Dynamic partitioning con LSF per Multicore

Stefano Dal Pra

INFN–T1

stefano.dalpra@cnaf.infn.it

CCR, 26/05/2015

Sommario

Multicore e cluster HEP

Motivazione

- Col Run–2 di LHC Gli esperimenti stimano di avere in produzione $\sim$ 50% di job multicore (8 slot, nello stesso host).
- Rischio starvation: non ci sono mai 8 slot liberi nello stesso host
- Hostgroup dedicato: Host inutilizzati se non ci sono abbastanza job multicore, sottodimensionato se ce ne sono troppi.
- Dynamic Partition: set dinamico di host dedicati a multicore, seguendo la richiesta.
  Ogni host aggiunto va in *Draining* $\rightarrow$ slot inutilizzati.

Configurazione al T1

194 WN, da 24 e 16 slots

- $132 \times 24 + 62 \times 16 = 4160$ slots, 45KHS06
- 4 rack; variabili a piacere

Utilizzo

- Dynamic Partitioning: numero di nodi dedicati a mcore varia secondo necessità.
- Job m–core ($m = 8$ slot) e himem (2 slot)
- max 4 himem per nodo
  - A regime un WN 24 slot ha 2 mc + 4 hm oppure 3 mc
  - i job hm permettono di ridurre l'inutilizzo dei cores dovuto alle fasi di Drain.

Come funziona

Componenti e logica

- script: elim, esub, director, in python; due programmi ausiliari in `C`, un conf. file
- elim: gira in ogni WN disponibile per mcore, pubblica un flag di stato `mcore==0/1`
- esub: gira ad ogni esecuzione di `bsub`, riconosce e modifica *tutti* i job:
  - i job multicore richiedono WN con `mcore==1`
  - i job singlecore richiedono WN con `mcore!=1`
- director: gira ogni 6 min in un nodo, decide chi entra e chi esce dalla partizione, logga lo stato (per monitoring e accounting).
- `nodeinfo.txt; badhosts.txt`: Potenza HS06, num. cores, slots; elenco host chiusi.

Transizioni

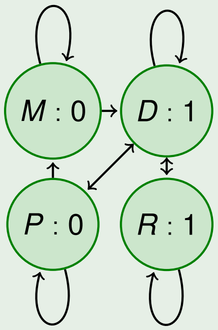
assegnare i WN alla partitione mcore

Gli stati dei nodi

I WN passano tra questi insiemi:

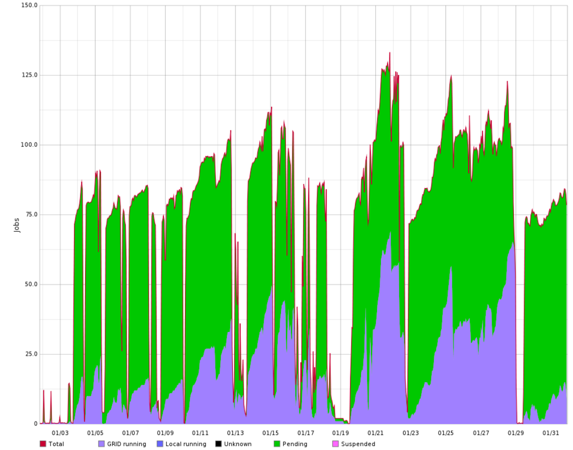
- *M*: disponibile per mcore
- *D*: assegnato a mcore
- *R*: solo job mcore in run
- *P*: tolto da mcore

Mappa delle Transizioni



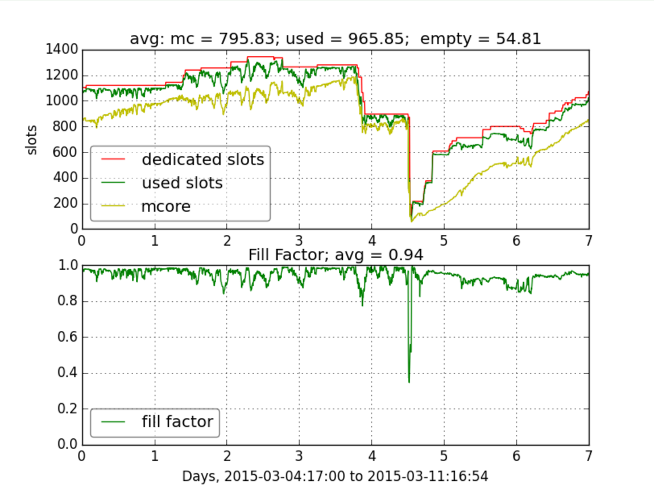
Dynamic partition
mcore queue activity

Multicore running and pending Jobs, Gen 2015



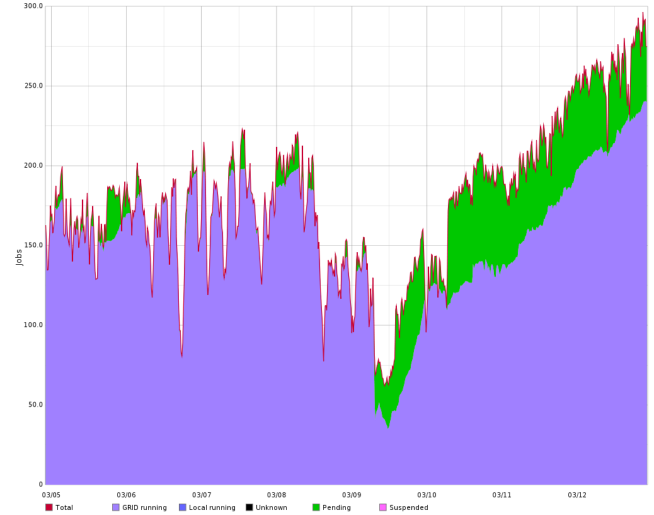
Mcore jobs (Mar 2015)

Mcore partition, 7 days



Himem jobs (Mar 2015)

HIMEM, 7 days



Configurazione, repository

Configurazioni (JSON syntax)

```
"mcore_groups":  ["rack20603", "rack20501"],
"badhosts_fn":  "badhosts.txt",
"log_fn":  "mcore.log",
"log_dbg":  "mcore_act.log",
"hist_fn":  "mcore_hist.json",
"max_hostdrain":18,
"max_emptyslots":157,
"max_empty_ratio":0.3,
```

git repository, con script e guide

- https://baltig.infn.it/dalpra/lsf_multicore_dynamic_partition/
- https://indico.cern.ch/event/304944/session/9/contribution/455

Applicazione del Dynamic partitioning con LSF per provisioning di risorse cloud

Stefano Dal Pra
Vincenzo Ciaschini
Luca dell'Agnello

INFN–T1
stefano.dalpra@cnaf.infn.it

CCR, 26/05/2015

Problem, usecase, motivation

- The whole INFN-T1 farm ($\sim$ 15000 cores) is currently accessible as a “traditional" Grid resource (CREAM Computing Element, LSF Batch System)
- Problem: We would like to be able to dedicate hardware resources to Cloud Computing for HEP purposes in a flexible and reversible manner.
- Use cases:
  - A VO may want to dedicate a certain amount of computing power to a “cloud computing campaign”, then move back the resources to Grid.
  - A VO may want to perform a “smooth migration” from Grid to cloud, moving resources a few at a time.
  - A team may need interactive usage of computing resources.

Shares

Shares in the Grid farm must be adjusted, so that:

- Any experiment moving *k* WN from Grid to Cloud, should have its share in LSF reduced accordingly.
- Any experiment not using cloud resources, should not be affected by the reduced power of the Grid farm.

Wall–clock Time

An overall Wallclock–Time must be accounted, by adding two components:

- Grid–side, the Wall–clock time is accounted per–job, as usual.
- Cloud–side, the Wall–clock time is accounted per–node

Exploiting a solution: dynamic partitioning

A dynamic partitioning mechanism has been deployed at INFN-T1 for the provisioning of multi–core resources. The same technique can be adapted to achieve a Cloud partition.

- The Cloud partition can grow or shrink on a per–need basis (Elasticity).
- On each node, both LSF and Openstack daemons are active. Only one or the other mode can be enabled at a time.
- A Draining phase is needed before moving from a partition to the other
- When a WN is assigned to the Cloud partition, LSF stops dispatching jobs to it (*Draining*). Then it becomes available to the Cloud Controller.

The implementation

- elim script. It runs on the WN and defines the value of the `dcloud` flag.
- esub script. It is executed at the submission host for each submitted job, enforcing a request for nodes having a resource `dcloud!=1`.
- director script. implements the logic of the partitioning model. It runs at regular times on a master node and selects which WNs or CNs are to be moved from the partition they belong to.

The Partition Director

- Implemented as a finite state machine
- LSF side:
  - manages the status of the `dcloud` flag on the nodes. This is achieved by customizing `esub,elim` scripts and enable/disable job dispatching.
- Cloud side:
  - enable/disable scheduling to the CNs (ref. to Openstack, Juno; this is done using api call to `nova-compute`).
  - destroy existing VM on the CN after a timeout (∼24h). This can be achieved thanks to the work done by the WLCG MachineJobFeatures TaskForce.

The Dynamic Partitioning model

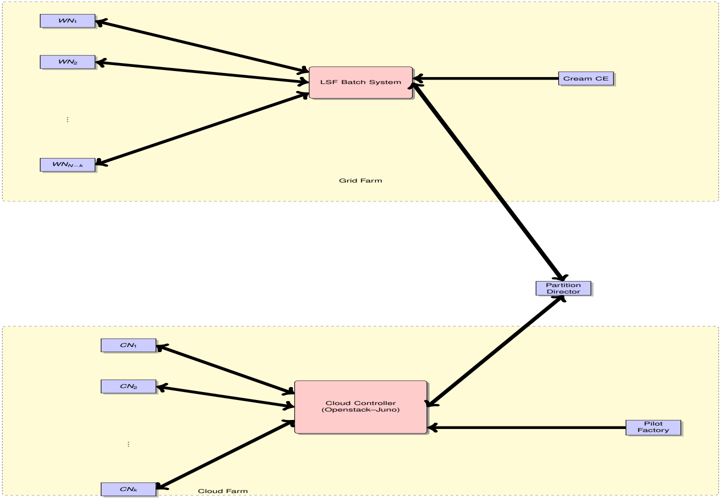


Figure: The partition director triggers role switch of nodes

The finite state machine

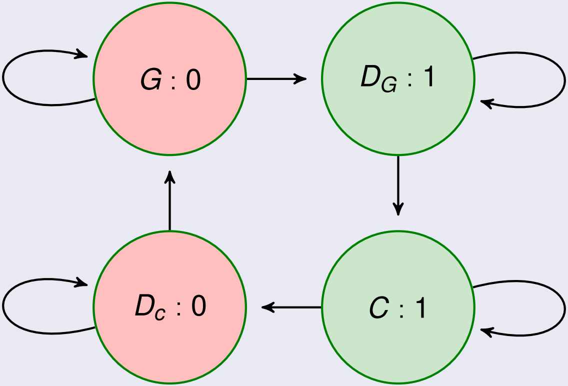


Figure: The Status Transition Map

Dynamic of the dcloud partition

- At $T = 0$, all nodes are $c_i \in G = \{c_1, \ldots, c_N\}$
- When k Compute Nodes are requested, they are moved to Drain from G to $D_G = \{c_1, \ldots, c_k\}$ by the director.
- When the drain finishes, it is moved from D_G to C and becomes available as a Compute Node.
- When a Compute Node $c_i \in C$ must work again as a WN, it is moved to D_C and begins a drain time. The duration can be specified through the `shutdowntime` parameter from the machinejob features.
- When a Compute Node $c_i \in D_C$ expires its `shutdowntime`, Existing VMs are destroyed and the node moves to G.
- The elim script on each node w_i updates its dcloud status:

$$dcloud(w_i) = \begin{cases} 1 & if\ c_i \in D_G \cup C \\ 0 & if\ c_i \in G \cup D_C \end{cases}$$

Driving the partition

Possible approaches

- Admin driven: specyfies number of nodes, ownergroup and direction of the migration, upon request from the experiment.
- User driven: Two alternatives
  - integration with the cloud–scheduler.
  - balancing pending grid jobs vs. rate of cloud resource requests: The higher would set the direction of the role switching. Similar to pilot style: a VM may be unsatisfied just like a pilot job may not get work to do.

Conclusions

- Dynamic partitioning enables cohexistence of Grid and Cloud applications.
- Transition from Cloud–mode to Grid–mode requires to deal with existing VMs after a draining time. User's applications should be *machinejob* aware.