ISTITUTO NAZIONALE DI FISICA NUCLEARE

INFN-14-19/CCR
16th December 2014

JOB PACKING: OPTIMIZED CONFIGURATION FOR JOB SCHEDULING

Stefano Dal Pra[1]

[1] *INFN-CNAF, Viale Berti Pichat, 6/2, I-40127 Bologna, Italy*

Abstract

The default behaviour of a batch system is to dispatch jobs to nodes having the lower value of some load index. Whilst this causes jobs to be equally distributed among all the nodes in the farm, there are cases when different types of behaviour may be desirable, such as having a completely full node before dispatching jobs to another one, or having similar jobs dispatched to nodes already running jobs of the same kind. This work defines the packing concept, different packing policies and useful metrics to evaluate how good the policy is. A simple farm simulator has been written to evaluate the expected impact on a farm of different packing policy. The simulator is run against a sequence of real jobs, whose parameters have been taken from the accounting database of INFN-Tier1. The effectiveness of two packing policies of interest, namely relaxed and exclusive, are compared. The exclusive policy proves to be better, at the cost of unused cores in the farm, whose number is estimated. The possibility of implementing the exclusive policy on a specific batch system, LSF 7.06, is exploited. Relevant configurations are shown and an overall description of the mechanism is presented.

Commissione Calcolo Reti

CCR-49/2014/P

Pubblicato da **SIDS–Pubblicazioni**
Laboratori Nazionali di Frascati

1 INTRODUCTION

1.1 The problem

When dispatching a job, the batch system selects the lesser loaded candidate and adequate node in the computing farm.

Candidate: The node has free computation slots and is adequate

adequate: The node has suitable resources for the job to run

lesser loaded: According to a given metric (usually the system load)

We want to modify the node selection according to one or more known job characteristics, such as its queue, group or other known property.

1.2 Motivation

A number of side effects of a different scheduling policy may be of interest:

Power saving: having jobs *packed* together into a small set of nodes may enable to turn the unused ones to a state of standby.

Isolating risk: at times, a family of unstable jobs may damage the stability of the node where they are running. Healthy jobs in the same node may get damaged in turn. Keeping them together would be of benefit for other jobs.

MPI jobs: MPI jobs would be more efficient if they were dispatched to the smallest possible set of nodes.

WnoDeS: When WNoDeS [2] is installed on top of a batch system, jobs are dispatched to virtual machines, no matter which hypervisor has instantiated them. A packing policy may enable to reduce latencies due to the copy of the physical vm image.

2 PACKING POLICIES

Although a rather wide number of different strategies may be defined, we shall focus our attention on the following:

- PACKING_RELAXED
    - Aggregation: a job with a property C(J==True) or *C–job* should prefer nodes with jobs having the same property already running in it, whenever possible. Jobs *without* the property are dispatched as usual.
- PACKING_EXCLUSIVE
    - Concentration: C–jobs should prefer nodes where other C–jobs are already running in it, whenever possible. However, jobs *without* the property *must avoid* nodes where C–jobs are running.
- PACKING_NONE
    - Spreading: C–jobs should prefer nodes *without* C–jobs running in it.

3 THE FARM SIMULATOR

A simple farm simulator has been built to ease the evaluation of what an impact should be expected from a given packing policy, if adopted on the production farm. This enables for quick and safe evaluation of how a new scheduling policy would impact on a production farm, if adopted.

Two synthetic indicators have been defined in order to measure how well a packing policy works:

- Packing Index: PI = Needed nodes / Used nodes

  This ratio indicates how good C–jobs are packed together (PI @ 1) or spread (PI @ 0)

- Fill factor: FF = Used slots / Available slots

  This ratio, when lower than one, indicates sub–optimal farm utilization, due to unused slots in the farm. This may happen when using the PACKING_EXCLUSIVE policy, which actually realizes a *node reservation*.

The simulator has been implemented with python, using specialized libraries for numerical and graphical applications (matplotlib, numpy).

A dataset of real data have been used to run the simulations, taken from the accounting database of the INFN-Tier1. The only data of interest are: *Start Time, End Time, queue name*. We restrict ourselves to Cj–jobs where the condition is decided by the queue name of the job. The first condition always indicates the default condition, i.e. no special policy at all.

For example: we want to simulate RELAXED_PACKING for jobs having `queuename == ams` and jobs having `queuename == cms`. Then C1 becomes: “the job is neither a `ams` nor a `cms` job”, similarly is: “the job is a `ams` job” and is: “the job is a `cms` job”. When apply to a job it is labeled as belonging to the `other` queue.

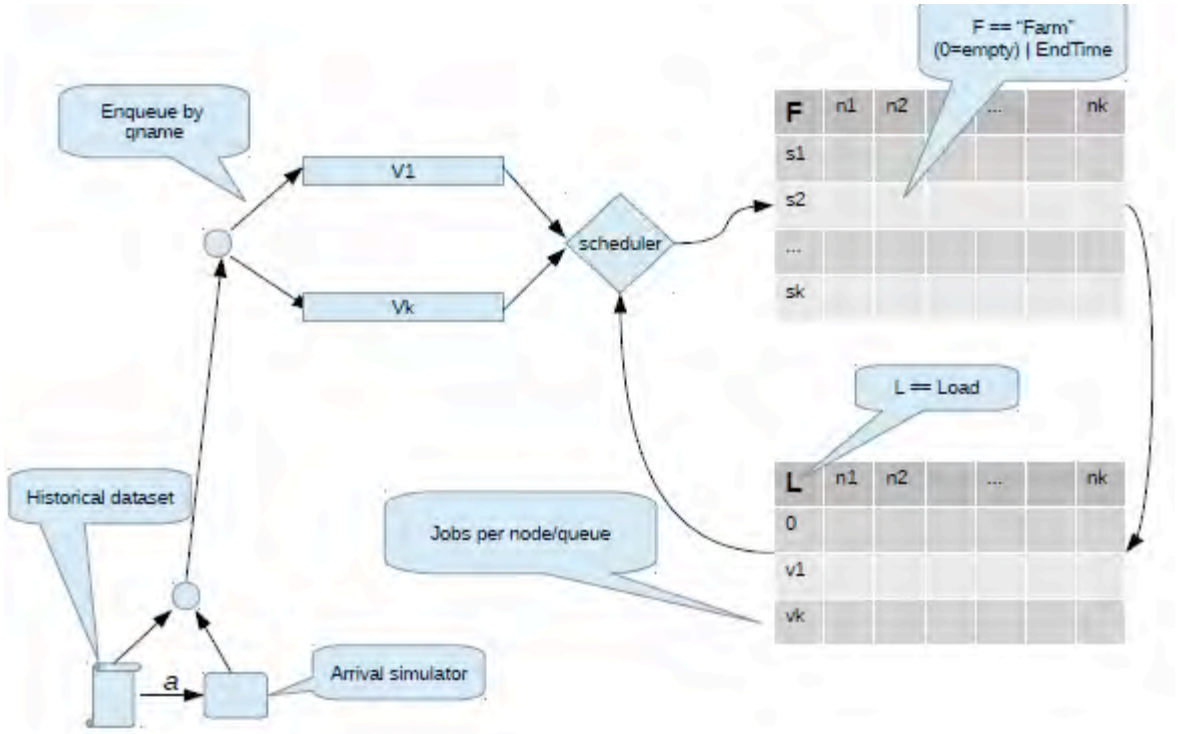

Fig. 1: The farm simulator. Job arrival is taken from an accounting dataset or simulated. V1 enqueues jobs with no special requirements, V2,...,Vk for those requiring a packing policy. The F matrix represents the status of each computation slot, the L matrix represents the "load" status.

3.1 Simulator description

The arrival of jobs is obtained by reading entries from the dataset or emulating them as a random process where each queue name has its own statistical description, modeled after the real data or customly crafted by the user. It must be noted that the timestamps collected in the dataset are not reporting the *submission time*; the absolute Start time (hence after the dispatching) and the End time are available, hence the job duration and, implicitly, the arrival order.

Arriving jobs are routed to the *V1,...,Vk* queues, according to the matching condition *C1,...,Ck* defined by the user.

Although the farm simulator implements a much simplified model of a computational center, it is expressive enough to evaluate the described packing policies. Moreover, implementing new ones is quite simple and adding them to the simulator is straightforward.

The "farm status" that we are interested to track simply comes from the status of every slot on each node at a given time. Assuming that all the nodes are equal, we can represent the farm with a matrix F, having one column per node and one row per slot. The status of the *i*–th slot in the *n*–th node is represented by the $f_{i,n}$ cell of the matrix whose value is zero when free. A job in a cell is simply represented by the timestamp T_e of its end time: $f_{i,n}=T_e$

An auxiliary matrix L represents the "load" of the farm, indicating how many jobs of each family *V1,...,Vk* are running on each node. The *0*–th row holds the sum of the values from the other rows, thus representing the number of busy slots in a node.

The scheduler implements the desired policy. Its status is an internal "clock" used as the absolute time who rules the farm. Each absolute timestamp read from the dataset is adjusted according to the scheduler's time. This occurs because of the different size between the simulated farm and the real one: a smaller farm has a lower throughput, thus the absolute

timestamp from the dataset only makes sense on the real farm where it was collected; of course runtimes and arrival order however maintains consistency.

3.2 Scheduling and dispatching

The simulator dispatches one job per cycle, as follows:

1. the elements where $f_{i,n} > T_s$, are set to zero. These are in fact finished jobs.

2. A job is selected for dispatch. It is the one having the lower start time on the queues *V1,...Vk*.

3. The current policy is enforced: an ordered list of eligible slots is computed and the first one satisfying the constraints is the selected one. If no eligible slots are found:

   - The farm is saturated: there are no empty slots. We need to wait for some job to finish. This is simply done by setting: $T_s = min(F) + 1$, and continuing to step 1.

   - The policy prevents dispatching: the free slot only accepts jobs from a queue $V_l, l > 1$. We check the V_l queue for entries and select from there the job to be dispatched, if possible. Otherwise we move time forward: $T_s = min(F) + 1$ and continue to step 1.

4. The job is dispatched to the selected slot: $f_{i,n} = T_s + R$, with R being the runtime.

The simulation is started by executing a command line, specifying which packing policy is being simulated, which queues are subjected to the packing policy (any other queuename is then renamed to *other*). If reading job arrivals from a histfile, a date time can also be specified, to have the simulation starting with data collected after a given time.

While running, the simulator displays pictorial representions of the farm status. These are updated every iterations, being specified at command line. Optionally, each frame is then saved into a file for subsequent use.

4 SIMULATION RESULTS AND COMMENTS

The simulator have been used to compare the two packing policies *relaxed* and *exclusive*, described in the previos section, applying them on an initially empty farm made of 800 nodes, 8 slots each. The job submission sequence is exactly the same in both cases. The farm status can be represented by a 80x80 matrix. Each row represents 8 consecutive slots of ten consecutive nodes, and each color represents the slot status: free, running a generic job, or a packing one.

At first the dynamic appears to be almost the same: jobs are dispatched uniformly across all the nodes. An early small difference can be observed (Fig. 1) when a single packing job is dispatched.

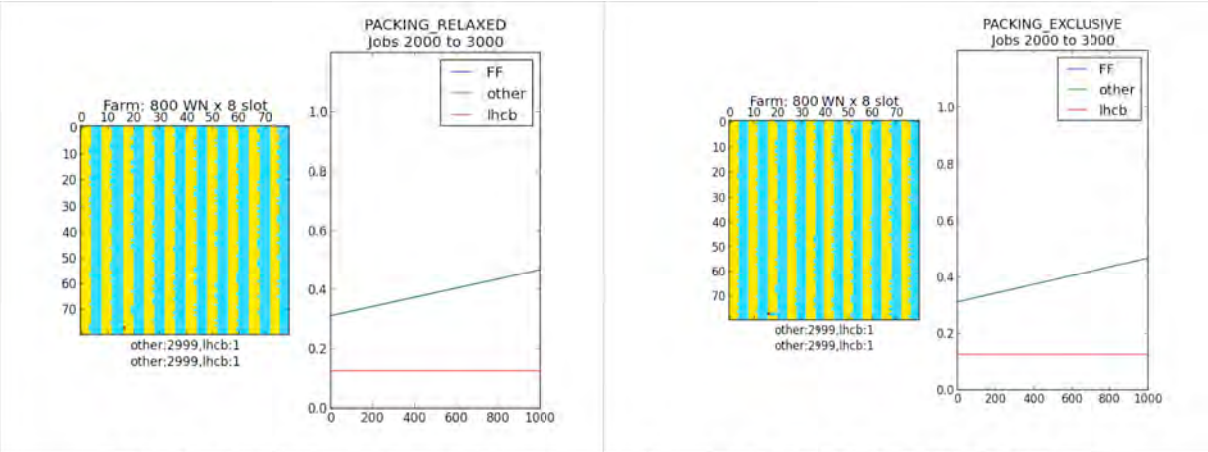

Fig. 2. Simulations starts with an empty farm. In PACKING_RELAXED mode, a packing job (the purple pixel) does not prevent other jobs to be dispatched in the same node. In PACKING_EXCLUSIVE, all the free slots in the node are reserved for other packing jobs only.

The two policies behave almost equally until farm saturation, i.e. until "general purpose" free slots are no more available (Fig 2). From then on, the *relaxed* policy fails in maintaining jobs packed together. This is because no reservation is enforced on the free slots. Conversely, the *exlusive* policy exhibits a satisfactory Packing Index, at the cost of a suboptimal Fill Factor. This means that we have to accept a number of unused slots, i.e. a slower throughput of the farm (Fig 3,4,5,6). The percentage entity of this slowdown can be measured (Fig. 7) by averaging the Fill Factor difference between the two policies and turns out to be around 1%. It must be noted that this values strongly depends on the job arrival sequence. From a theoretical point of view, a specific job arrival sequence could be forged to force a "farm deadlock".

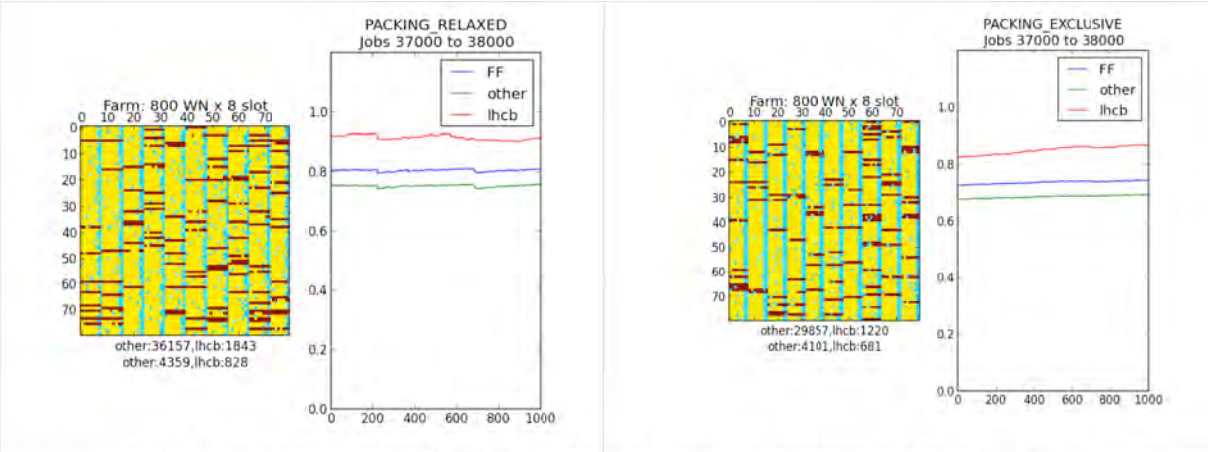

Fig. 3. Before saturation (almost no more free slots available) the two policies still look very similar.

For example, if we have a single free slot on each node, and a pending sequence of exactly one packing job per node, then they would be scheduled one per node, with the result of having reserved every node to packing jobs only. If only non-packing ones are submitted, we turn out with a "locked farm", where no job could be dispatched, because of the *exclusive* policy, until a node ends all of its running jobs. Although this is an extreme example, it helps to make clear how the arrival order matters and how an extreme impact the job submission pattern could affect the overall farm throughput. The deadlock risk can be removed by

introducing a "Time To Live" parameter on the node: a reserved node declares itself "unreserved" after TTL seconds without receiving new packing jobs. It can be noted that when TTL=0 produces the *relaxed* policy, while TTL→∞ gives the *exclusive* one.

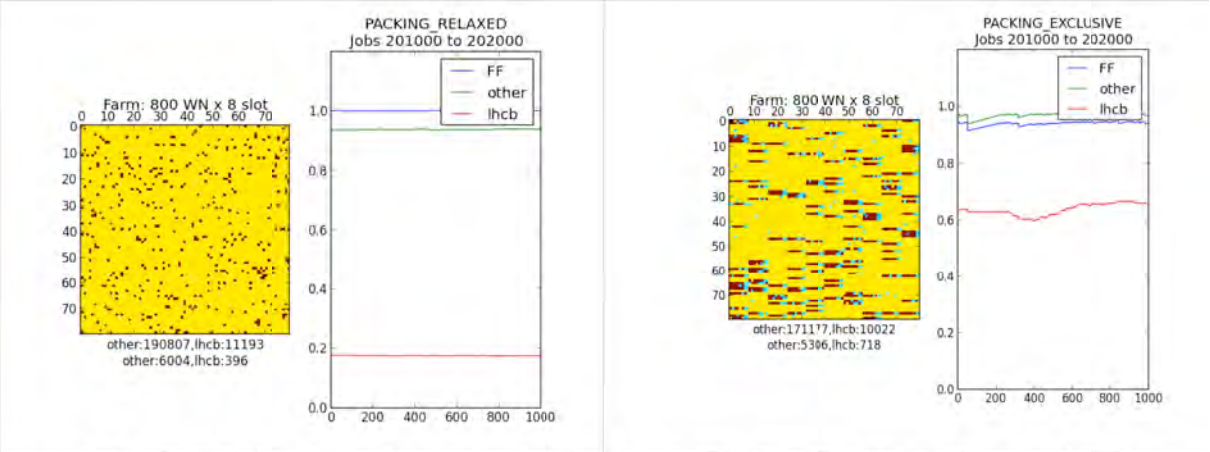

Fig. 4. A little while after saturation has begun, the RELAXED policy clearly misses its goal; packing jobs are radomly spread across the farm and the Packing Index has a poor value. The EXCLUSIVE policy shows better performances, and a much higher P.I. However this comes at the cost of unused slots (Fill Factor < 1).

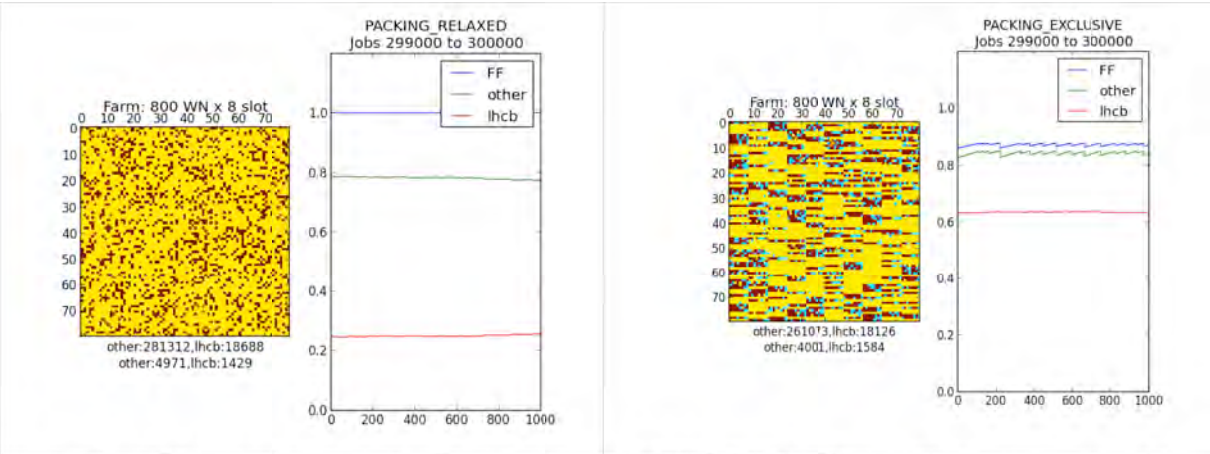

Fig. 5. Long after saturation has begun, the RELAXED policy still shows poor packing performances. The EXCLUSIVE policy still exhibits a good P.I. while remaining with a suboptimal Fill Factor ~= 0.9

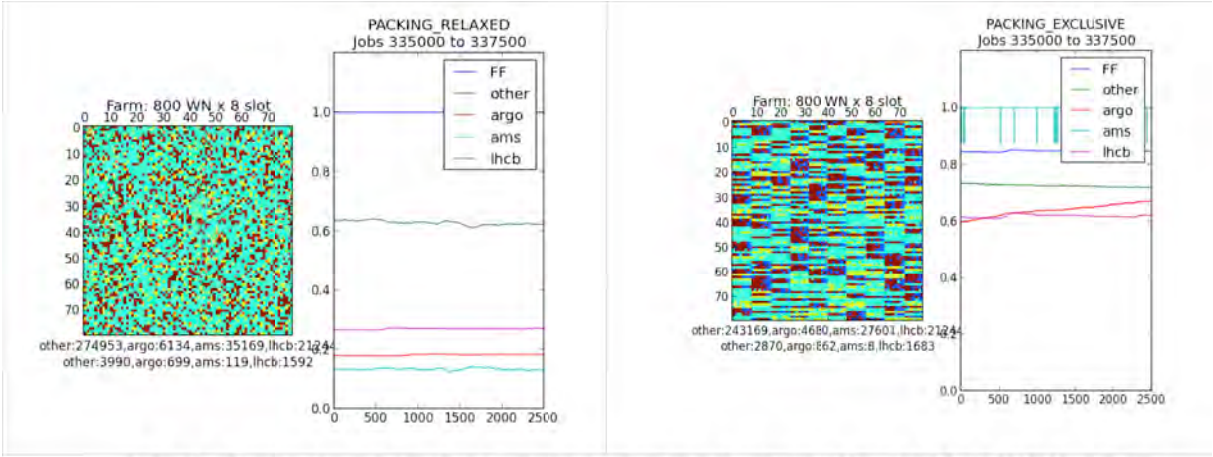

Fig. 6. Simulation with three distinct packing families. The EXCLUSIVE policy exhibits quite good P.I. and F.F.

5 IMPLEMENTATION

This section describes how the policies can be implemented on a real scheduler. We specifically refer to LSF 7.06 as this is the production batch system at INFN-T1.

The overall configuration follows three main steps:

1. Define one or more custom dynamic resource. These are an external load indexes, whose meaning and behaviour is configured by the LSF administrator.
2. Write an *elim* script. This will run on each node in the farm, cyclically reporting the value of the aforementioned load index to the Load Information Manager (LIM).
3. Write an *esub* script. This will run on the submission host at submission time for each job. It will add Resource Requirements based on the custom load indexes, thus applying the "packing policy" as needed by that specific job. These will be then evaluated bu the scheduler at dispatch time, influencing the host selection. A "packing job" is recognized by its usergroup.

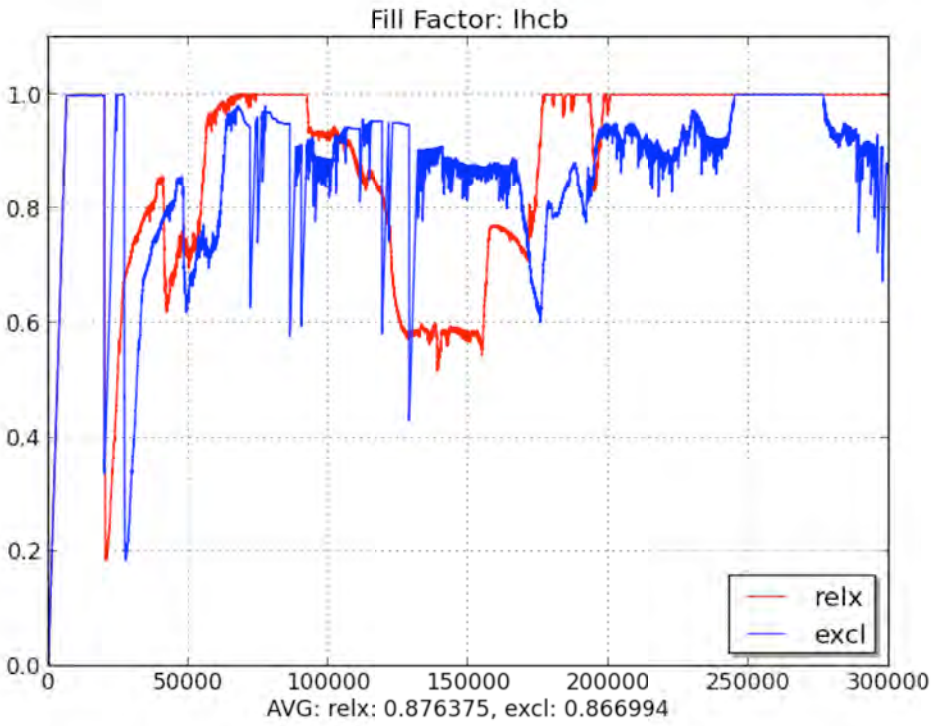

Fig. 7. A farm providing exclusive packing has a lower throughput. The difference can be measured by comparing the Fill Factor of the two policies and averaging the difference: avg(ff_relx – ff_excl) * Numslots ~= 0.01 * 6400 = 60.

5.1 Defining External Load Index

Two configuration files are to be edited

5.1.1 lsf.shared

Begin Resource

RESOURCENAME TYPE INTERVAL INCREASING DESCRIPTION

two resources to exploit Job Packing

pkyes Numeric 15 Y (Pack)

pknone Numeric 15 N (no Pack)

5.1.2 lsf.cluster.<clustername>

Begin ResourceMap

RESOURCENAME LOCATION

pkyes [default]

pknone [default]

The configuration changes, are the activated running: lsadmin reconfig ; badmin mbdrestart.

5.2 Writing the *elim* script

The elim script is executed on each node by the local LIM. It must be located under the `$LSF_SERVERDIR` path and its name must be of the form `elim.`*`name`*.

It runs an endless loop, computing one or more values that are printed at regular times to standard output on a single line. For example:

```
[root@wn-xyz ~]# ./elim.jp
2 pkyes 1 pknone 0
```

The general output format is:

`<number of values> <name_1> <value_1> … <name_n> <value_n>`

The elim script must be able to determine how many jobs are running on the node, how many of them are "packing jobs" and how many are not.
It is important to *not* retrieve the list of the running jobs by querying the batch system. That would generate heavy traffic and excessive load on the Master batch daemon. Since running jobs are launched by the `sbatchd`, they can be retrieved using the `ps` command instead:

ps -o pid --ppid `pidof sbatchd`

Then, the usergroups of the running jobs must be determined. Again, this is achieved with a `ps` command, passing the process id of the running jobs:

ps -o group -p *pid1,...,pidn*

The value for `pkyes` and `pknone` is finally computed by mapping and counting usergroup names to the two external indexes.

5.3 Writing the *esub* script

This is a shell script executed at submission time. It can be written by the batch system administrator to enforce a packing policy to a category of jobs.

Here is a dimostrative example script: jobs submitted to the `pk1` queue are modified to require nodes having the higher possible value for the `pkyes` external index. Any other job will require `pkyes==0`, i.e. nodes without packing jobs.

```
#!/bin/sh
if test "$LSB_SUB_PARM_FILE" != ""
then
  . $LSB_SUB_PARM_FILE
  if [ $LSB_SUB_QUEUE  = "pk1" ] ; then
    eval  echo 'LSB_SUB_RES_REQ=\
"\"select[pkyes > 0 || pkyes == 0]\"" > $LSB_SUB_MODIFY_FILE
  else
    eval  echo 'LSB_SUB_RES_REQ=\
"\"select[pkyes == 0]\"" > $LSB_SUB_MODIFY_FILE
  fi
fi
```

The esub script must be named `$LSF_SERVERDIR/esub.<name>` and it must be declared as LSB_ESUB_METHOD=<name> in lsf.conf.

The configuration changes, are the activated running: lsadmin reconfig ; badmin mbdrestart.

5.4 Checking the setup.

The configuration of the external indexes can be checked before having written the esub script:

```
#find nodes with packing jobs
lsload -I pkone -R "select[pkyes>0 || pkyes==0]"

#find nodes without packing jobs
lsload -I pkyes -R "select[pkyes==0]"
```

Packing and non-packing jobs can be manually submitted this way

```
bsub -q pk1 -R "select[pkone > 0 || pkone == 0]” 'sleep 3600'
bsub -q pk2 -R "select[pkone == 0]” 'sleep 3600'
```

and the effect can be checked after a while using `lsload:`

```
[root@lsf ~]# lsload  -I pkyes:pknone
HOST_NAME       status      pkyes   pknone
wn-104-03-01-08    ok      0.0     1.0
wn-104-03-01-12    ok      0.0     0.0
wn-104-03-01-06    ok      2.0     0.0
```

6 CONCLUSIONS

A set of scheduling policies has been defined and a family of packing policies has been selected as a case study. To quickly and safely evaluate their impact on a computing farm, a simulator has been built and applied to compare exclusive and relaxed packing policies. The simulation demonstrates how the exclusive policy is more effective, at the cost of a lower throughput of the farm. The average loss of computing power has been estimated by comparing results from the simulator. An example configuration to implement exclusive packing with the LSF batch system has been described and tested. The mechanism adopted for the implementation, based on esub and elim custom scripts and external load index is quite general and more applications may be investigated. One usecase of interest for further study is given by multicore jobs. In this case an exclusive policy should guarantee multicore jobs to have a fair number of nodes with the needed number of free slots for them to run.

7 REFERENCES

(1) http://www-304.ibm.com/support/customercare/sas/f/plcomp/platformlsf.html (Platform LSF admin guide)

(2) **Accessing Scientific Applications through the WNoDeS Cloud Virtualization Framework**
Elisabetta Ronchieri, Marco Verlato, Davide Salomoni, Gianni Dalla Torre, Alessandro Italiano, Vincenzo Ciaschini, Daniele Andreotti, Stefano Dal Pra, Wouter Geert Touw, Gert Vriend, Geerten W. Vuister; **proceeding of: The International Symposium on Grids and Clouds (ISGC), PoS, At Academia Sinica, Taipei, Taiwan**

(3) https://indico.cern.ch/event/220443/session/5/contribution/9 (**Job packing: optimized configuration for job scheduling;** talk, HEPiX Spring 2013 Workshop).