GENI Operational Monitoring: status and next steps

Chaos Golubitsky

October 28, 2013

www.geni.net

Sponsored by the National Science Foundation

Introduction

- Since GEC14, we've been prototyping pieces of an operational monitoring system for GENI
  - Code contributed by GMOC and by AM operators
- Benefits so far:
  - Outages in early rack deployments reported by monitoring system (rather than by experimenters)
  - GENI metadata reported by monitoring used to
    - help plan outages and upgrades
    - get information about who is using GENI
- Next: make the system robust and complete
- Goal: largely complete system by GEC21

Overview

- "Complete" monitoring system functionality motivated by three representative use cases:
  - LLR inquiry
  - usage and health report
  - Problem alerting and status
- Components of system needed to support use cases:
  - Overview of components
  - Current status (what's complete, what's not started, what's in progress)

Use cases: a few comments

- Please offer feedback:
  - What use cases are missing?
  - What details are missing/wrong for these use cases?
- When investigating a problem, we always want as much data as possible!
  - ...but programmer time is needed to add each data item
  - “everything that would be useful” is overwhelming
- My proposed strategy:
  - List a broad range of useful pieces of data
  - Form vs. content – get a working system, then add
  - When adding data types, focus on “bang for buck”

Use Case 1: LLR Inquiry (1)

- Summary: a complaint is made to GENI LLR about some resource misbehaving in some way. The LLR needs to look up that resource and find contact info for a person who is responsible for it.
- Details:
  - Inquiry starts with a report about a misbehaving resource at a time (maybe in the past)
  - What type of resource?
  - What type of contact info?

Use case 1: LLR Inquiry (2)

- What type of resource?
  - IPv4 address, which could be:
    - statically assigned control IP of a physical resource
    - dynamically-assigned control IP of a virtual resource
    - dynamically-assigned dataplane IP (e.g. on a shared VLAN)
  - VLAN assigned to an experiment on a device
  - Pattern of network traffic from a device or rack:
    - an unusually large quantity of traffic
    - traffic matching some specified pattern other than source IP

Use case 1: LLR Inquiry (3)

- What type of contact info?
  - Experimenter whose sliver is causing the observed symptom
  - Project lead of the project under which the responsible experiment is being run
  - Operational contact for the rack or for the location where the rack is hosted
  - Name and some method of contact: e-mail, phone
- Note:
  - Experimenter-identifying information must be protected
  - This use case doesn’t specify a particular interface

Use case 2: usage and health report (1)

- Summary: operators produce a periodic report about GENI resource availability and utilization by experimenters.
- Details:
  - What can reports measure?
  - How can reports be run?

Use case 2: usage and health report (2)

- What can reports measure?
  - Experimental activity on GENI:
    - Number of slivers over time (breakdown based on distinct experimenters, distinct aggregates, etc)
    - *Active* slivers over time (this is harder)
  - Saturation of GENI resources over time:
    - When are bare-metal nodes reserved?
    - CPU, memory, and disk utilization of shared nodes
    - Bandwidth utilization (dataplane, control plane)
  - State/health of aggregates and resources over time:
    - Aggregate version information
    - Number and type of resources on each aggregate
    - Reachability of aggregates (ping, AM API) and resources

Use case 2: usage and health report (3)

- How can reports be run?
  - Scriptable/self-service access to at least some reporting (esp. current/recent state) is very desirable:
    - Have all aggregates updated?
    - How busy are resources needed for upcoming tutorials?
    - How many experimenters will be affected by an outage?
  - Manual intervention is probably okay for more detailed/ infrequent reports:
    - Breakdown of GENI experimental usage over the past 4 months

Use case 3: problem alerting/status (1)

- Summary: GENI operators are notified of certain types of problems automatically, and can view/ query current status of those detected problems.
- Details:
  - What conditions merit alerts?
  - Who receives alert notifications?
  - What does the status display need to do?

Use case 3: problem alerting/status (2)

- What conditions merit alerts?
  - Moving target, so it should be easy to add checks later!
  - Are aggregates reachable/responsive to AM API (getversion, listresources, trust of expected anchors)?
  - Are resources alive and in an expected state?
  - Do important API state anomalies exist (e.g. slivers outliving their slices)?
  - Are resources used up (no available VLANs, VM servers out of memory, shared interfaces saturated)?
  - Are experimental network paths down?

Use case 3: problem alerting/status (3)

- Who receives alert notifications?
  - Rack operations, GMOC, and site admins (by request)
  - Operators of important non-API resources (e.g. I&M web services) by request, if possible
- What does the status display need to do?
  - Some real-time way to see current state besides notifications, to find out whether debugging succeeded
  - If no alerts contain sensitive information, maybe this can be public (or ACLs may be needed)
  - Some mechanism for seeing historical status information (could be part of a different system)

Components: introduction

- General principles for building functionality:
  - Centralization:
    - Not everything needs to be centralized, but simplify data storage/polling/access where we can
    - We require a global definition for GENI metadata used by monitoring, to tie slices, slivers, and resources together
  - “Store” and “query” as requirements:
    - To answer questions about the past or about trends, we need to store data, not just cache most recent value
    - If automated alerting is to use collected data, need interfaces to collect or query programmatically

Components: high-level functionality

- What we need:
  1. Store and query GENI-specific relational data
  2. Store and query measurement data associated with GENI objects
  3. Run real-time checks of health checks, send notifications, and update live status
  4. ...and a lot of glue code and aggregate support
- What we have:
  – Existing tools fully or partially implement these pieces
  – I'll lay out what i think the existing tools still need
  – What tools should we be using in addition/instead?

Components: relational data (1)

- GRNOC maintains:
  - relational state model (schema) for GENI metadata
  - APIs for submitting and retrieving data via authenticated HTTP
  - Python client, gmoc.py, which implements those APIs
  - Webserver, gmoc-db.grnoc.iu.edu
  - Web UI for real-time view of current relational data
- This is running code --- GPO has been using it for health checks of rack and mesoscale AMs since GEC16

Components: relational data (2)

- What's needed to make this complete and robust?
  - Missing pieces of GENI relational model, e.g.:
    - "projects" (being added now, needed for LLR)
    - slivers from non-GENI slices
    - interfaces and circuits exist, but not yet tested
  - Provide access to historical relations, currently stored but only accessible via SQL
  - Debug state anomalies found during use so far
  - Make queries much faster
- ...or use something else?

Components: measurement data (1)

- What's needed for measurement data?
  - A reasonable disk storage format (e.g. RRD)
  - API for aggregating data from sources
  - API for querying data (for alerting, trending)
  - Tagging of data with GENI metadata on both storage and retrieval
  - Realistically, a UI for displaying graphs
- Does not need to be centralized
- ...but consistency is very helpful

Components: measurement data (2)

- Operational prototypes we've looked at:
  - GRNOC SNAPP (not tagged with GENI metadata)
  - ganglia (not tagged with GENI metadata)
  - GPO graphite prototype (not tested at scale)
- ...or use something else?

Components: real-time status checks

- Nagios:
  - Well-known and widely deployed
  - Supports alerting and red/green status display out of the box
  - GPO nagios currently running ~700 “GENI-specific” checks against CHes, AMs, and web services
  - Issues:
    - default UI customization – how to integrate with ops workflow?
    - dependency logic is hard to get right
- ...or use something else?

Components: User interfaces

- Not a primary focus of monitoring system except where specified:
  - Tool builders should retrieve monitoring data and display it for their own users
- However, some UI tools are useful:
  - Diagnostic UI for real-time view of relational data (GRNOC has an implementation of this)
  - Tools for visualizing circuit/topology data: any ideas?

Components: glue code

- Code needed for components to exchange data:
  - Real-time alerting system needs to retrieve relational data to run checks against it
  - Real-time alerting system needs to retrieve measurement data to run checks against it
  - Real-time alerting system needs to report live check results as measurements so trending can be done
  - Measurement stores may need to retrieve relational data to use it for tagging
- Nagios relational and measurement retrieval are prototyped at GPO

Components: AM/CH code

- Support needed by AMs and CHes for all desired relations and measurements:
  - CHes report experimenter projects, contacts, slices
  - AMs report relational and measurement data using APIs or client modules
  - Real-time alerting runs custom network checks against CH and AM APIs
- FOAM, ORCA, and ProtoGENI support this code:
  - Need support from new AM types (e.g. stitching AMs)
- ...or do this another way?

Components: statically maintained data

- A variety of data has to be maintained statically in order for monitoring to work:
  - List of clearinghouses to monitor
  - List of aggregates to monitor
  - Contact info for CH/AM operators and POP (CH/AM physical location) operators
  - Set of non-AM health checks to run (e.g. web services)
  - Contacts to whom health check results should be sent
  - Dependency logic about what checks rely on what other checks

Summary: what do we need?

- Relational data service: complete and fix bugs
- Measurement data service: choose/implement one or more solutions
- Real-time alerting: maintain and hand off
- GENI topology data: choose/implement solution
- Glue code and AM/CH code:
  - Submit/retrieve glue for measurement store
  - Add measurements for existing AMs
  - Add relational support for all GENI CHes
  - Add relational/measurement support for all AMs
  - Statically maintained data: hand off as needed