Queue and RAPI Daemon Speed

Christos Stavrakakis

Greek Research & Technology Network

cstavr@grnet.gr

September 4, 2013

Question:

Question:

Say we want to create 100 instances...

Question:

Say we want to create 100 instances...
at once...

Question:

Say we want to create 100 instances...
at once...
How long do yo think this would take?

How Synnefo uses Ganeti (1)

- Use Ganeti RAPI: no need for custom API and agent

How Synnefo uses Ganeti (1)

- Use Ganeti RAPI: no need for custom API and agent
- Critical to keep request-response cycle low: enqueue jobs to Ganeti, and get the job ID

How Synnefo uses Ganeti (1)

- Use Ganeti RAPI: no need for custom API and agent
- Critical to keep request-response cycle low: enqueue jobs to Ganeti, and get the job ID
- No polling! Instead, receive asynchronous notifications from the Ganeti to track job progress

How Synnefo uses Ganeti (1)

- Use Ganeti RAPI: no need for custom API and agent
- Critical to keep request-response cycle low: enqueue jobs to Ganeti, and get the job ID
- No polling! Instead, receive asynchronous notifications from the Ganeti to track job progress
- Use Ganeti hooks?

How Synnefo uses Ganeti (1)

- Use Ganeti RAPI: no need for custom API and agent
- Critical to keep request-response cycle low: enqueue jobs to Ganeti, and get the job ID
- No polling! Instead, receive asynchronous notifications from the Ganeti to track job progress
- Use Ganeti hooks?
  - Post hooks run only on success. How to track errors?

How Synnefo uses Ganeti (1)

- Use Ganeti RAPI: no need for custom API and agent
- Critical to keep request-response cycle low: enqueue jobs to Ganeti, and get the job ID
- No polling! Instead, receive asynchronous notifications from the Ganeti to track job progress
- Use Ganeti hooks?
    - Post hooks run only on success. How to track errors?
    - Time overhead to total job execution time.

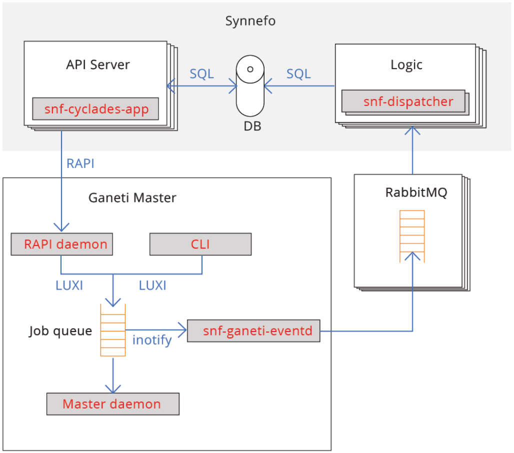
How Synnefo uses Ganeti (1)

- Use Ganeti RAPI: no need for custom API and agent
- Critical to keep request-response cycle low: enqueue jobs to Ganeti, and get the job ID
- No polling! Instead, receive asynchronous notifications from the Ganeti to track job progress
- Use Ganeti hooks?
    - Post hooks run only on success. How to track errors?
    - Time overhead to total job execution time.
- inotify: Notice changes to Ganeti job queue directory!

How Synnefo uses Ganeti (1)

- Use Ganeti RAPI: no need for custom API and agent
- Critical to keep request-response cycle low: enqueue jobs to Ganeti, and get the job ID
- No polling! Instead, receive asynchronous notifications from the Ganeti to track job progress
- Use Ganeti hooks?
  - Post hooks run only on success. How to track errors?
  - Time overhead to total job execution time.
- inotify: Notice changes to Ganeti job queue directory!
- reconciliation: Query via RAPI to reconcile state of DB with Ganeti

How Synnefo uses Ganeti (2)



Enqueue jobs

Time to create those instances

Enqueue jobs

Time to create those instances

First count how long it takes to enqueue the jobs:

Enqueue jobs

Time to create those instances

First count how long it takes to enqueue the jobs:

- Enqueue 1 job $\rightarrow 0.2s$

Enqueue jobs

Time to create those instances

First count how long it takes to enqueue the jobs:

- Enqueue 1 job $\rightarrow 0.2s$
- Enqueue jobs sequentially $\rightarrow 10 jobs/sec$

Enqueue jobs

Time to create those instances

First count how long it takes to enqueue the jobs:

- Enqueue 1 job $\rightarrow 0.2s$
- Enqueue jobs sequentially $\rightarrow 10 jobs/sec$
- Enqueue jobs in parallel

Enqueue jobs

Time to create those instances

First count how long it takes to enqueue the jobs:

- Enqueue 1 job $\rightarrow 0.2s$
- Enqueue jobs sequentially $\rightarrow 10 jobs/sec$
- Enqueue jobs in parallel

Enqueue jobs

Time to create those instances

First count how long it takes to enqueue the jobs:

- Enqueue 1 job $\rightarrow 0.2s$
- Enqueue jobs sequentially $\rightarrow 10 jobs/sec$
- Enqueue jobs in parallel $\rightarrow 10 jobs/sec$

Enqueue jobs

GanetiRAPIClient

- Powered by `PycURL` $\rightarrow$ `libcurl`

Enqueue jobs

GanetiRAPIClient

- Powered by `PycURL` → `libcurl`
- Does not work well with `gevent`

Enqueue jobs

GanetiRAPIClient

- Powered by `PycURL` → `libcurl`
- Does not work well with `gevent`
- Trivial client implementation using `Requests`
    - Powered by `urllib3`
    - Greenlet-safe
    - HTTP connection pooling

Enqueue jobs

GanetiRAPIClient

- Powered by PycURL $\rightarrow$ libcurl
- Does not work well with gevent
- Trivial client implementation using Requests
  - Powered by urllib3
  - Greenlet-safe
  - HTTP connection pooling

- Enqueue 1 job $\rightarrow 0.2s$
- Enqueue jobs sequantially $\rightarrow 10 jobs/sec$
- Enqueue jobs in parralel $\rightarrow 10 jobs/sec$
- Enqueue jobs with green client $\rightarrow 33 jobs/sec$

Enqueue jobs

GanetiRAPIClient

- Powered by `PycURL` → `libcurl`
- Does not work well with `gevent`
- Trivial client implementation using `Requests`
  - Powered by `urllib3`
  - Greenlet-safe
  - HTTP connection pooling

- Enqueue 1 job → $0.2s$
- Enqueue jobs sequantially → $10 jobs/sec$
- Enqueue jobs in parralel → $10 jobs/sec$
- Enqueue jobs with green client → $33 jobs/sec$

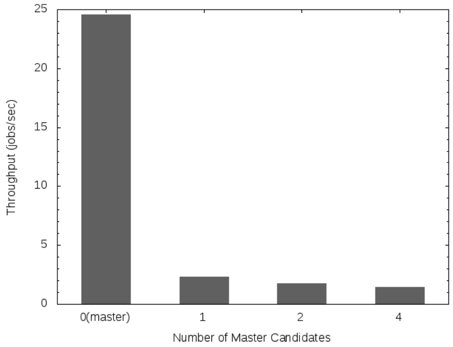
But, on a different Ganeti cluster enqueuing jobs was much slower. Why?

Master Candidates

- Number of master candidates affects job submission rate
- Ganeti writes job to disk and then concurrently replicates it to master candidates

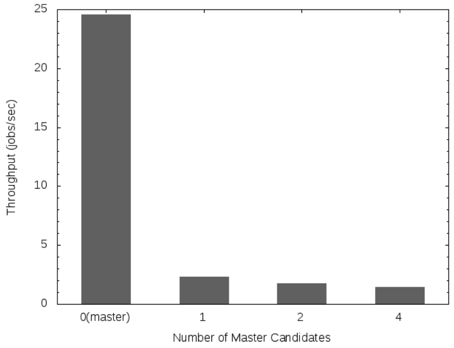
Master Candidates

- Number of master candidates affects job submission rate
- Ganeti writes job to disk and then concurrently replicates it to master candidates



Master Candidates

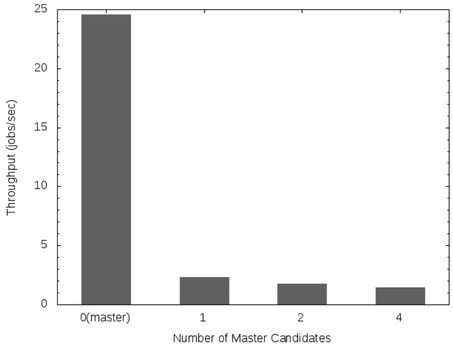
- Number of master candidates affects job submission rate
- Ganeti writes job to disk and then concurrently replicates it to master candidates



- With $1 MCs \rightarrow \approx 7$ times slower

Master Candidates

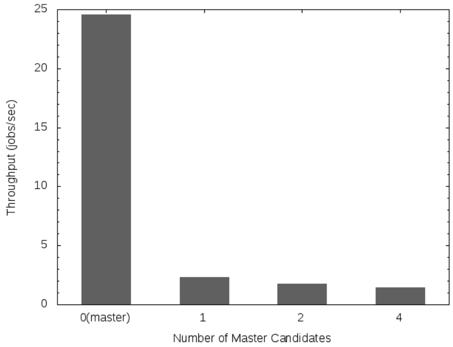
- Number of master candidates affects job submission rate
- Ganeti writes job to disk and then concurrently replicates it to master candidates



- With $1MCs \rightarrow \approx 7$ times slower
- $> 2MCs \rightarrow$ no significant performance drop

Master Candidates

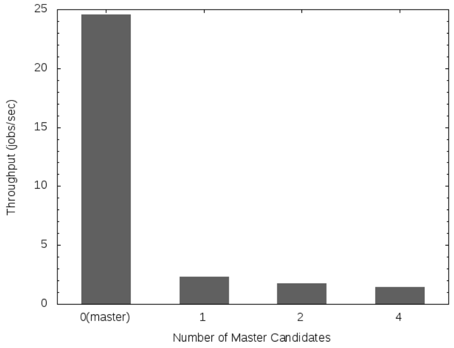
- Number of master candidates affects job submission rate
- Ganeti writes job to disk and then concurrently replicates it to master candidates



- With $1 MCs \rightarrow \approx 7$ times slower
- $> 2 MCs \rightarrow$ no significant performance drop
- Job queue directory in SSDs + MCs in non vm_capable nodes

Master Candidates

- Number of master candidates affects job submission rate
- Ganeti writes job to disk and then concurrently replicates it to master candidates



- With $1MCs \rightarrow \approx 7$ times slower
- $> 2MCs \rightarrow$ no significant performance drop
- Job queue directory in SSDs + MCs in non vm_capable nodes
- Asynchronous replication?

Job execution phases

Jobs enqueued! Awaiting for completion...

Job execution phases

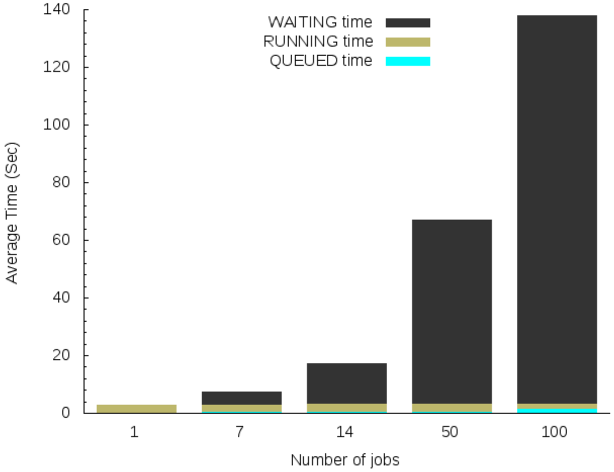
Jobs enqueued! Awaiting for completion...

- Used a Ganeti cluster of 7 vm_capable nodes
- Minimized running time (no-install, 1G file disk) $\approx 3sec$

Job execution phases

Jobs enqueued! Awaiting for completion...

- Used a Ganeti cluster of 7 vm_capable nodes
- Minimized running time (no-install, 1G file disk) $\approx 3sec$



Locking

- start/stop/modify/.. $\rightarrow$ lock instance

Locking

- start/stop/modify/.. $\rightarrow$ lock instance
- remove/migrate/.. $\rightarrow$ lock primary/secondary node

Locking

- start/stop/modify/.. $\rightarrow$ lock instance
- remove/migrate/.. $\rightarrow$ lock primary/secondary node
- create:
  - allocate instance $\rightarrow$ lock all nodes
  - afterwards $\rightarrow$ lock primary/secondary node
  - Use opportunistic locking($>=$ *Ganeti* 2.7)

Locking

- start/stop/modify/.. $\rightarrow$ lock instance
- remove/migrate/.. $\rightarrow$ lock primary/secondary node
- create:
  - allocate instance $\rightarrow$ lock all nodes
  - afterwards $\rightarrow$ lock primary/secondary node
  - Use opportunistic locking($>=$ *Ganeti* 2.7)
- verify cluster $\rightarrow$ lock all nodes
  - Use nodegroups

Real-World Scenario

100 instances: 2 CPU, 4 GB RAM, 40GB Disk
2 x Ganeti cluster of 7 nodes

Real-World Scenario

100 instances: 2 CPU, 4 GB RAM, 40GB Disk
2 x Ganeti cluster of 7 nodes

- drbd: $\approx 14.3min$
- archipelago: $\approx 4.1min$

Ongoing Work

- Based on "[RFC] Distributed Database Config Storage"

Ongoing Work

- Based on "[RFC] Distributed Database Config Storage"
- Store job queue and config data in a distributed `NoSQL` database (`CouchDB`)

Ongoing Work

- Based on "[RFC] Distributed Database Config Storage"
- Store job queue and config data in a distributed `NoSQL` database (`CouchDB`)
- config.data: JSON-serialized file
  - Is read once from file and then kept in memory
  - Is written to file in every update
  - Updating config file can be slow when the size of clusters increases
  - Why not break it into instances, nodes, ...

Ongoing Work

- Based on "[RFC] Distributed Database Config Storage"
- Store job queue and config data in a distributed `NoSQL` database (`CouchDB`)
- config.data: JSON-serialized file
    - Is read once from file and then kept in memory
    - Is written to file in every update
    - Updating config file can be slow when the size of clusters increases
    - Why not break it into instances, nodes, ...
- Implemented as optional mode of operation

Ongoing Work

- Based on "[RFC] Distributed Database Config Storage"
- Store job queue and config data in a distributed `NoSQL` database (`CouchDB`)
- config.data: JSON-serialized file
  - Is read once from file and then kept in memory
  - Is written to file in every update
  - Updating config file can be slow when the size of clusters increases
  - Why not break it into instances, nodes, ...
- Implemented as optional mode of operation
- Replication in DB, easier master failover, ...

Thank you.

Questions?