while true; do campaign.py; done

Javier Rojas Balderrama & Matthieu Simonin

March 27, 2019

Previously in XUG G5K: The massively parallel case

Experimental loop

```
@parallel
for parameter in parameters:
    bench(parameter)
```

- `bench(p)` launches one process
- `bench(p)` and `bench(p')` are independents

$\rightarrow$ You can defer most of the experimentation logic to the batch scheduler and go for a parallel execution.
Ideal case: one (idempotent besteffort) job per parameter.

The not so massively parallel case

Experimental loop

```
@parallel -> sequential
for parameter in parameters:
    bench(parameter)
```

- `bench(p)` launches a set of processes (10, 100, ...)
- Processes aren't independents
  - *Configuration dependency*. e.g. `proc1` needs to know the ip of the machine where `proc2` is launched
  - *Runtime dependency*. e.g. `proc1` starts only if `proc2` is reachable

→ this could still fit the MP case. But sometimes:

- `bench(p)` is taking a significant resources (time and space)
- `bench(p)` and `bench(p')` aren't independents.

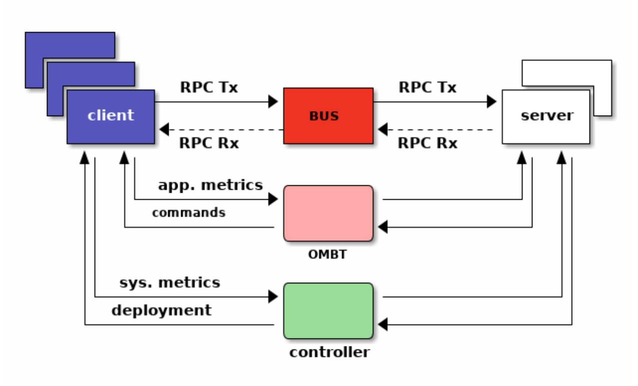
Today's XUG G5K Use Case

General question

Impact of the geo-distribution over the communication bus of a IaaS

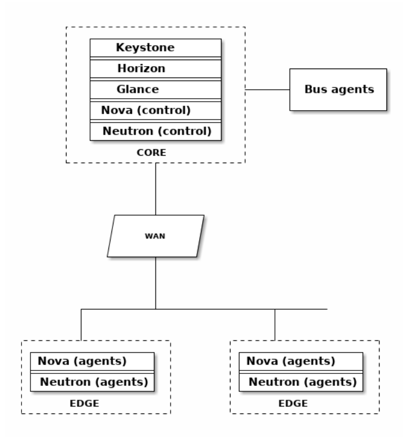
1. Synthetic benchmark: Isolated bus communication layer
2. Operational benchmark: OpenStack internal messaging at the Edge

Synthetic evaluation: overview



- Buses: RabbitMQ, Apache Qpid-dispatch-router
- Clients: 1000 to 10000
- Servers: 20 to 10000
- Messages: up to 300000
- RPC patterns: anycast, unicast, multicalst
- Bus configuration: 3 to 5 brokers/routers
- Latency: 0 to 100ms
- Packets loss: 0 to 2%

Operational evaluation: overview



- full-fledged OpenStack
- 100 to 400 computes at the edge
- 10 Rally scenarios (nova and neutron)
- latency from 0ms to 200ms RTT
- loss from 0 to 2%
- periodic network dropout (different level of aggressivity)

Experimental campaign overview

On Grid'5000

```
# data provenance
make_reservation(conf)
for parameter in parameters:
    bench(parameter)
    backup env
    clean env
# data analysis
until satisfied
    intermediate_data()
# until statisfied
    visualize()
```

- single job to run all the parameters (need some calibration) (one cluster for everything)
- clean env is TBD
- bench includes deployment / network emulation
- backup stores raw data in a storage5k space (500GB) (system metrics / application metrics)
- `create_store_` push intermediate data to a git (JSON metrics or even binary files :( )
- `visualize` using Jupyter

Experimental campaign overview

More practically:

- Centralized GIT (link)
  - Launchers + configurations (ex1, ex2)
  - Intermediate data
  - Jupyter notebooks
  - report (paper)
- Experimental framework code is elsewhere: (link)
  - The experimental loop is implemented here: (code)

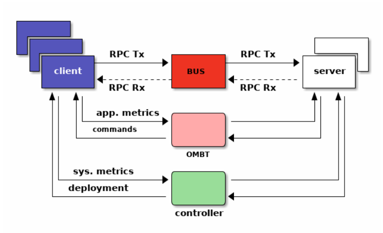
Scaling the experiment

To scale an experiment

You need good abstractions

- Infrastructure level (horizontal scaling of hardware resource)
  - Optimize use of available physical resources (e.g start more agents without wasting resources)
  - Optimize deployment time (e.g can you afford a kadeploy3 run between each parameter ?)
- Application level (cope with higher number of nodes)
  - Optimize the configuration phase (e.g what internet bandwidth is required for your deployment ?)
  - Scale the instrumentation (monitoring stack, the experimentation controller)
  - Tune system parameter (number of file descriptors, ARP tables...)
- User level (increase the number of experimenters)
  - Make things explicit
  - Improve user experience (cleaner interface / packaging of experimentation code)

How do we scale? densification



For a given bus deployment, we want to scale the number of agents

- One agent is one python process (1 CPU core)
- Agents have low CPU utilisation

→ We can pack together agents (up to 200 per machine in our case)

- BUT the deployment logic need to be deeply adapted
  - e.g handle port collisions
  - need to be scaled also
  - can be hard to maintain

How do we scale? virtualisation

- Virtualisation let us optimise the use of nodes
- Avoid the modification of the machinery (deployment and execution scripts)
- It is complementary to densification strategy
  For a fixed number of machines:
  densification + virtualisation = more resources to use
- Limits are shifted to external concerns (no more than 400-500 computes in OS)
- Deployment time is reduced (with alt-reference image from 1h to 15 min)

Calibrating 1/2

Some questions (and answers for our use case) :

- Do we need specific hardware ?
  - development phase: we want to be able to test quickly on any machine
  - production phase: we want to run all the xps on the same hardware (same cluster)
- What storage capacity ?
  - We use a 500GB storage space provided by Storage5k
  - We asked it to be shared between 3 users[1]
- How much time a campaign will take (we ran dozen of campaigns)
  - We target one run in a night duration (14 hours)
  - If that's not finished we should be able to restart easily the missing runs

[1]this is now a standard feature of G5K

Calibrating 2/2

Issues faced:

- Too many parameters (because we wanted too many points in our graphs)
- Too long duration for each parameter
- Underestimation of the effect of delay when emulating the network (think of 10^6 messages + ack with 200ms latency)
- Not anticipated scaling issues

Automating the data provenance/analysis

- Intermediate data to graphs
- Exploratory/Explanatory work
- Code looks like this: code
- Notebooks look like that: notebook

Automating the full process

- Embrace existing tools:
  `https://www.grid5000.fr/w/Grid5000:Software`
- Learning curve is most probably justified: reuse and reduce (code/errors)
- Think ahead (ACM 3Rs)
  Repeatability $\rightarrow$ Replicability $\rightarrow$ Reproductible

Backup slides

Want some graphs?

Choose in the list:

AMQP1.0 / Qpid-Dispatch-Router achieves

- Lower latency in message delivery for anycast and multicast for both RPC.cast and RPC.calls
- Significantly less resource consumption
- Supports the geo-distribution of its agent in achieving better locality
  - RabbitMQ (cluster) support for distributing its agent is (very) limited

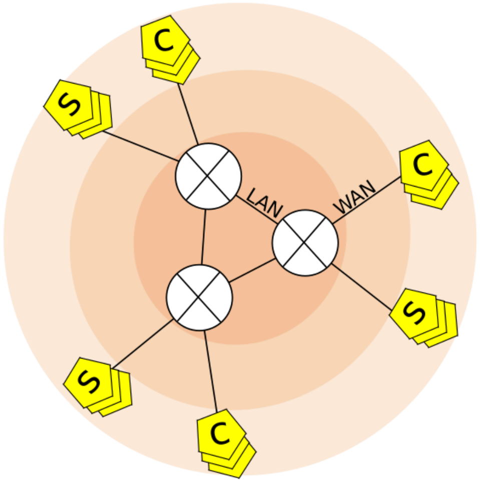
Conclusion of Operational evaluation

- In front of WAN latency and loss, the routers (no message retention) is as effective at delivering messages as the brokers (message retention)
- Routers is less resilient in the case of network dropouts
- Routers stil consumes less resources than the broker
- In both cases packet loss seem impact of the loss can be significant

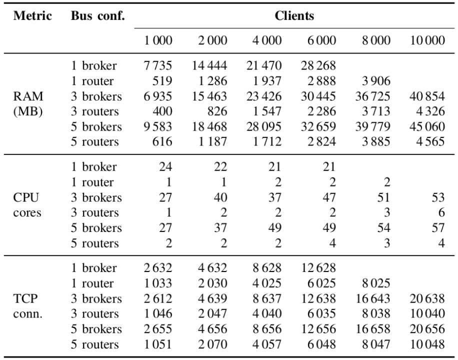
Synthetic evaluation

Synthetic evaluation

Synthetic evaluation: centralized deployment



Synthetic evaluation: Resource Consumption

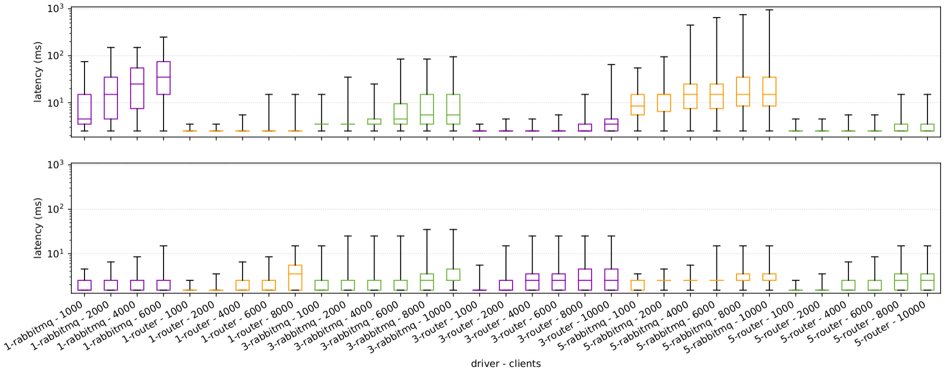


Metric	Bus conf.	Clients					
		1 000	2 000	4 000	6 000	8 000	10 000
RAM (MB)	1 broker	7 735	14 444	21 470	28 268		
	1 router	519	1 286	1 937	2 888	3 906	
	3 brokers	6 935	15 463	23 426	30 445	36 725	40 854
	3 routers	400	826	1 547	2 286	3 713	4 326
	5 brokers	9 583	18 468	28 095	32 659	39 779	45 060
	5 routers	616	1 187	1 712	2 824	3 885	4 565
CPU cores	1 broker	24	22	21	21		
	1 router	1	1	2	2	2	
	3 brokers	27	40	37	47	51	53
	3 routers	1	2	2	2	3	6
	5 brokers	27	37	49	49	54	57
	5 routers	2	2	2	4	3	4
TCP conn.	1 broker	2 632	4 632	8 628	12 628		
	1 router	1 033	2 030	4 025	6 025	8 025	
	3 brokers	2 612	4 639	8 637	12 638	16 643	20 638
	3 routers	1 046	2 047	4 040	6 035	8 038	10 040
	5 brokers	2 655	4 656	8 656	12 656	16 658	20 656
	5 routers	1 051	2 070	4 057	6 048	8 047	10 048

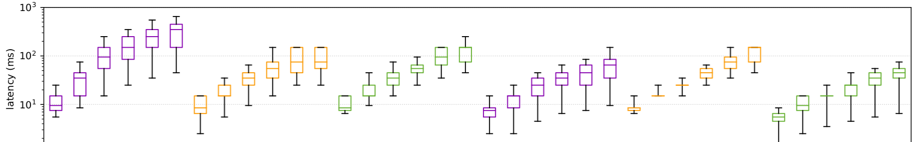
TABLE II: Results of the anycast scenario. System metrics for `rpc-call` call type. Maximum values obtained during the benchmark for memory usage, number of processors and TCP connections.

- Rabbit / QDR
  - MEM: x9 - x17
  - CPU: x8 - x27
  - TCP: x2

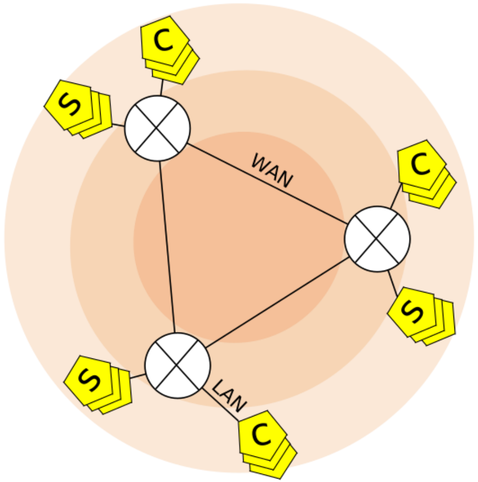
Synthetic evaluation: latency with centralized deployment



(a) Anycast scenario (top `rpc-call`, bottom `rpc-cast`).



Synthetic evaluation: decentralized deployment



Synthetic evaluation: latency with centralized deployment

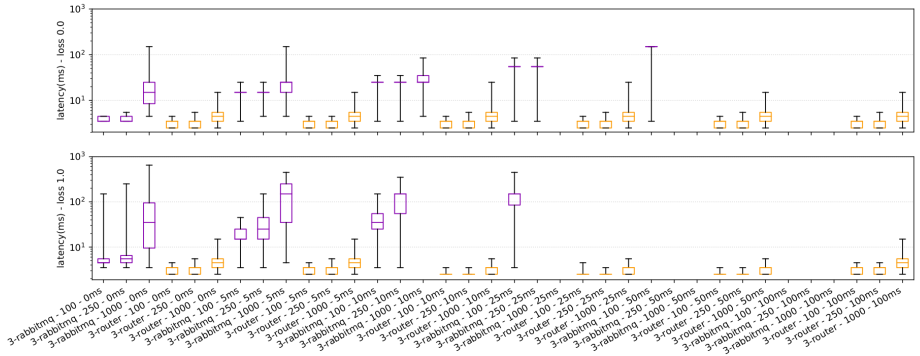


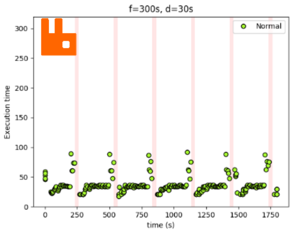
Fig. 4: Results of the anycast scenario in a decentralized deployment. Latency boxplots for bus implementations, number of clients, and link delay of `rpc-call` (top 0% loss, bottom 1% loss).

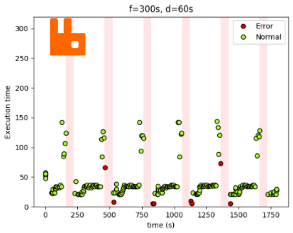
Operational evaluation

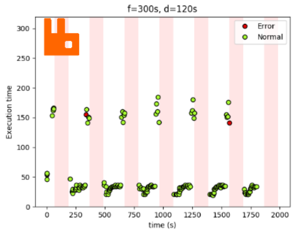
Operationnal evaluation

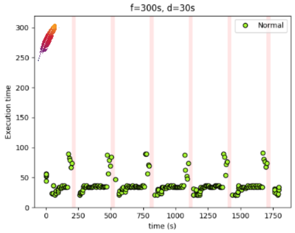
Operational evaluation: network dropout

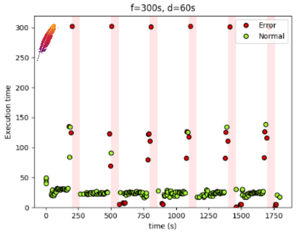
Aggresively drop network every 5min for different durations.
Boot and delete scenario.

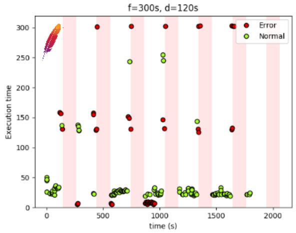








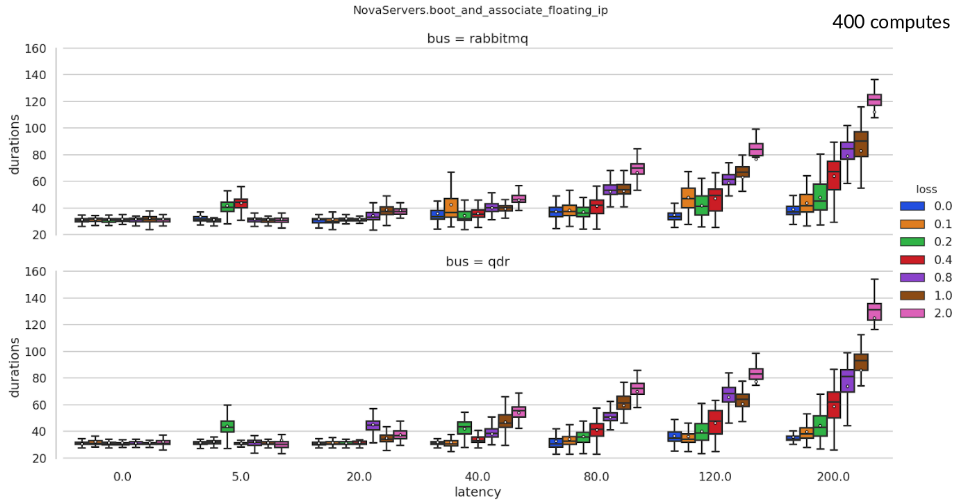




Operational evaluation: latency, loss impact

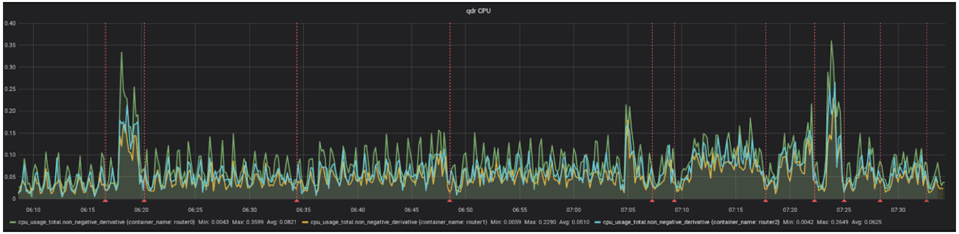
Latency and packet loss is enforced between core and edge

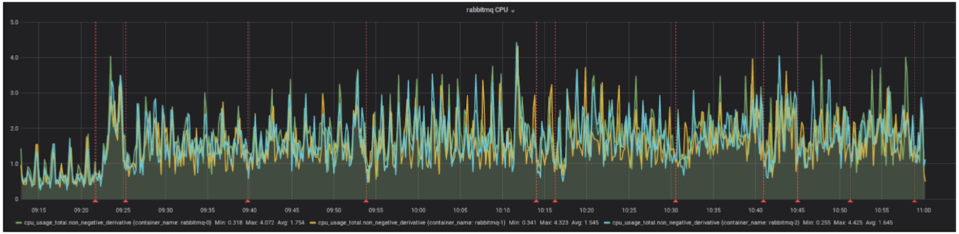
400 computes



Operational evaluation: consumption

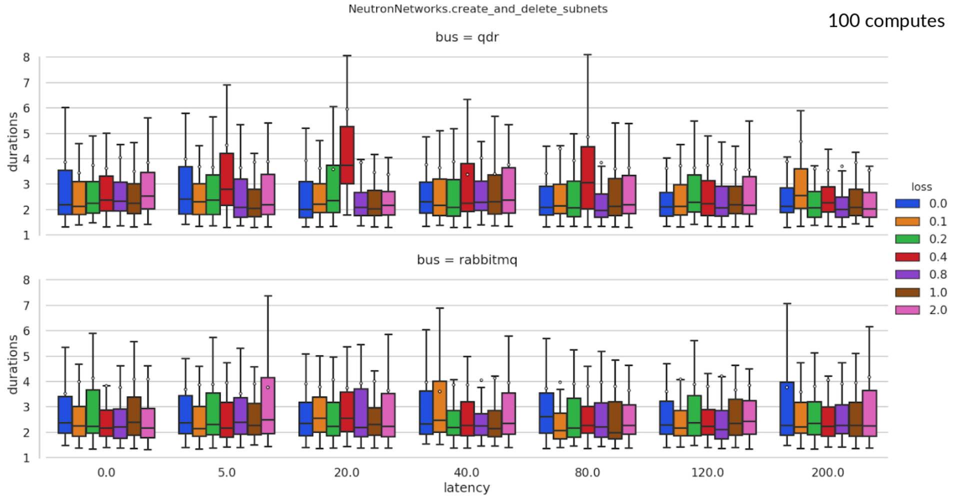
CPU consumption of QDR and RabbitMQ





x10 CPU consumption for RabbitMQ (same conclustion with RAM - x5)

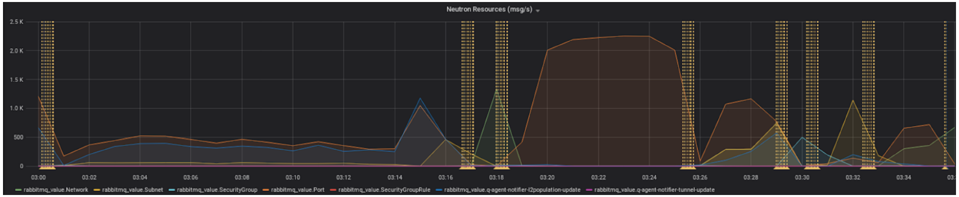
Operational testing: behind the scene



Not impacted ? Are we measuring the right thing ?

Operational testing: behind the scene

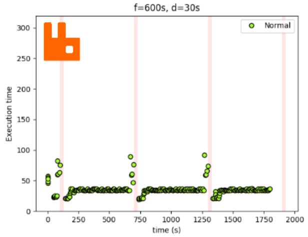
- Behind the scene
  - boot-server-and-attach-interface
  - create-and-delete-network
  - create-and-delete-port
  - create-and-delete-router
  - create-and-delete-security-groups
  - create-and-delete-subnet
  - set-and-clear-gateway

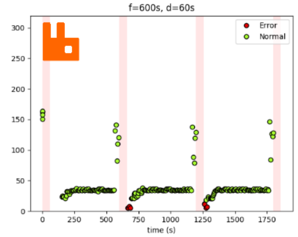


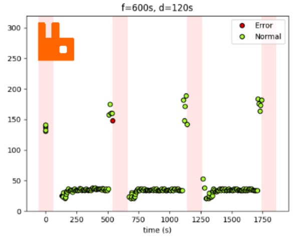
What is the impact of message loss on neutron multicasting system ? We probably need some ad'hoc methods now...

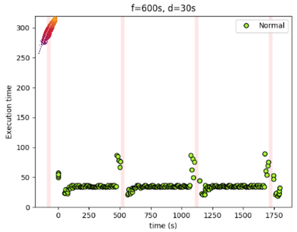
Operational evaluation: network dropout

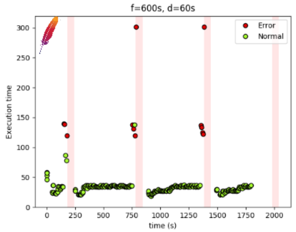
Network dropout every 10min for different durations.
Boot and delete scenario.

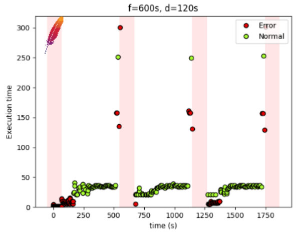




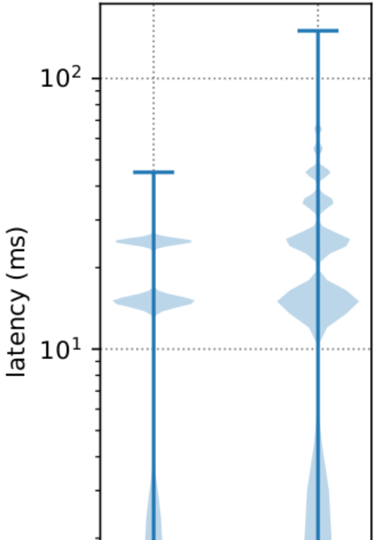


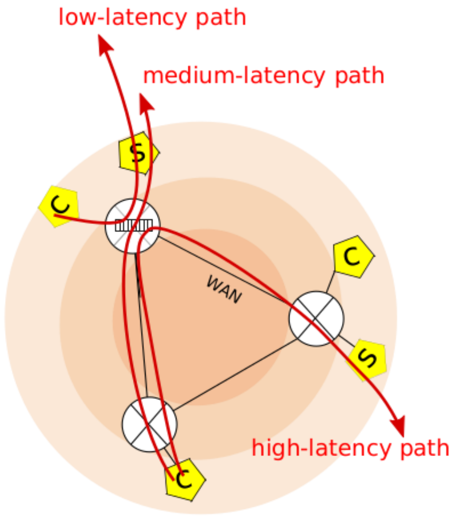






Different message paths in a RabbitMQ cluster





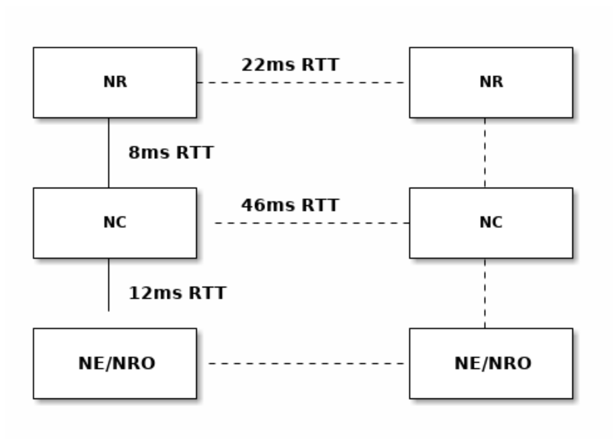
Openstack Message Bus Evaluation: What? (1/)

What ?

- Evaluation of the internal message bus of OpenStack in a Fog/Edge context

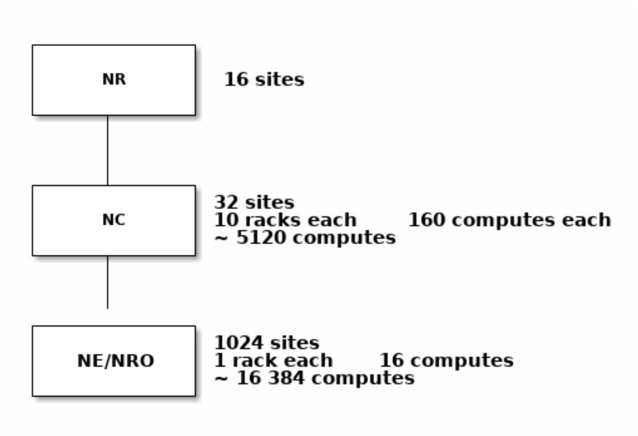
Openstack Message Bus Evaluation: What? (2/)

- Orange use case: topology

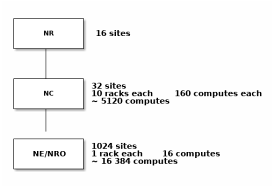


Openstack Message Bus Evaluation: What (3/)

- Orange use case: capacity



Openstack Message Bus Evaluation: What (4/)



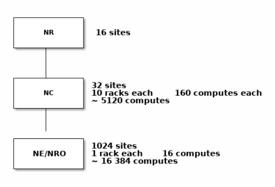
Scenario: One single distributed OpenStack

- In NRs (computes in NCs): 1x OS
  - 5120 computes - 22 ms latency (NRs)
- In NR/NC (computes in NE/NRO): 1x OS
  - 16384 compute - 44 ms latency (NCs)

Challenges

- Scalability
- Locality

Openstack Message Bus Evaluation: What (5/)



Scenario: Sharded control planes

- In NRs (computes in NCs): 16x OS
  - 320 computes each - 8ms RTT latency
- In NCs (computes in NE/NRO): 32x OS
  - 512 computes each - 12 ms RTT latency
- In NRs (computes in NE/NRO): 16x OS
  - 1024 computes each - 20 ms RTT latency

Challenges

- Top layer management
- Collaborative management: Goal of Discovery for OpenStack

Openstack Message Bus Evaluation: How

- Different access patterns
  - Unicast: direct messaging
    e.g: n-api -> n-cpt to shutdown vm
  - Anycast: queue abstraction with multiple producers/consumers
    e.g: n-cpt -> n-cond to report state (periodic tasks)
  - Muticast: notification like message to a set of subscribers e.g: q-server -> all q-ml2 agents security group change
- Different garantees
  - Call: "true" RPC (wait the return value of the remote invokation)
  - Cast: Fire-and-forget

Openstack Message Bus Evaluation: How

- Two steps
  - Synthetic evaluation: consider only low-level RPC agents
    - Evaluate the access patterns / garantee
    - In face of latency, message loss
    - Decent scale
  - Operational evaluation
    - Evaluate OpenStack
    - In face of latency, message loss and dropout
    - Reasonnable scale