintained by the Order of the Green Polo
Supported, sponsored by my employer
Consistent model designed for huge scale
100% GPLv2 codebase
Will never suck
Will always be Free (as in Freedom)
Not Just for Systems

This talk focuses on managing servers.
OpenNMS manages much more.
→ Infrastructure (switches, routers, UPS)
→ Storage (SAN, NAS)
→ Environmental sensors, PDUs
→ Telco gear (TDM, 3G/GSM networks)
→ Anything with an IP address
→ Quite a few things without
Not “Based On X”

Built from the ground up
100% GPLv2 code base, Java
Makes extensive use of good libraries
Does not duct-tape in other apps
→ That way lies the end of scalability
→ Not to mention maintainability
Architectural decisions dictated by requirement to scale huge.
Use What Works For You

If you're happy, don't mess with it.
But maybe it wasn't designed for that...

Photo credit: Wikimedia Commons
Analogy: Alex Finger <af@genevainformation.ch>
Your Systems are Important

• The network exists to connect systems.
• Success → more systems all the time
OpenNMS is...

- Free
- Flexible
- Powerful
- Supported
- Designed to save you time
  - We consciously avoid design decisions that would put up scale walls, but HW matters
  - Properly sized hardware → awesome scale
  - Undersized hardware → EPIC FAIL
Sizing OpenNMS

Money-spending order:

→ Storage
→ Memory
→ CPU cores

→ More cores > faster cores
Sizing OpenNMS: Storage

Direct-attached
- Many and fast spindles
- Good SAS HBA w/ big BBWC
- RAID5 is harmful. Seriously.

SAN
- FC, iSCSI, NFS all fine
- Mind the pathing
Sizing OpenNMS: Database

Give PostgreSQL its own server
Use PostgreSQL 8.4 or 9.0
If using < 8.4, you MUST tune it.

8.4+ tunes its own FSM. WIN.

Use the C-language IPLIKE sproc.

Images: pgforge; Wikimedia Commons; venganza.org
Sizing OpenNMS: Filesystems

1. Give PostgreSQL tablespaces and/or transaction logs own FS.
2. Give RRD files own FS.
2a. Separate perf, response
Each FS on its own I/O path, i.e.
RAID volume, LUN.
Two RAID1 > One RAID10.
Sizing OpenNMS: Disk I/O

This is why disks matter :) 

Enable store-by-group RRD persisting

# In opennms.properties
org.opennms.rrd.storeByGroup=true

Use MNIO JRobin back-end

→ In OpenNMS 1.8.10+

# In rrd-configuration.properties
org.jrobin.core.RrdBackendFactory=MNIO
Sizing OpenNMS: Memory

For large-scale, start at 4GB 64-bit kernel so you can use it!
Give PostgreSQL plenty.
Give the JVM plenty.
→ Default 256MB heap too small
→ Max PermGen size also
Mapping Needs to Capabilities

How to add nodes to be managed?
→ Discovery and Provisioning

Services up and responding quickly?
→ Service Monitoring (polling)

Something happened, how to know?
→ Event Management and Notifications

What's happening under the hood?
→ Performance Data Collection
Performance Data Thresholds

Basic threshold: single variable
Expression-based: multiple variables
Configurable time-over-threshold trigger

Evaluate:
→ High / low (optional re-arm)
→ Relative change (ratio, no re-arm)
→ Absolute change (optional re-arm)
Data Collection Protocols

The great thing about standards...

**SNMP**: Standards-track, tunable, robust

→ **Use SNMP when you can.**

**HTTP**: Handy when SNMP is impractical

**JMX**: Peek inside JVM, app containers

**NSClient**: Flaky at scale; can be handy

**WMI**: Outside scope of this talk :)

**XMP**: Next-generation P2P protocol
Discovery and Provisioning

**Discovery:** Awareness of a previously unknown IP address, usually via ping

**Provisioning:** Finding out all we can and representing results in our model.

Service(s) → Interface(s) → Node
Provisioning

Capsd: Legacy capabilities scanner.

Automatic Provisioning: Seed an IP address; scan for interfaces and services.

Directed Provisioning: Seed an exact set of known IP interfaces and services.

Policy-Based Provisioning: Seed an IP address; scan for interfaces and services, deciding on persistence, data collection, service monitoring, categorization...
Provisioning (cont'd)

External provisioning sources too!

*DNS import:* Do a zone transfer, create nodes and interfaces from 'A' records

*Your DB:* Write a CGI that generates XML describing your systems, feed the URL to Provisiond, watch the magic happen

*EC2-compatible APIs:* In a feature branch, track me down if you want to talk
Service Monitoring

Is a service on an interface responding?

*Simple*: ICMP Ping, HTTP GET

*Moderate*: Processes via SNMP

*Advanced*: Page Sequence,
Mail Transport

Optionally store response times
Event Management

Something happened in the network...

*Internal:* A service was found to be down

*External:* SNMP traps, syslog, TL1

*Custom:* XML-formatted events over TCP

Events optionally “de-duplicated” to alarms with a “count” attribute.
Notification Management

...now tell one or more people about it.

**E-Mail:** JavaMail API. Avoid `/bin/mail`.

**XMPP:** To individuals or group chat.

**Asterisk:** “Press 1 to acknowledge…”

**Custom:** Fork a command. Use sparingly. Reusable destination paths, escalations, auto-acknowledgement for certain cases.
Performance Data Collection

Peer inside the system to find out...

Network: Traffic, discards, errors, *cast...

CPU/Memory: Utilization, time-in-state...

Filesystem/Disk: Utilization, reads, writes

Derived: Load average, users, processes

Whatever: Straightforwardly extensible

Store data for graphing, TopN reporting

Threshold data in real-time (→ events)
Net-SNMP

Multiplies the power of OpenNMS.

*Ubiquitous:* The FLOSS UNIX SNMP agent

*Capable:* Many useful MIBs built in

*Extensible:* Glue in arbitrary commands

*Autonomous:* Self-monitor, send traps
Net-SNMP self-monitoring

**Log Files:** logmatch, file

**Processes:** proc, procfix

**System:** load, swap

**Whatever:** Net-SNMP can self-monitor any MIB object right on the box.

**Send DISMAN traps:** iquerySecName, trap2sink, monitor

(→ Events, alarms, notifications)
Extending Net-SNMP

*Old Directives:* `sh`, `exec` (don't use)

*New Directives:* `extend`, `pass`

Run a command, glue its output into the MIB tree – with configurable caching

**Example:** Provide missing CPU count

```plaintext
extend .1.3.6.1.4.1.5813.255 cpuCount /bin/egrep -c \\
'\^processor.*?\.\.*?[0-9]+' /proc/cpuinfo
```

Collect this new object in OpenNMS:

```xml
<mibObj oid=".1.3.6.1.4.1.5813.255...100" instance="1"
    alias="nsExtCpuCount" />
```
Net-SNMP Issues

Old Releases: 64-bit problems abound
5.2.1.2: Nasty interface counter problem
Release <5.5: dskTable w/FS >= 2TB :'(
Annoyance: No CPU core count object

Image Credit: teh internets (sadly, unknown)
Anti-patterns

Image Credit: teh intarweb
Anti-patterns

- Forking a lot of stuff on the NMS box (notification commands, GpMonitor)
- Using SSH + `$SHELL` as an agent
- Favoring TCP-based protocols because SNMP / UDP is “unreliable”
- Keeping RRD performance data forever
- Keeping RDBMS event data forever
- Styling your OpenNMS configs after your Nagios configs
Something Shiny for the Boss

• Resource graphs for visualization of response and performance data
• PDF availability reports (month, YTD)
• JasperReports integration
  – Events, alarms, etc. from RDBMS
  – Response and performance data from RRD files
  – Generated ad-hoc or scheduled; e-mail
Performance Data Eye Candy

Bits In/Out (High Speed)

- In Avg: 172.25 k
- Out Avg: 51.93 k
- Total In: 14.76 G
- Total Out: 4.46 G

Load Average

- 1 minute Avg: 3.68
- 5 minute Avg: 3.67
- 15 minute Avg: 3.64

System Memory Stats

- Used (Other) Avg: 631.57 M
- Total Swap Avg: 2.00 G
- Total Real Mem Avg: 1.98 G

Disk Space on /

- Total Avg: 33.03 G
- Used Avg: 15.25 G

CPU Usage

- User Avg: 27.88
- System Avg: 1.08
- Interrupts Avg: 39.29
JasperReports Eye Candy
MOAR!1! JasperReports Eye Candy
Questions, Contact

Ask away!

identi.ca: @jeffg (group !opennms)
E-mail: jeffg@opennms.org
IRC (Freenode): jeffg – #opennms
Twitter: @jgehlbach
License

This work is licensed under the terms of the Creative Commons Attribution-ShareAlike 3.0 license.

http://creativecommons.org/licenses/by-sa/3.0/