CAF Benchmarking

Marco MEONI

CERN - Offline Week

Outline

- SpeedUp test: scalability.
- Cocktail test: usability.
- Dataset test: staging capability.
- CPU quota: fairshare.

Evaluation of PROOF

- 40 machines, 2 CPUs each, 200 GB disk
- DEV and PRO clusters
- Test suite (proofsession.C) developed by Jan Fiete

I

SpeedUp Test

Aim

- Scaled speedUp estimates how much faster parallel execution is over same computation on single workstation
- Assumes problem size increases linearly with number of workers
- Sub-linear, linear or super-linear (if different algorithms or cache effect)

Performance and Scalability Issues

- Parallel overhead: workers creation, scheduling, synchronization. Can impact scalability and provoke high kernel time: keep reusable workers and pool
- Granularity: too few/much parallel work. A higher number of workers not always increases performance and efficiency. System must be adaptive.
- Load imbalance: improper distribution of parallel work
- Difficult debugging: not always easy to debug if the complexity of the system increases (data distribution, deadlocks...)

Amdahl's Law

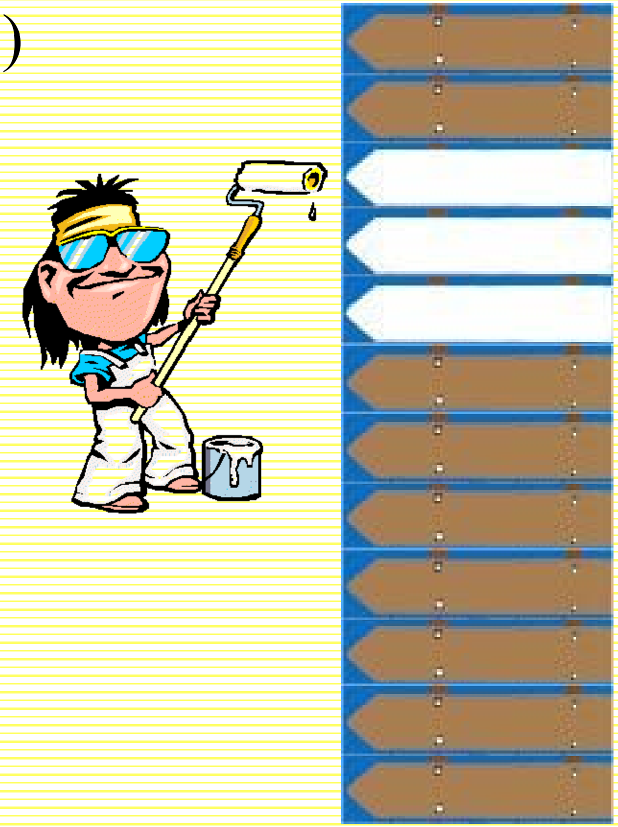
- SpeedUp: $F(n) = 1 / (1 – p + p/n)$
- Efficiency: $E(n) = F(n) / n$

p=parallizable code
n=number of workers

Example: painting a fence (300 pickets)

1. 30 min preparation (serial)
2. 1 min to paint a single picket
3. 30 min of cleanup (serial)

Painters	Time	Speedup	Efficiency
1	360 = 30 + 300 + 30	1.0x	100%
2	210 = 30 + 150 + 30	1.7x	85%
10	90 = 30 + 30 + 30	4.0x	40%
100	63 = 30 + 3 + 30	5.7x	5.7%
∞	60 = 30 + 0 + 30	6.0x	low



Parallel/Serial tasks in PROOF

- Parallel code:
  - Creation of workers
  - Files validation (workers opening the files)
  - Events loop (execution of the selector on the dataset)
- Serial code:
  - Initialization of PROOF master, session and query objects
  - Files look up
  - Packetizer (file slices distribution)
  - Merging (biggest task)

SpeedUp Parameters

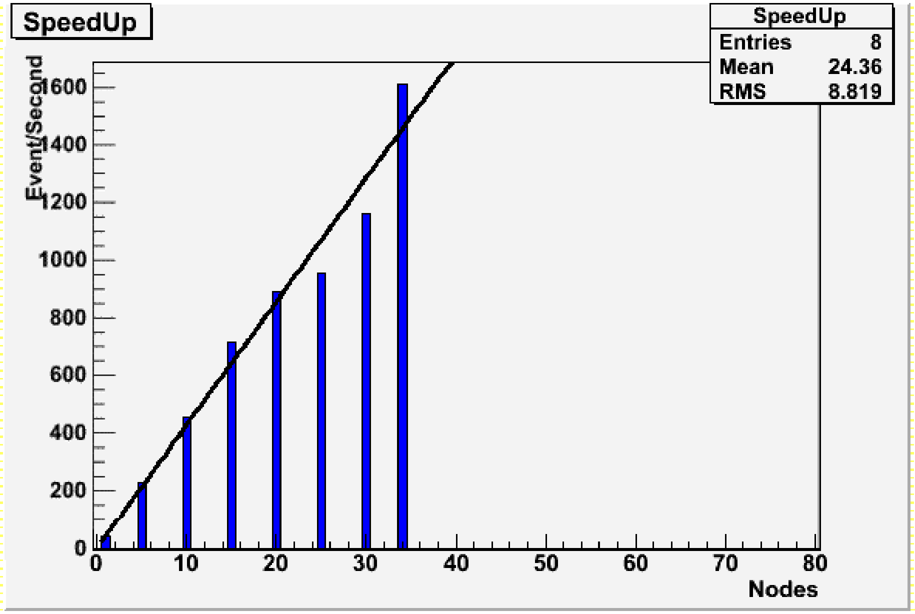
- The test runs 8 times a sample selector with a number of proportionally increasing parameters:

Workers	Input Files	#Events
1	8	16.000
5	40	80.000
10	80	160.000
15	120	240.000
20	160	320.000
25	200	400.000
30	240	480.000
33	272	544.000

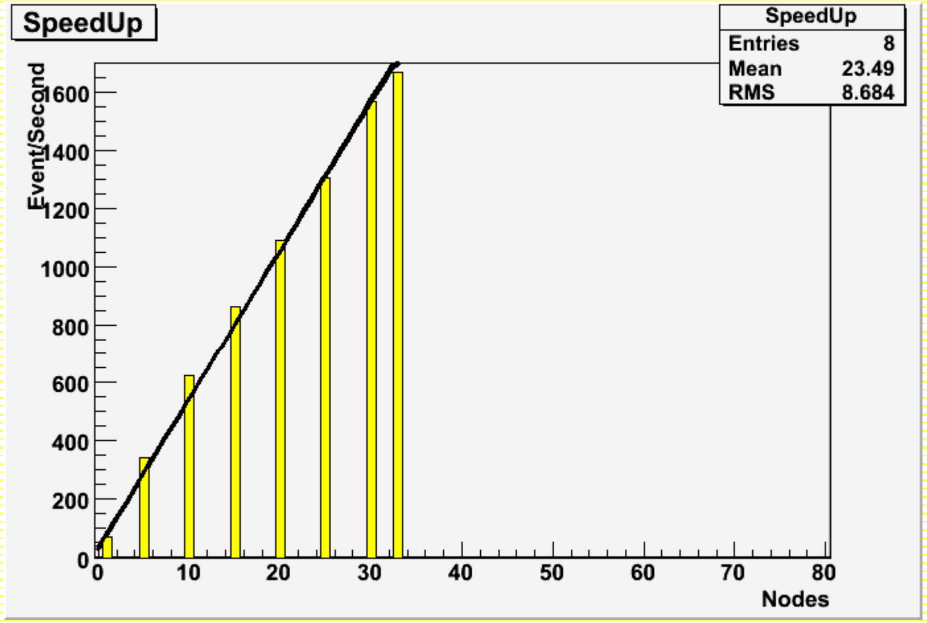
- Average of 16.000 events processed at each worker node

Comparison

February 2007



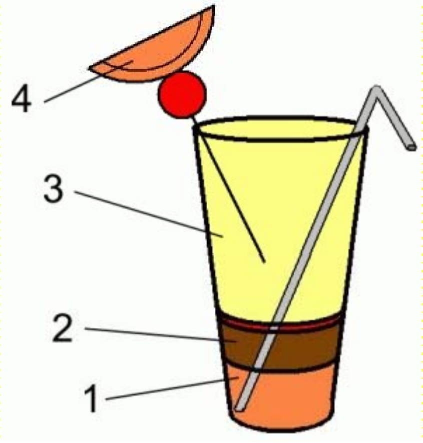
September 2007



- Same Selector
- Same input files per each query
- Same hw/memory configuration
- Same ROOT profile (debug/head)

- Adaptive packetizer improved for unifom datasets distribution
- 1.6 factor slower in debug version

II
Cocktail Test



Aim

- A realistic stress test consists of different users that submit different types of queries (10 max workers per each user)
- 4 different query types
- Tuned to run the four query types at the same time for 2 hours in a row

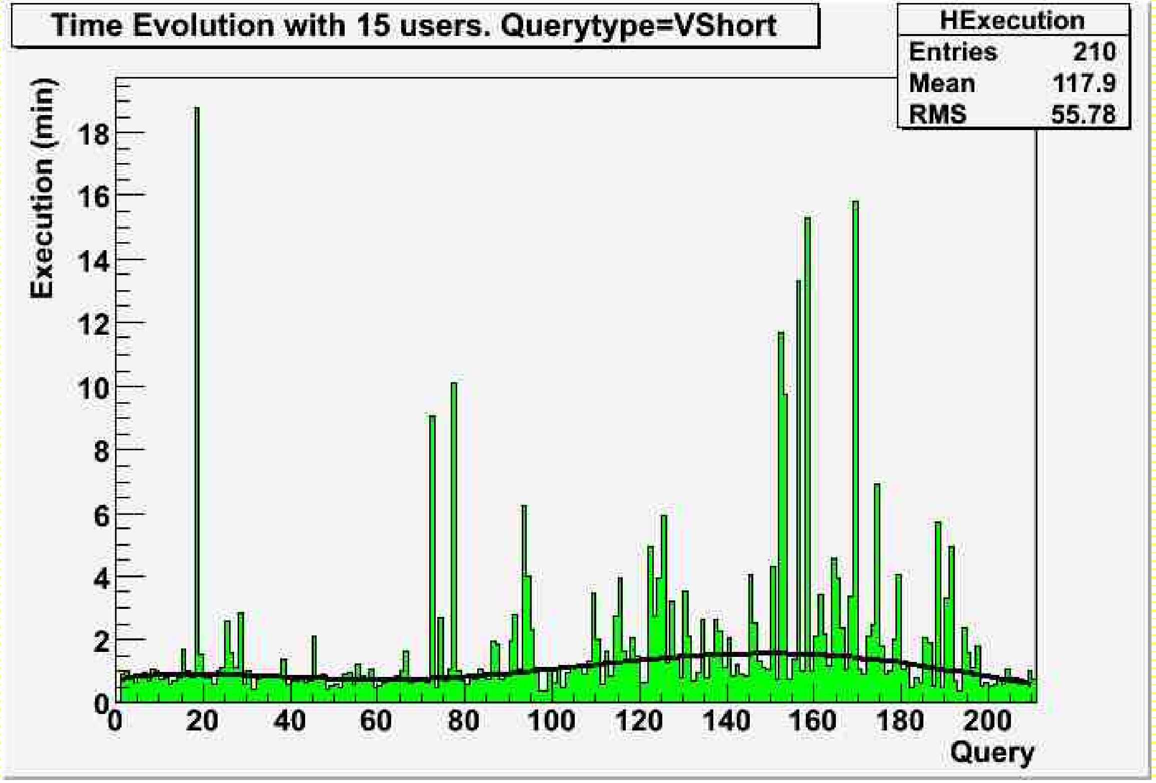
Query Type	#Queries	#Events	#Files (random)
20% very short	210	2k	20 small files
40% short	42	40k	20
20% medium	8	300k	150
20% long	3	1M	500

Parameters

- number of users
- number of workers
- number of files
- file selection method
- number of events
- execution time
- pause time
- average execution time
- median execution time

Spikes

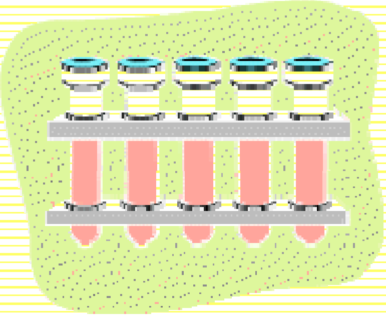
- “slow” packets (execution time > twice the median)



- found two less performing machines (Jan, Gerardo)
- limit on the #workers reading from same server (avoid bottlenecks)

III

Dataset Test

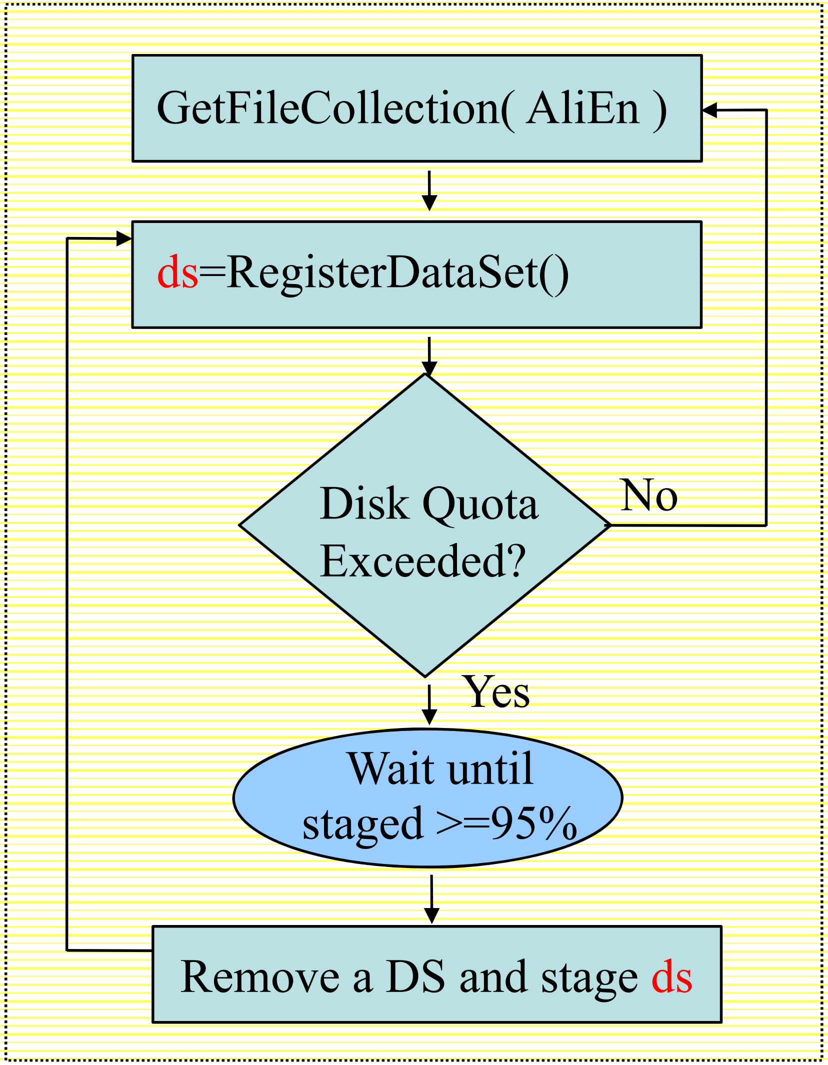


Aim

- Test the staging capabilities
- Staging demon developed by Jan Fiete
- Dataset API provided (see presentation by Gerhard)

Test Flow

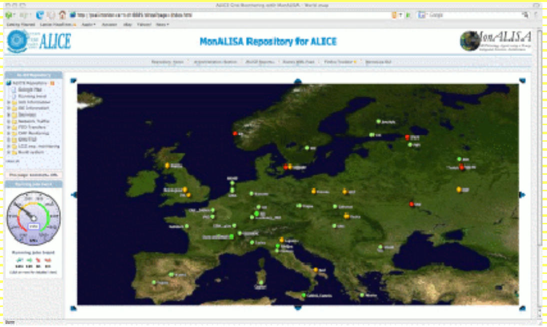
- 1000 files from AliEn catalogue
- ~60GB of data
- 9 input datasets (TFileCollection)
- Tested disk quota: 30 GB
- Successfully used to validate disk quota management



IV
CPU quota

Data Flow

Get groups' usage. Interval defined per each one: [α*quota..β*quota]



↓

measure difference between real usages and quotas

↓

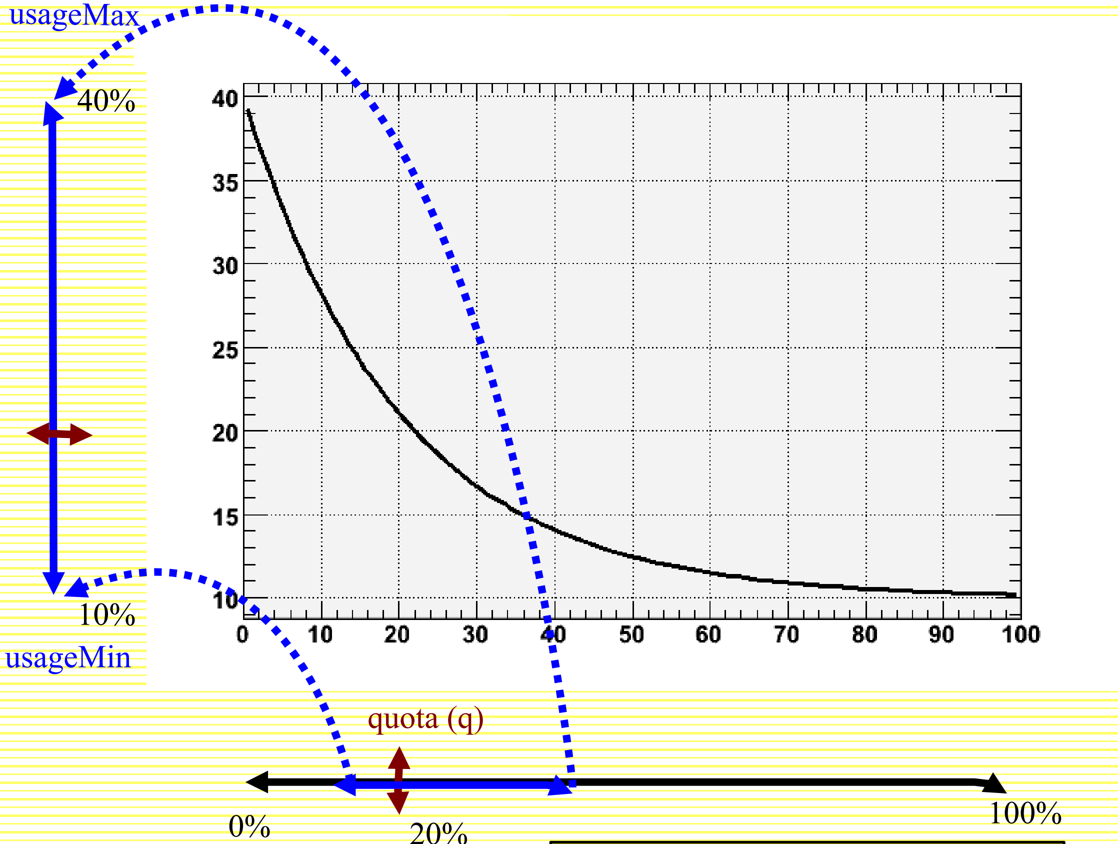
Compute new usages applying a correction formula

↓

CAF

Store computed usages

- Average every 6 hours
- Retrieved every 5 mins

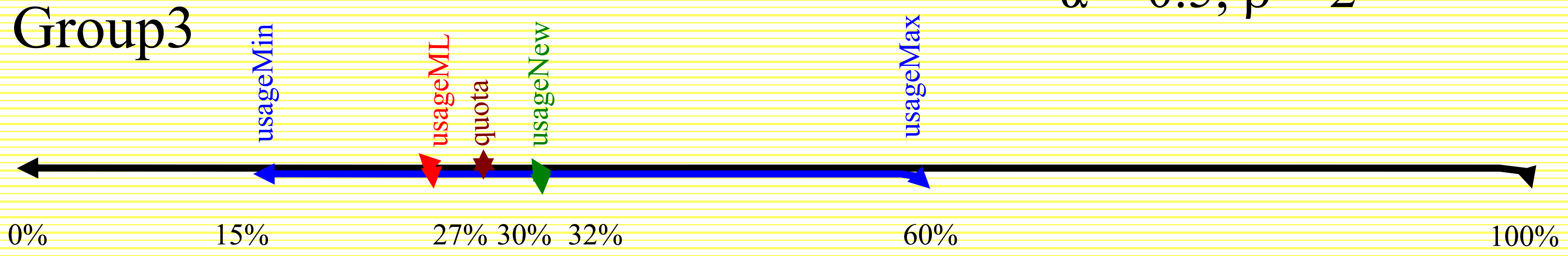


$$f(x) = \alpha q + \beta q^{*}\exp(kx)$$
$$k = 1/q^{*}\mathrm{Ln}(1/4)$$

Example

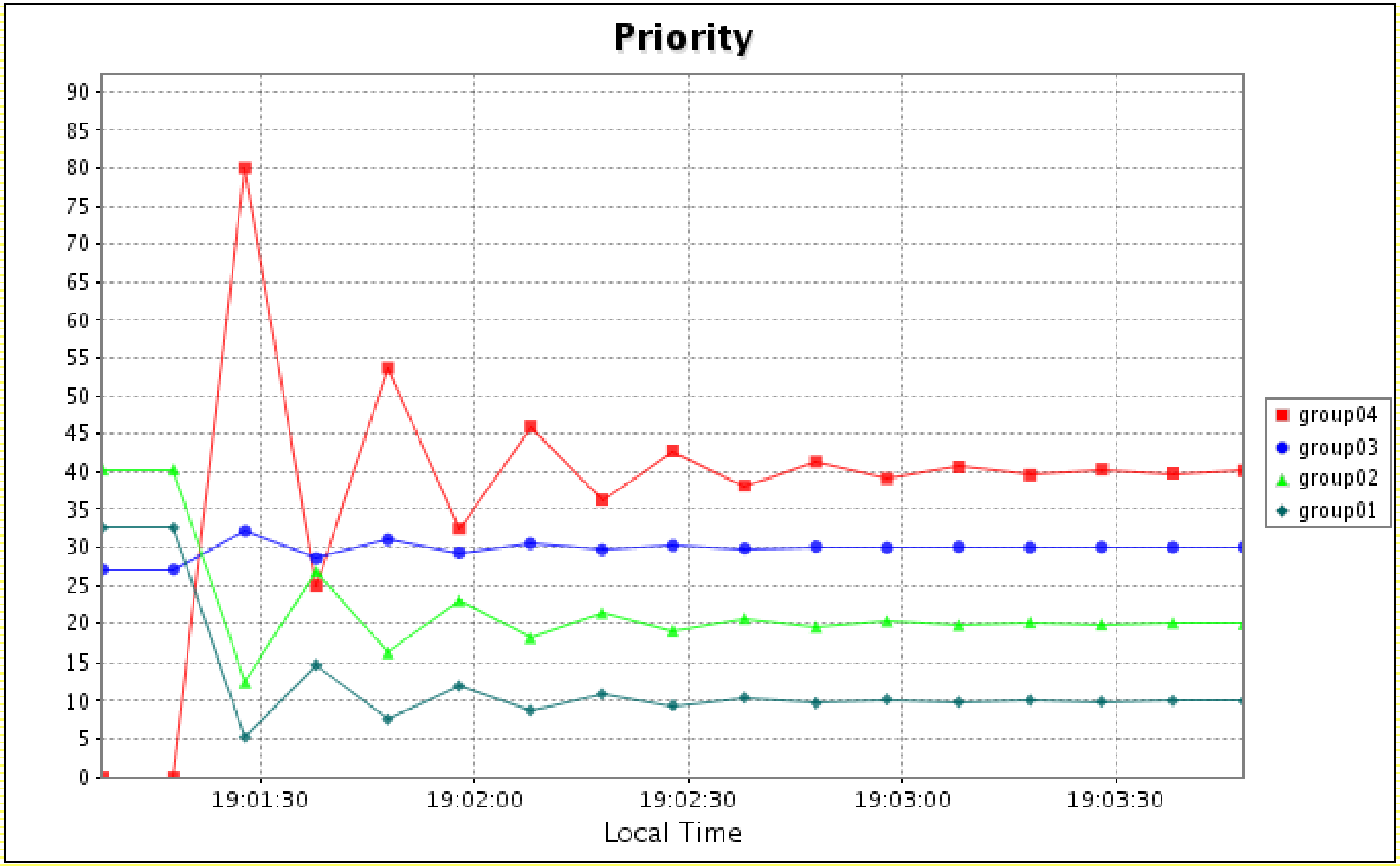
GROUP	Quota	Usage Interval	Last Usage from ML	“Corrected” Priority
group1	10%	5%..20%	32.59%	5.21%
group2	20%	10%..40%	40.30%	12.44%
group3	30%	15%..60%	27.09%	32.15%
group4	40%	20%..80%	0%	80%

- [α*quota..β*quota]
- α = 0.5, β = 2



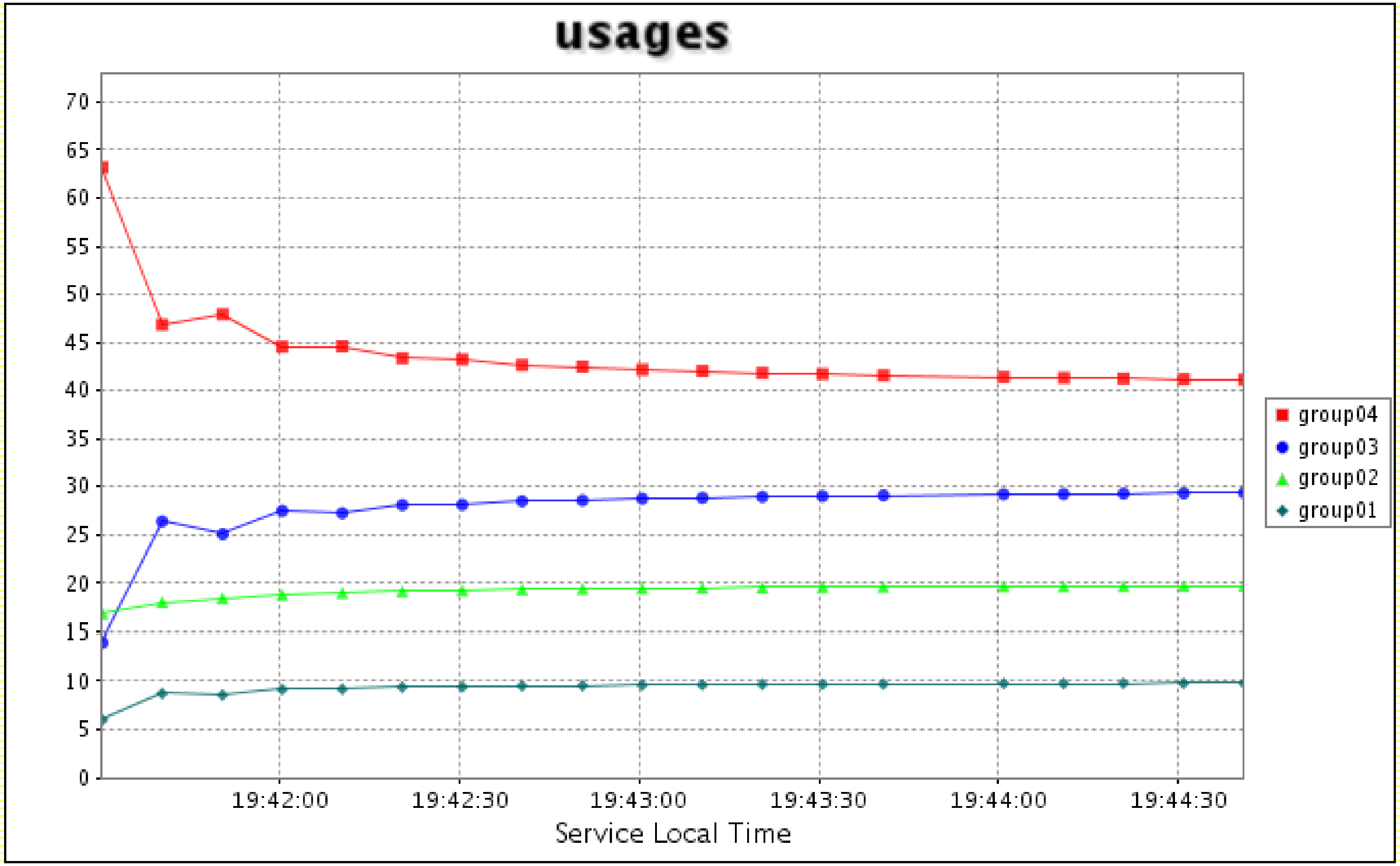
Priority Simulation

- Priorities from correction function converge to quotas



Usage Simulation

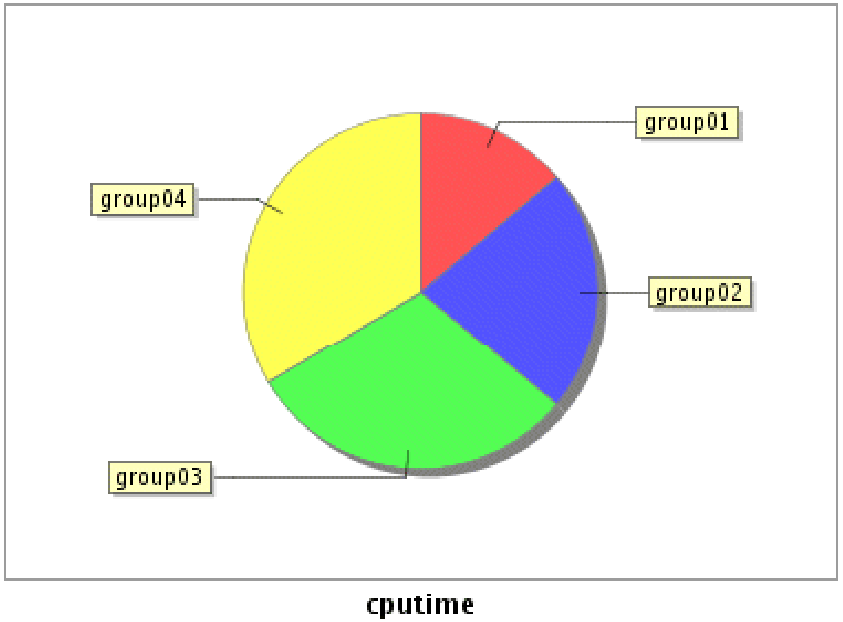
- Usages are gracefully steered to quotas without oscillating



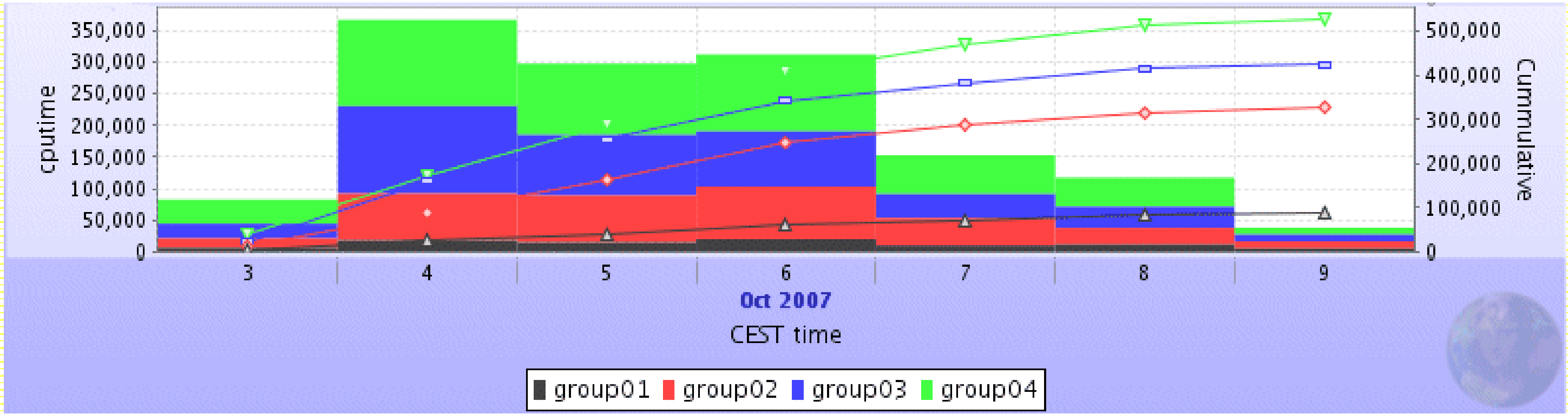
First day fully running (Oct 2nd)

- No query gets stuck
- Usages from MonALISA are averaged by 6 hours
- Priorities are not far from the quotas
- Some groups can last more than the others

Group	Usage	Quota
group04	34%	35%
group03	30%	30%
group02	22%	20%
group01	14%	10%



One Week Run (Oct 3rd-9th)



Group	Cpu Time	Usage	Quota
group04	526.623	38%	35%
group03	425.554	31%	30%
group02	327.561	24%	20%
group01	89.485	7%	10%
default	0	0%	5%

Conclusions

- Speed up tests over the last months have confirmed a linear behaviour
- Test for scalability on bigger cluster (currently 40 servers, bigger cluster will be setup soon)
- Cocktail tests optimized after initial behaviour showing unexpected peaks of execution time
- Cocktail tests are running continuously on a DEV cluster
- Observed a general stability of CAF (crashes are rare)
- Tested almost 900 queries in a row
- PROOF development team working hard, feedbacks from final users very important
- Successfully tested the disk quota deamon
- CPU quotas successfully tested on DEV cluster
- Priority mechanism ready to be put into PRO cluster