# Adjusting the fairshare policy to prevent computing power loss

**Stefano Dal Pra**

INFN–T1
stefano.dalpra@cnaf.infn.it

CHEP 2016, October $x^{th}$

# INFN–T1 Farm

# Introduction

## INFN–T1

- main WLCG computing centre in Italy
- serving the 4 LHC and $\sim$ 25 minor experiments
- $\sim$ 1000 physical WN, $\sim$ 21500 computing slots
- IBM / Platform LSF 9.1.3 Batch system

## Usage

- Grid and local users in HEP and other physics communities
- There are **always** pending jobs (no spare time)
- Several different (competing) requirements and workloads
- Quite large cluster, tuning and optimization matters.

# Reasons to investigate short jobs

## Short jobs and Unusable cputime

- Let $w$ be **turnover time** between consecutive jobs on a computing slot.
- During this time the slot is **unusable**
- The number $N$ of such timelapses over a time window $T$ yields the average number of unusable slots:

$$U = \frac{1}{T} \sum_{n=1}^{N} w_n$$

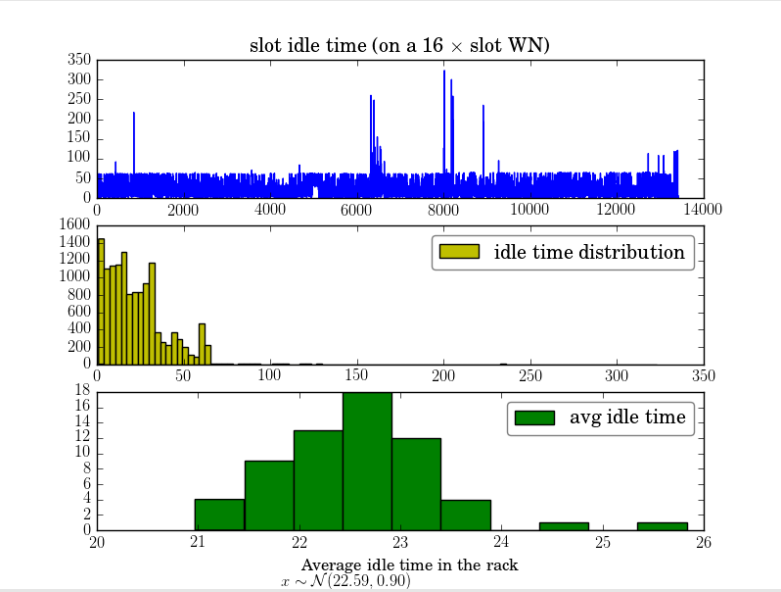
- $U$ grows with bigger clusters and shorter jobs.
- A job is *short* when $WCT_j \ll E[WCT]$ (mins vs hours)

# Estimating $w$



Investigating the time to fill the only free slot in a WN, whenever a single–core job ends on a full WN.

## slot turnover time $w$




## Time to fill latest slot

- average on 16 slot WNs
  - $\rightarrow 0 < w < 60$
  - $\rightarrow E[w] \backsimeq 22sec$
- average over different WNs
  - $\rightarrow 21 < E[w] < 26sec$
  - $\rightarrow \sigma_w \backsimeq 25sec$

# fairshare and short jobs

## Dynamic Priority

- Each user has a **Dynamic Priority**. Pending jobs of users with higher DP are dispatched first.
- Prevents job starvation and underutilization of resources.
- The user's DP is continously updated by the **fairshare formula**:

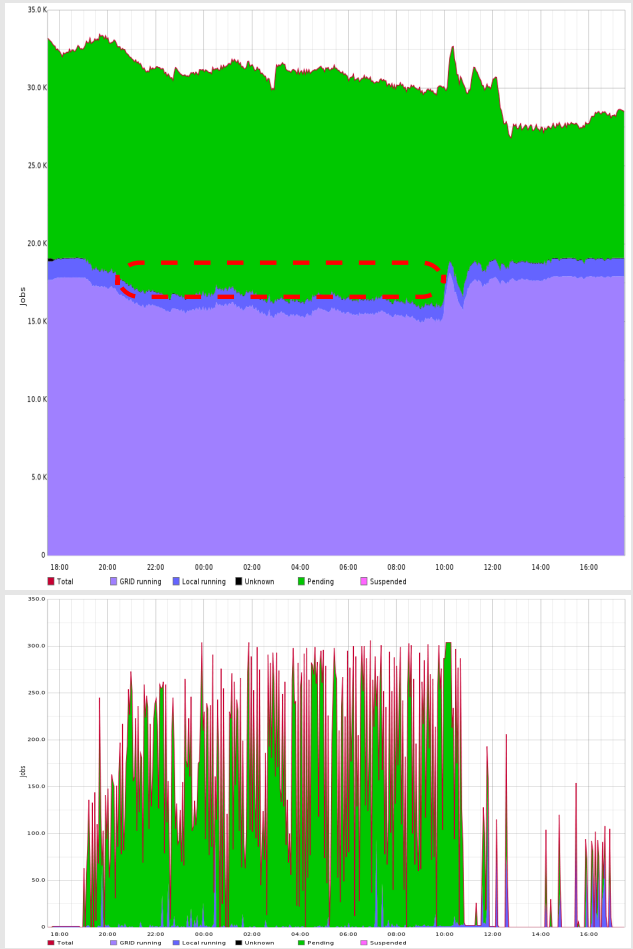
$$U_{prio} = \frac{U_{share}}{\varepsilon CPT + \alpha WCT + \beta(1 + SLOTS) + \gamma ADJUST}$$

- Usually, $\alpha \gg \varepsilon$, $ADJUST = 0$, $U_{prio}$ driven by $WCT$
- short jobs contribute negligible $CPT$ and $WCT$
  - $\rightarrow$ user's priority does not decrease
  - $\rightarrow$ more jobs of the same user are dispatched at next round

# job submitters, short jobs, flooding



## Short job flooding




## types of short jobs

- local jobs running few seconds to few minutes.
- broken jobs, submitted by unaware user.
- empty pilots (Grid users)

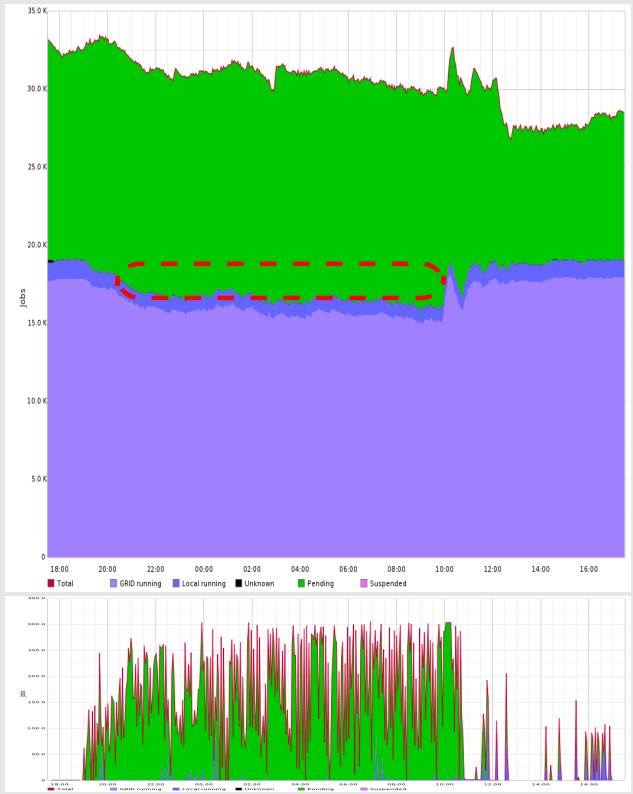
## submitters

- several custom job submitters.
- Popular strategy: *keep a steady number of pending jobs*
- risk of short jobs flooding!

# Issues with short jobs



## Short job flooding




## Events and actions

- 8PM, short jobs ($\sim 1$ sec) flow begins. Total running drops by $\sim 2K$ slots.
- 10AM, close the user's queue.
- 10:30, open the user's queue, ban the user.
- 11AM, enable fairshareadjust.

# Mitigating the problem



## At userland side

- Encourage users to perform multiple executions in a single job submission
- provide example submitter scripts to do so

## Batch system side

Need to be more robust against short job flooding

- Add sleep time on the post exec scripts
  - → sleep time accounted to user :(
  - → We add our own inefficiency
- temporarily inactivate submission from the user's queue
  - → impact on all queue users :(
- **Customize the fairshare formula** to add "missing WCT"

# Customize the fairshare formula



## Customize ADJUST in the fs formula

- add a run time penalty to short jobs
- treat short jobs as if running a **minimum fixed time**.
- The DP of the submitter would then decrease accordingly
- This would act like a “submission rate limiter”.
- Accounting remains unaffected

## Adjust factor

- The runtime penalty can be added by customizing the `fairshareadjust C` function.
- It returns the ADJUST value for the fairshare formula
- invoked at each scheduling cycle for each known user and group in the LSF cluster

# Implementation



## Problems

- The function is invoked very frequently
- Needed data (done jobs per user) are out of scope
- computing values inside the function is not an option.

## Solution

- Number of sh jobs per user is **retrieved externally** by a python script and updated to a **ramdisk filesystem** every 3 min.
- `fairshareadjust( )` reads data from ramdisk into a lookup table and returns the ADJUST value

# algorithm

## python: `dj_stats.py`, at time $t$

- computes $N_s(t)$ and $T_s(t)$ penalty per user/group (holder)
- load previous status $A(t-1)$ from ramdisk, then update:

$$A(t) \leftarrow \lambda A(t-1) + (1-\lambda)T_s(t)\ ;\ \lambda = 0.9$$

- dump $holder : A_u(t)$ map to ramdisk as a `C struct` lookup table `lkt`

## `fshareadjust(holder)`, when invoked by LSF

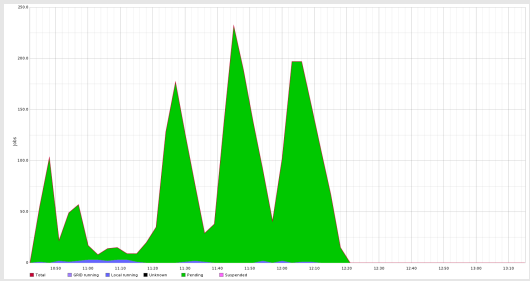
- load lookup table `lkt` from ramdisk
- returns $A(t)$ $\leftarrow$ `bsearch(holder,lkt);`
- if error or not found, returns 0.0

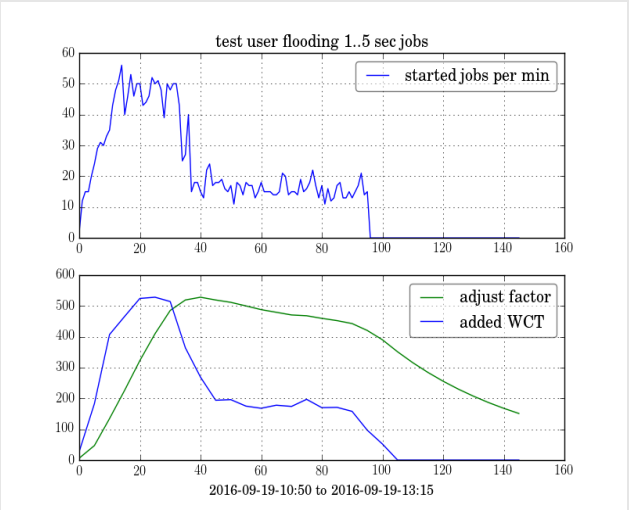
# Effect of fairshare ADJUST (test)



$$U_{prio} = \frac{U_{share}}{\varepsilon CPT + \alpha WCT + \beta(1 + SLOTS) + \gamma ADJUST}$$

## Short job flooding test






## Test user with high share

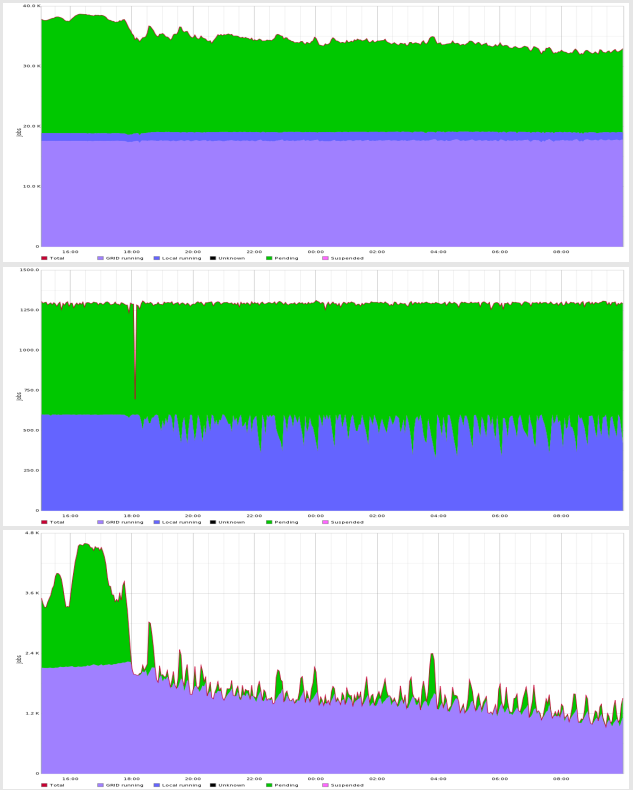
- High dispatch rate at first
- → Penalty WCT $T_s(t)$ grows
- → ADJUST $A_u(t)$ follows
- → subm. rate hardly cope with disp. rate
- User's dyn. prio. drops
- → dispatch rate stabilizes
- → submission rate reduces
- → $A_u(t)$ decays after submission flow ends

# Effect of fairshare ADJUST (Production)

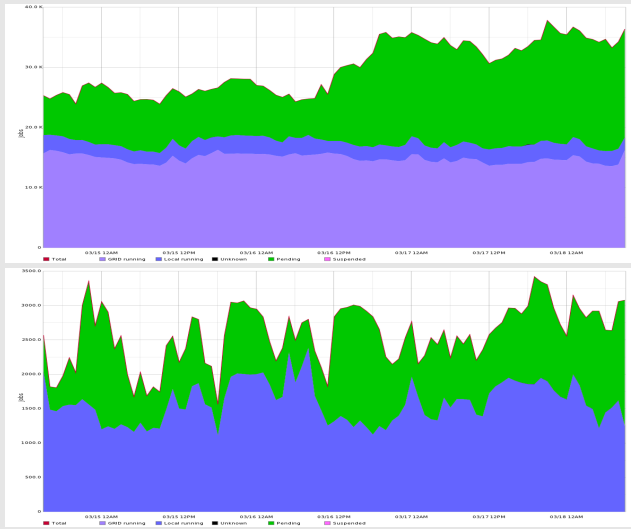
All + 2 sj submitters, 30% of recently done are short

All + one sj submitters, 50% of recently done are short

## Sep 30, with ADJUST



## March 2016, no ADJUST



→ Sep 30: $avg(r) > 21K$

→ March: $avg(r) \sim 18.5K$

# Summary

- High submission rate of short jobs can significantly decrease the number of usable computing slots in the cluster.
- The problem can be mitigated by adding a “minimum fixed runtime” to finished short jobs.
- This prevents “black hole” effect and improves the behaviour of the dynamic priority as implemented by the fairshare policy.
- The implemented solution is specific to LSF, however the general problem and the adopted strategy should be generic enouugh to be extensible to other batch systems too.