Hewlett Packard
Enterprise

ORCHESTRATION OF EXASCALE FABRICS USING THE SLINGSHOT FABRIC MANAGER

PRACTICAL EXAMPLES FROM LUMI AND FRONTIER

Forest Godfrey (Distinguished Technologist)
Jose Mendes (Fabric Manager Architect)

May 11, 2023

AGENDA

- Fabric Manager Overview
- Architecture
- Performance and recent optimizations
- Upcoming improvements

AGENDA

- Fabric Manager Overview
- Architecture
- Performance and recent optimizations
- Upcoming improvements

FABRIC MANAGER OVERVIEW: BASIC TECHNOLOGIES

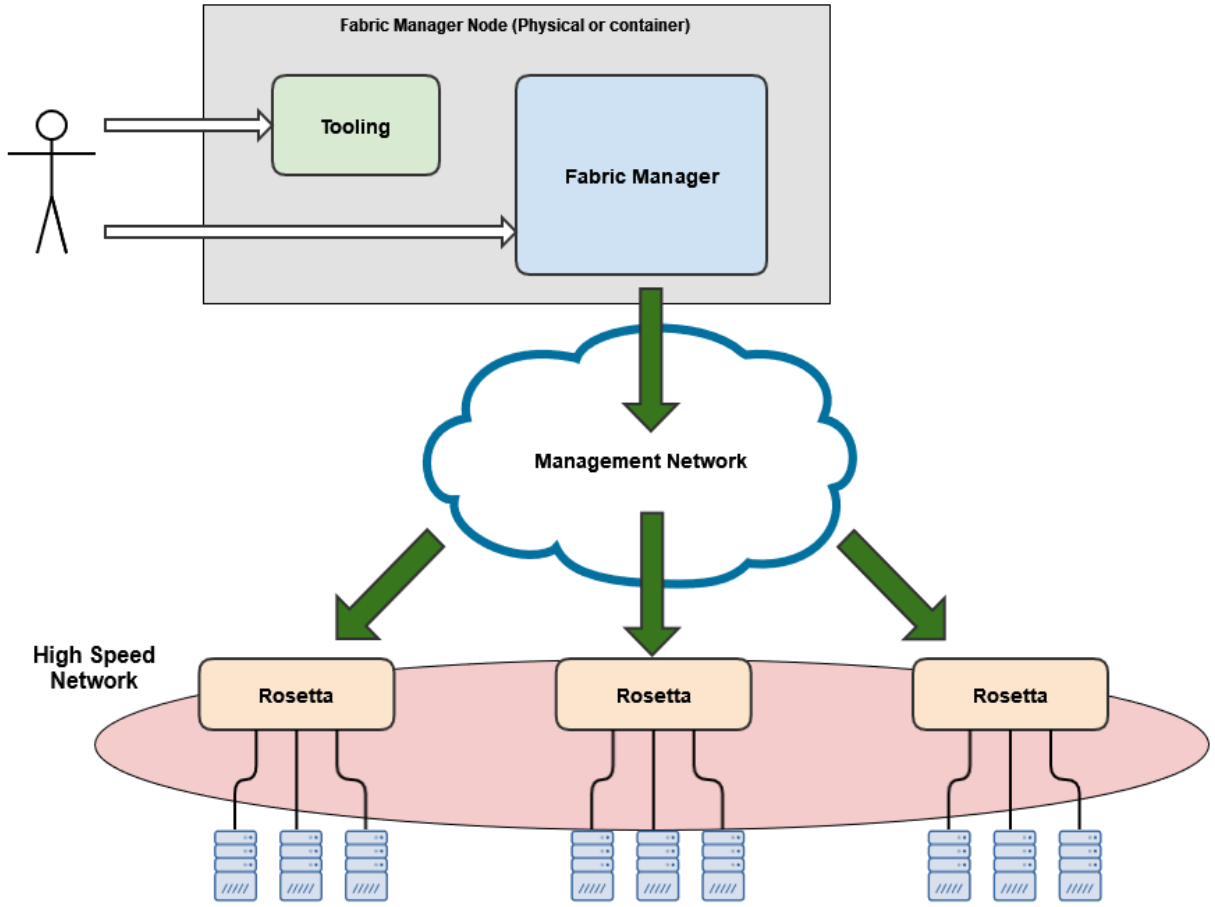
- Written in Java
- Based on existing core technologies
  - DCS Core: open source project from VMware
    - Provides the REST interface and interface to the database
  - Apache Lucene database
    - Provides permanent storage for fabric data
    - Used in many open source projects
      - ElasticSearch is an example

FABRIC MANAGER OVERVIEW

- Responsible for configuring the Slingshot network
- Primary point of contact for the network administrator
- FM vs. FMN:
  - FM: Fabric Manager
  - FMN: Fabric Management Node
- Wide variety of fabric setups
  - From simple clusters with 128 nodes and 3 switches to the largest computers in the world

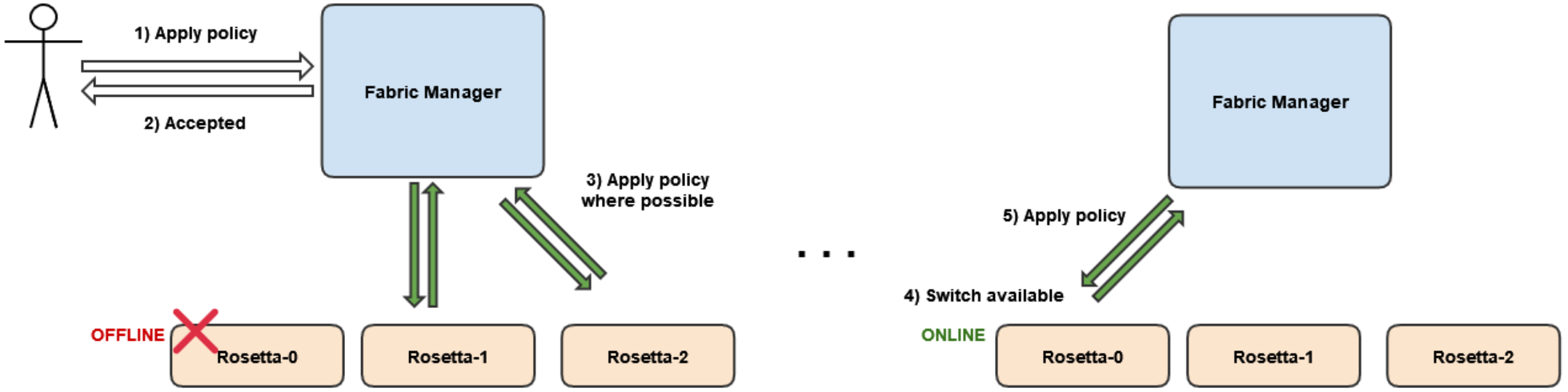
- LUMI
  - 874 switches
  - >4000 nodes , ~12.000 network endpoints

- Frontier
  - 2500 switches
  - ~10.000 nodes, ~40.000 network endpoints



FABRIC MANAGER OVERVIEW: DEPLOYMENT MODEL

- Eventual consistency model
  - Most actions are **not** guaranteed to have been completed when a command finishes
  - Policies will apply when it is possible to do so
  - Covers scenarios where devices are not immediately reachable
  - At large enough scale, equipment failure cannot be ignored
    – "Unlikely" times 40,000 endpoints = Likely
  - Includes redeployment after failures or intentional reboots



FABRIC MANAGER OVERVIEW: FABRIC AGENT

- Won't go into too much detail on the agent
- The Fabric Agent runs on the Rosetta Switch Controllers
- Responsible for the high level interface to lower level switch programming
- The Fabric Agent is built from the same code base as the fabric manager

AGENDA

- Fabric Manager Overview
- Architecture
- Performance and recent optimizations
- Upcoming improvements

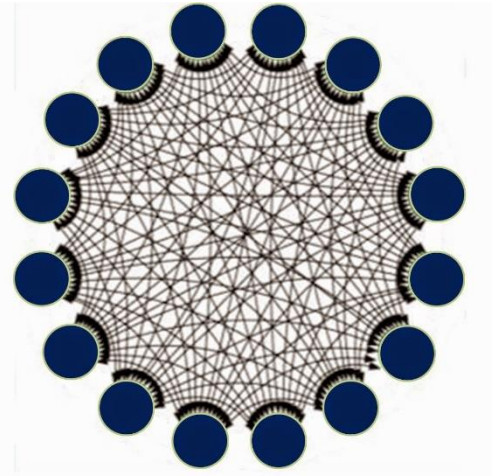
FABRIC MANAGER ARCHITECTURE

- The Fabric Manager is composed of several “services”
- Each service:
  - Is a unit with business logic
  - Exposes a REST interface, ALL services have an associated endpoint
  - Has associated JSON data
  - Interactions done only through REST API calls
    - This includes internal interaction between the services
- Services can be storage-backed or resident in memory

FABRIC MANAGER ARCHITECTURE

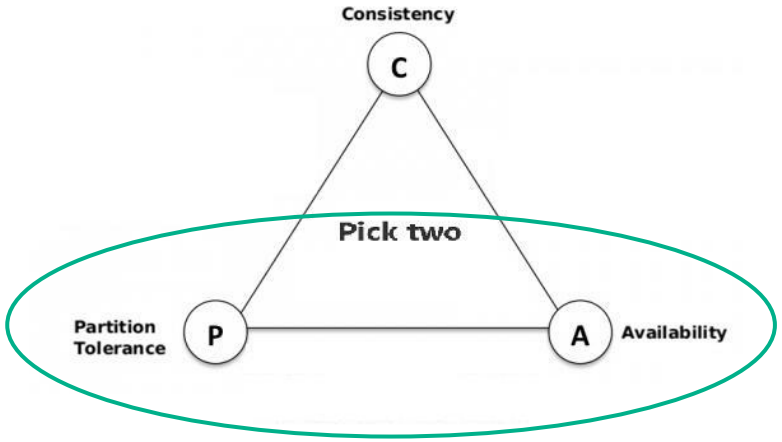
Challenges

- Operate equally well on small and large systems
- Equipment failures can occur
- Ensuring that the entire system is present at any moment is non-practical



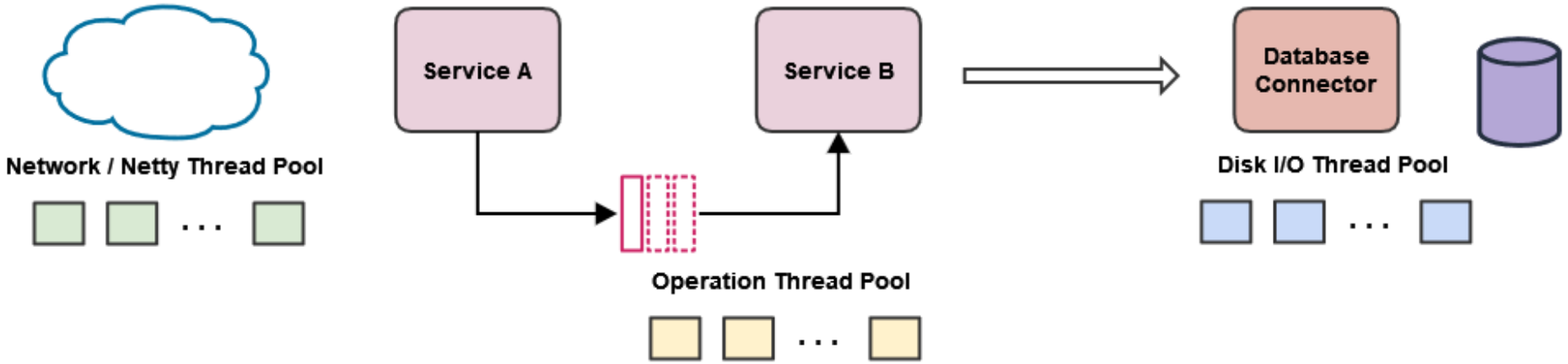
Solutions

- Periodic sweeps / polling of fabric status
  - ALL timers are adjustable at runtime for tuning
- Schedule redeployment of policies when applying a policy fails
- Isolated blocks of business logic and data
- Per-device state machines
  - Ensure actions happen in correct order
  - Allow multiple devices to progress at different rates
    - Eliminate the “no node left behind” issues with xtbounce



FABRIC MANAGER ARCHITECTURE

- Business logic is placed in request handling
  - Fully asynchronous and non-blocking
  - This guarantees that threads are released as soon as possible and can be reused
  - Scheduling is performed by the core framework
- 3 sets of thread pools
  - Operation handling – work-queue processing of operations and services' business logic
  - Network I/O handling
  - Disk I/O and data indexing



FABRIC MANAGER ARCHITECTURE

- Services interact through REST messages or “operations”
  - These are standard HTTP actions: GET, PATCH, PUT, etc
  - Initiated by external requests or by other services
  - Since request handling cannot block, services create chains of asynchronous operations

- That is, services interact through message passing
  - There is a cost incurred in serialization/deserialization of messages
  - Trade-off accepted to achieve a higher degree of parallelism

- With small non-blocking processing request handling blocks, Fabric Manager’s resource consumption largely correlates to:
  1. amount of messages and
  2. size of messages (cost of serialization/deserialization)

FABRIC MANAGER ARCHITECTURE

- Periodic tasks are actions that execute periodically and can trigger requests to services
  - E.g Redeployment of configurations. Periodically checking the state machine for configuration deployment and, if FAILED, start it again to attempt redeployment

- Pros of periodic actions and polling:
  - Provide a more predictable load profile
  - Are easy to tune: timers can be accelerated or slowed down based on a set of conscious trade-offs. E.g. accelerate fabric sweeps for faster re-routing while slowing down redeployment of configurations

- Cons:
  - Delay between an action (e.g applying a policy to a port) and something happening
  - Higher CPU overhead during idle periods

More to be covered in “upcoming improvements”!

AGENDA

- Fabric Manager Overview
- Architecture
- Performance and recent optimizations
- Upcoming improvements

PERFORMANCE AND RECENT OPTIMIZATIONS

- Fabric Manager's resource consumption is tied to the volume and the size of messages
  - The ***format*** of the messages also has an impact
    - Deeply nested data structures incur a much higher serialization and deserialization effort
    - Flat structures offer performance gains

- Slingshot 2.0 introduced over 20 Fabric Manager optimizations

Hash comparisons

- Retrieve and compare hashes of configurations instead of having to retrieve and compare large configurations
  - Slingshot configurations are large and scale linearly with the number of switches
  - Decreases in overall resource usage, one optimization – retrieving only hashes of remote routing configuration – cut CPU and memory usage to half at scale

PERFORMANCE AND RECENT OPTIMIZATIONS

Flatten port and configuration data fields

- At scale there are 10s to 100s of thousands of ports
- There are several per-port configurations
- Automatically makes the process of applying policy to ports more efficient

Restructuring of data used by Forwarding (Routing) Engine

- Routing calculations are expensive and require a full graph containing **all** switches and **all** links
- Simplifying data used by the routing engine provides a large benefit at scale

The above are some of the major optimizations introduced. There were others such as

- Use of caches. Instead of independent services trying to do remote operations, keep a local availability cache that services can consult before attempting to go remote
- Optimize state machine transitions (less operations)
- And many others

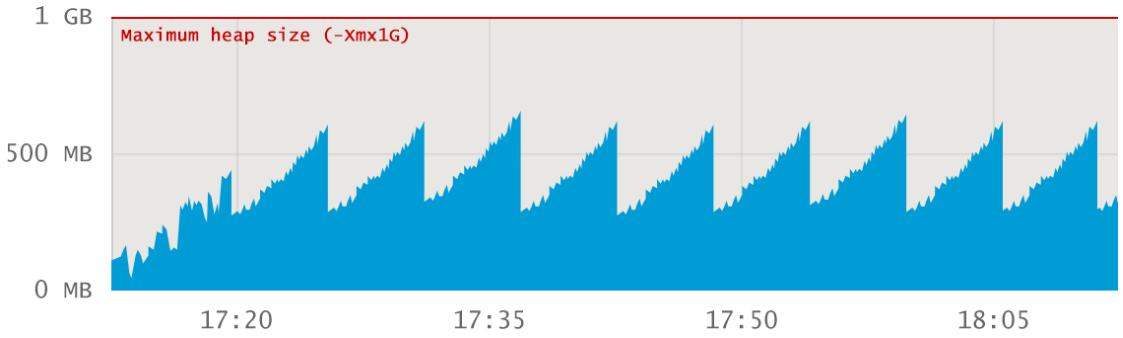
PERFORMANCE AND RECENT OPTIMIZATIONS

- Fabric Manager is written in a Garbage Collected language (Java)
- Memory profiling needs to take into account
- In non-GC languages, memory usage at any point indicates total amount of **actively used memory**
- In GC languages, it represents a cumulative value of **active** and **not yet collected** memory

- Memory profile is different pre and post-GC runs

To eliminate measurement artifacts, GC interference needs to be reduced.

- Run the GC at a deterministic point before measuring
- Not running GC during benchmarking
- Compare apples to apples



PERFORMANCE AND RECENT OPTIMIZATIONS - METHODOLOGY

All numbers obtained on:

- Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz with 192GB of RAM
- OpenJDK 11.0.16, launched with **128GB of heap memory** to minimize GC interference

After bringup, using Frontier's template, the GC was manually run before measurement started.

Simulated environment with no real switches, as close to a real load profile as possible.

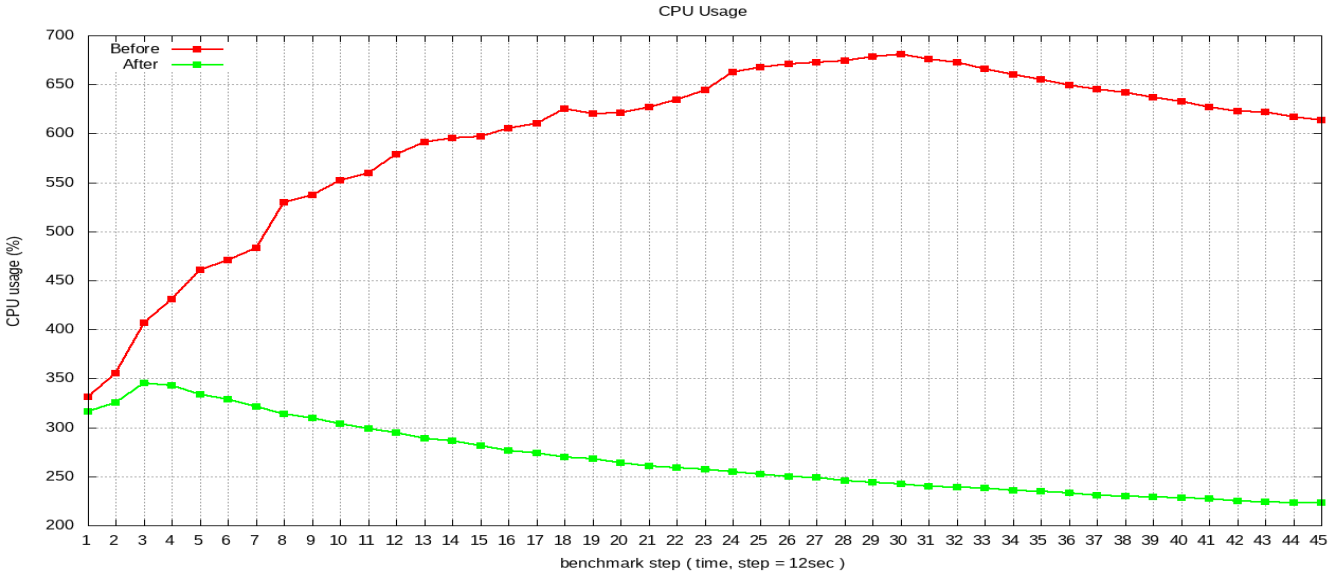
Comparisons between resource usage in 1.7.3 and 2.0.

2 sets of measurements:

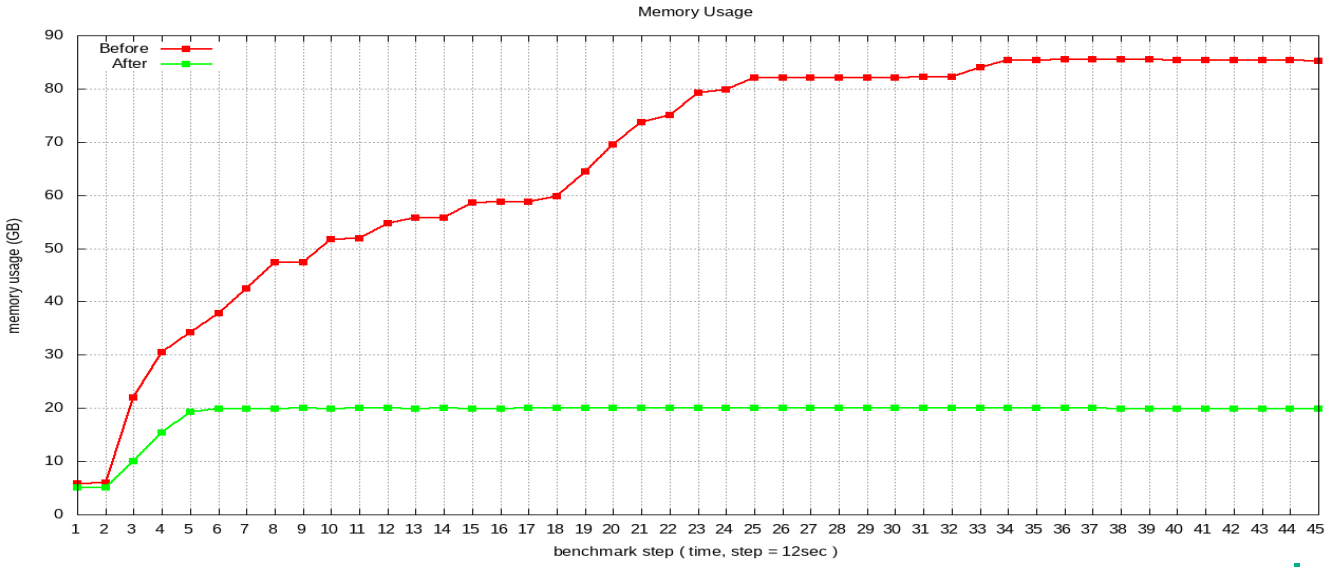
- **Idle state**, following bringup, measurement of resources needed for calculating routing init and apply switch and port configurations
- **Re-routing stress test**, continuously simulate the loss of global connections and cost of recalculating and redeploying routes

PERFORMANCE AND RECENT OPTIMIZATIONS - IDLE

- After optimizations CPU usage in idle reduces a lot (about HALF peak usage)
- Importantly, the peak resource is achieved much earlier
  - Initial configurations pushed much earlier



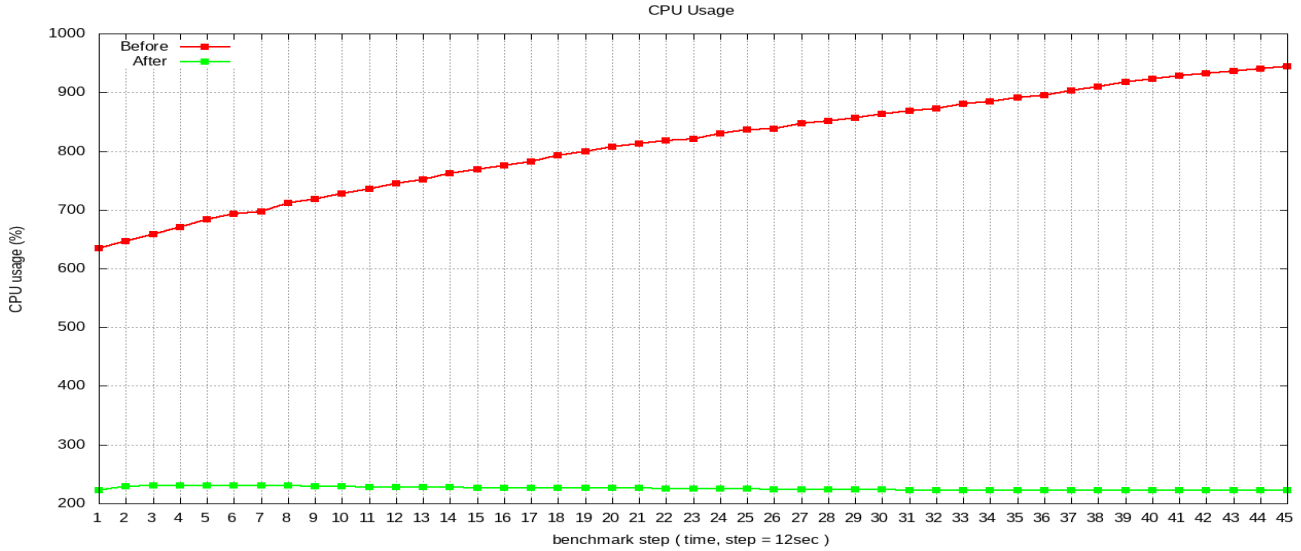
- Cumulative memory usage is reduced by a factor of 4x
- Memory usage directedly correlated to cost of message passing
- Impact of reducing message sizes and frequency of messages is visible
- As for CPU, the stabilization point is reached much earlier

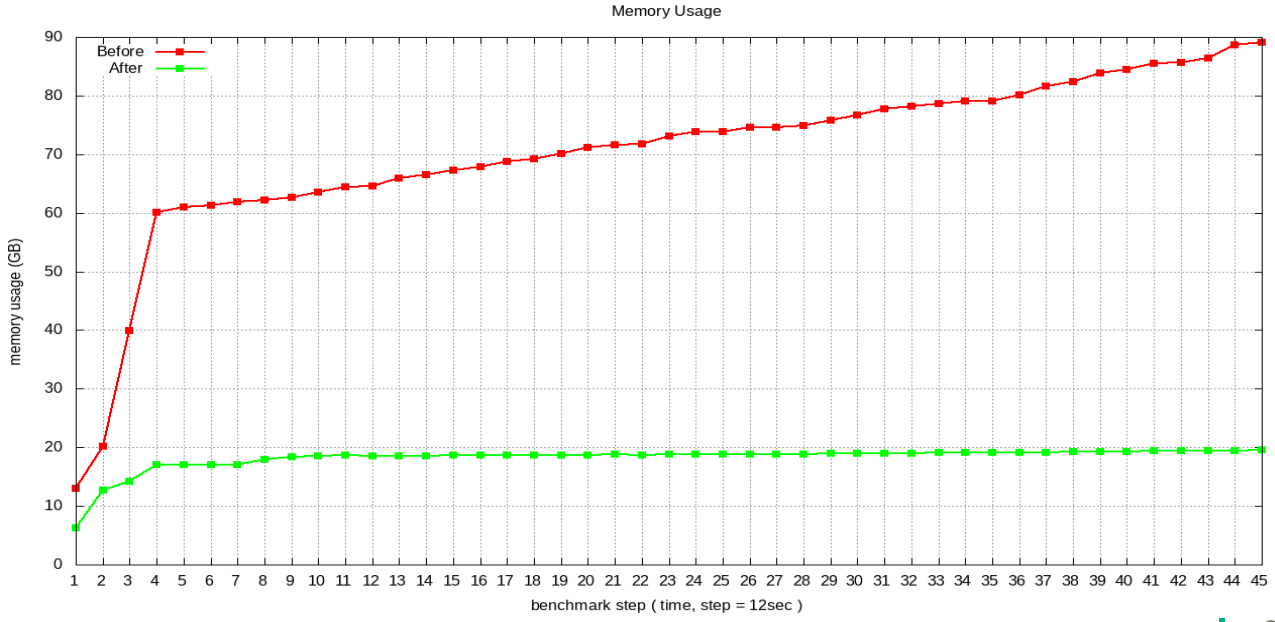


PERFORMANCE AND RECENT OPTIMIZATIONS – REROUTING STRESS TEST

- In 1.7.3 continuous (global) rerouting resulted in rising resource consumption
- CPU and memory could spike out of control (reality check: in this scenario, routing is continuously recalculated. It wouldn't happen in practice due to flap quietening)

- In 2.0, **cumulative** memory usage rises gradually (memory is being consumed for routes)
- CPU usage appear liner due to sampling resolution: when the CPU usage is sampled, routes have been calculated and distributed. It shows that CPU usage does go down.





AGENDA

- Fabric Manager Overview
- Architecture
- Performance and recent optimizations
- Upcoming improvements

UPCOMING IMPROVEMENTS – HYBRID SWEEP AND EVENTING

- Polling and periodic tasks have brought us so far
- Some of you have tuned these timers for production!

- However, “pure” polling presents a number of cons
- An hybrid model is being introduced: events for immediate actions, still maintaining polling cycles
  - Pure event based mechanisms present their own set of pros and cons

- Many Advantages:
  - Faster applications of policies
  - Lower resource usage during idle periods
  - Faster re-routing, leading to less HSN packet loss in case of failure

UPCOMING IMPROVEMENTS (II) – FURTHER OPTIMIZATIONS

- A number of other improvements to come in the future

- We haven't talked about disk usage optimization yet!
- Data restructuring, better decouple data that needs to persist from data that only resides in memory
- Besides better storage usage it also incurs less disk I/O

- Separate sending queues for local and remote operations to avoid Head-of-line blocking

- Obviously, performance optimizations will continue to be guided by profiling
- Prioritization is on changes that have a measurable impact and are noticeable by you: faster rerouting, faster fabric reconfiguration and that will make the Fabric Manager more robust

THANK YOU

Forest Godfrey fgodfrey@hpe.com
Jose Mendes jose.mendes@hpe.com