Outline

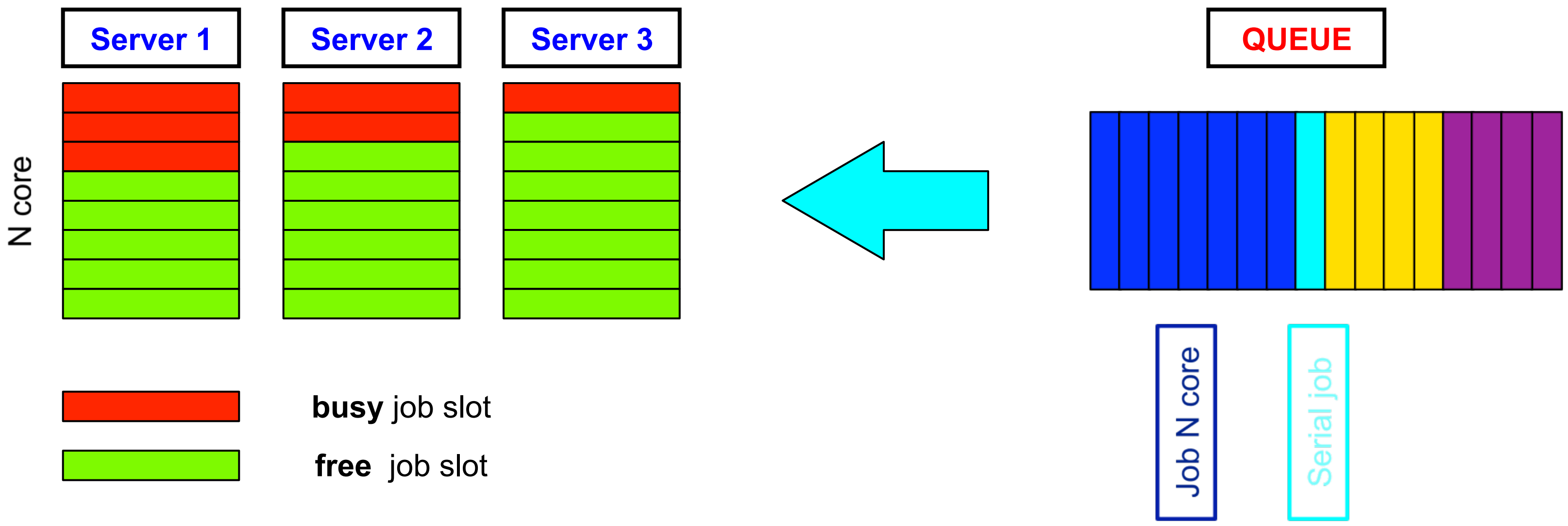
- **Introduction**
  - Let's start from the problem
  - Aims
  - Batch queue system features
- **Job migration**
  - Idea
  - Implementation
  - Requirements and use cases
- **Test and results**
  - Performance improvement
- **Conclusions**
  - Benefits

Introduction

What's the problem?

- given a computing farm composed by multicore [N] servers
- given a batch queue system: LSF, PBS, SGE
- given mixed serial [mono-core] and parallel [multi-core] jobs

…a **stuck** situation may occur!



Aims and issues

Aims

- Improve the farm exploitation in terms of running jobs
- Reduce the free job slots

Batch queue system features

- The batch queue system can not modify the queued jobs order
- The scheduler has to respect fairshare and job priorities
- **The batch queue system can not move jobs at runtime**

Solutions

Possible solutions:

- **Cluster partition** [serial, parallel]
  - **CONS**: no shared resources benefits, cluster under-exploitation in case of only serial or parallel jobs
- **Job Rearrangement**
  - **PRO**: farm full exploitation

The project

The idea

- Set up the batch system behavior in order to fill the minimum number of server, instead of balance the load between all the available servers
- Rearrange jobs allocation at runtime
  - at scheduled time interval
  - considering the free resources available in the farm

The simulator

- FARM simulator
- QUEUE simulator
- JOB MOVER algorithm
- Statistics collector

Requirements and use cases

Requirements

- Batch queue system needs to provide the job migration feature
- Jobs have to be checkpointable, independent, restartable
- Jobs requirements in terms of CPU, RAM, disk and I/O need to be compliant to the **given acceptance schema**:

  N core, N/C % {RAM, disk, I/O} N being the number of required cores, and C the single server cores number

Use cases

- Mixed serial [mono-core] and parallel [multi-core] jobs; where parallel jobs are spread between 2 and C core, C being the server cores number
- Jobs running time: 1 hour to 15 days
- Data acquisition: 1 year
- Queued jobs distribution: random or sequential

Problem complexity

How many job slots permutations?

Given J running job, each one requiring 1 to L number of cores, running over a farm composed by S servers with N cores, how many permutations in the jobs disposition are possible?

Surely TOO MUCH to be analyzed! It is probably a NP-complete problem.

How to do?

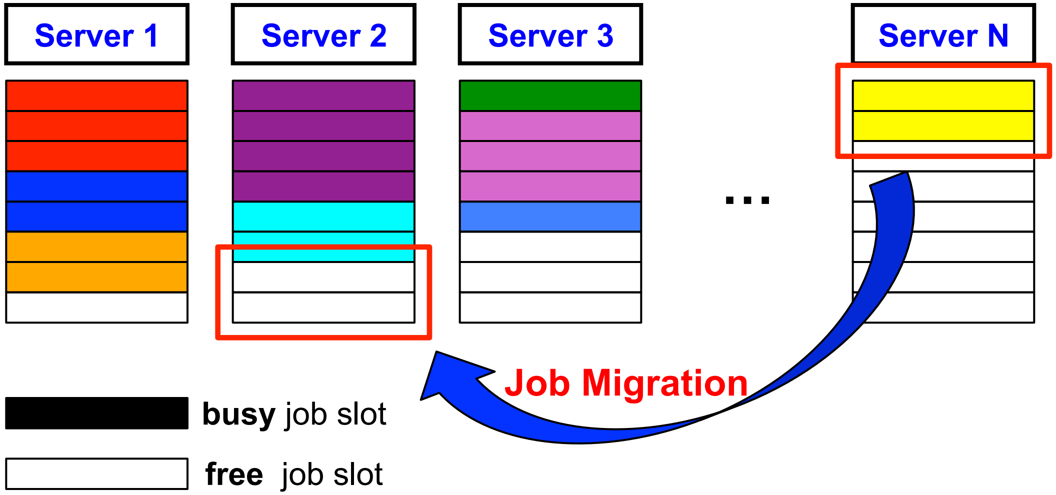
- We are not searching for the optimum solution, but simply for a solution better than the current one.
- The farm simulator, combined with the job mover method, may be used to test other algorithms, in order to find a new one more efficient than ours.

Algorithm

The chosen algorithm

How to rearrange the jobs:

- reverse sort the servers by busy job slots
- try to fill the most full server with jobs coming from the most free server



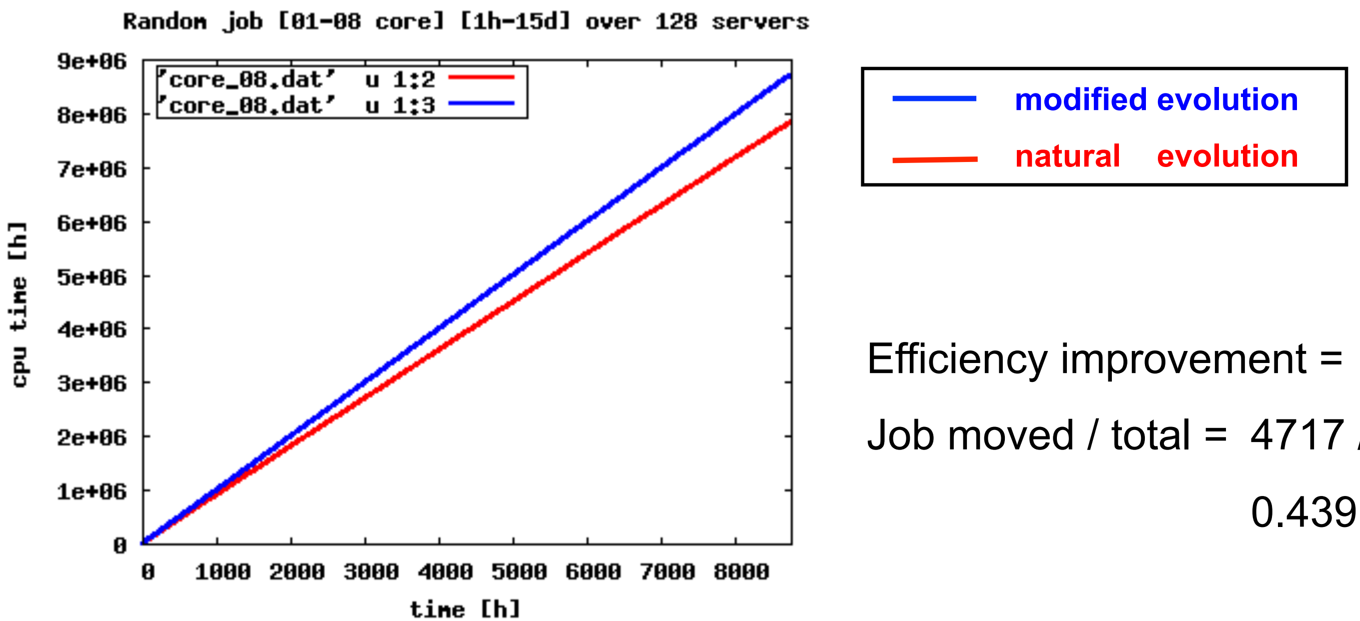
Use case 1

Random

Jobs distribution:

- Cores number 1 to 8
- Running time 1 hour to 15 days
- Queue filling random

in a 128 servers, 8 core farm – 1 year of data acquisition



Efficiency improvement = 12 %

Job moved / total = 4717 / 10726

0.439

Use case 2

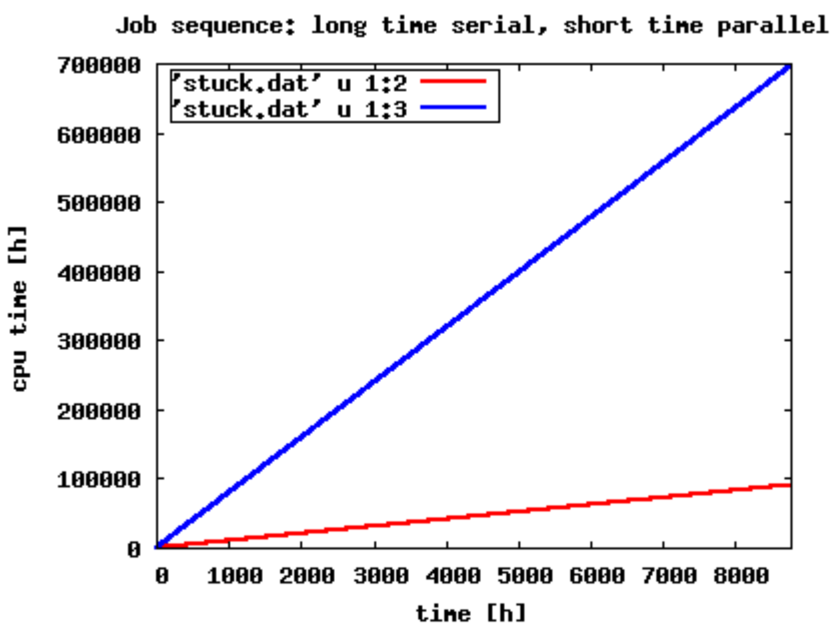
Worst [or best] situation [depending on the point of view]

Jobs distribution:

- Repeated sequence of:

  [serial mono-core long term jobs, followed by parallel full-core short term jobs]

  in a 10 servers, 8 core farm – 1 year of data acquisition



modified evolution

natural evolution

Efficiency improvement = 800 %

Job moved / total = 2239 / 17156

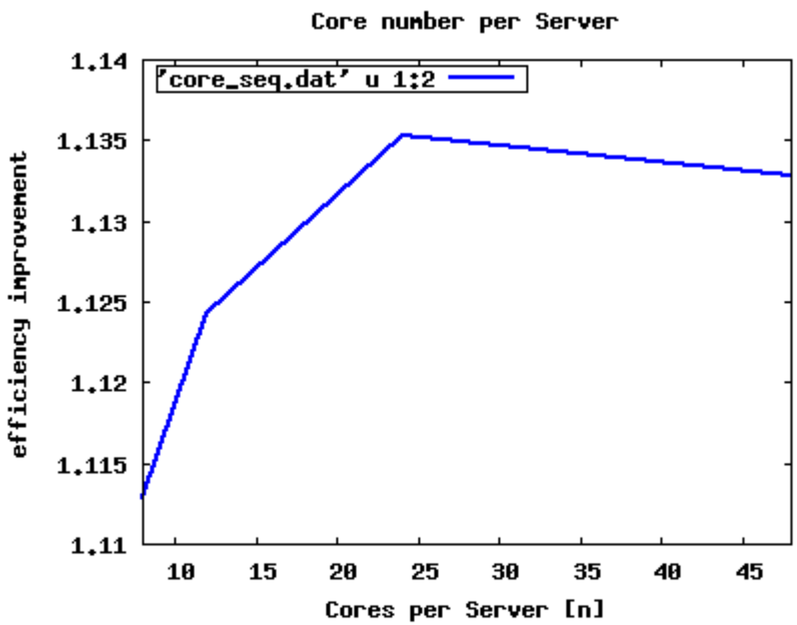
0.130

Algorithm efficiency

Server with 8, 12, 24, 48 cores

The algorithm efficiency with respect to the number of Cores per server in a 128 servers farm – 1 year of data acquisition

- Random job sequence [1-N core], [1h-15d]



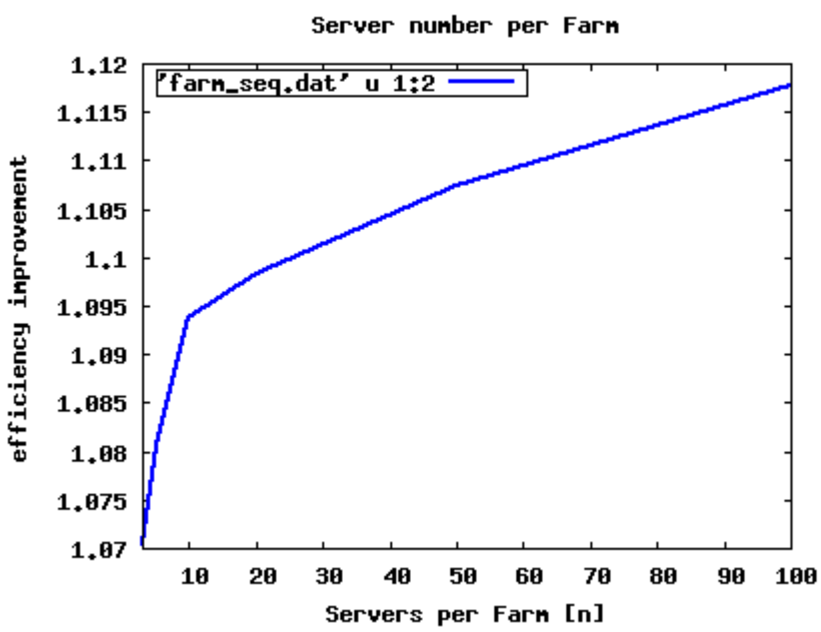
Efficiency improvement = [11-13]%

Algorithm efficiency

Farm with 3, 5, 10, 20, 50, 100 servers

The algorithm efficiency with respect to the number of Servers per farm with 8 core servers – 1 year of data acquisition

- Random job sequence [1-8 core], [1h-15d]



The farm efficiency increases with the increasing of the server number

Efficiency improvement = [7-12]%

Job moved / total = [10-50]%

depending on farm size and jobs type

A touch of Green Computing

- In case of empty queue, it is possible to use the Job Migration strategy in order to free resources and switch off the unused hosts and improve the electrical power efficiency of the farm.

- Using a remote controlled power supply, it is possible to switch off the unused hosts, waiting to be switched-on at request.

Conclusions

- A job displacement, executed at runtime in order to stack up the maximum processes number over single multi core servers, is able to free extra resources - and consequently host new processes in the computing farm.

- The runtime job rearrangement in a computing farm may provide an improvement in terms of efficiency of about 7-13 % depending of the use case.

Any other idea with respect to new algorithms is welcome.

Acknowledgments and Questions

Thanks for your attention

Please feel free to send questions, [criticisms], **suggestions to the authors**

Contact email: Federico Calzolari <federico.calzolari@sns.it>