Convergent Scheduling for Large Computing Farms

R. Baraglia, G. Capannini, *M. Pasquali*,
D. Puppin, L. Ricci,

ISTITUTO DI SCIENZA E TECNOLOGIE DELL'INFORMAZIONE "A. FAEDO"

Institutions Markets Technologies
IMT
LUCCA INSTITUTE FOR ADVANCED STUDIES

An agile approach to job scheduling

- *A modular framework that can manage multiple constraints*
- *Several heuristics able to solve many typical cases*
- *An evaluation strategy based on simulation and aggregated metrics*

Main features

- *The system can schedule job streams on a large heterogeneous cluster*
- *Respecting deadlines, running requirements, license constraints*
- *Very fast execution of scheduling*

Convergent Scheduling

- *Composed of many passes, each of which implements a specific scheduling heuristic.*
- *Passes share a common interface, which allows passes to be run multiple times, and in any order.*
- *The final scheduling plan will result out of this combined approach.*

Scheduling problem

Machines set *M*

- *benchmark score*
- *number and type of the CPUs*
- *licences*

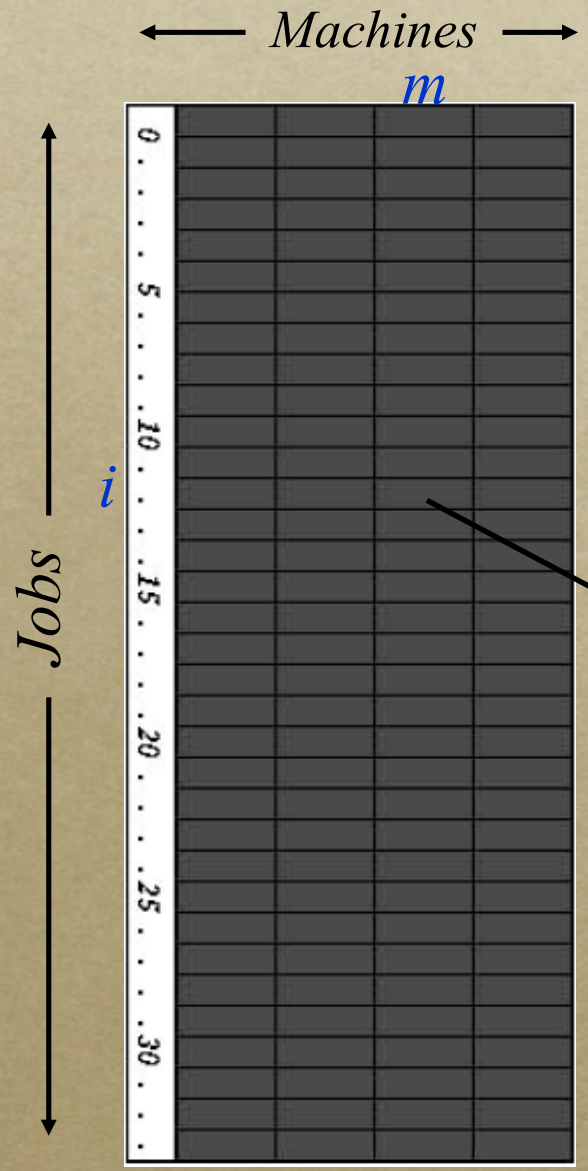
Licences set *L*

- *floating*
- *NON floating*

Jobs set *N*

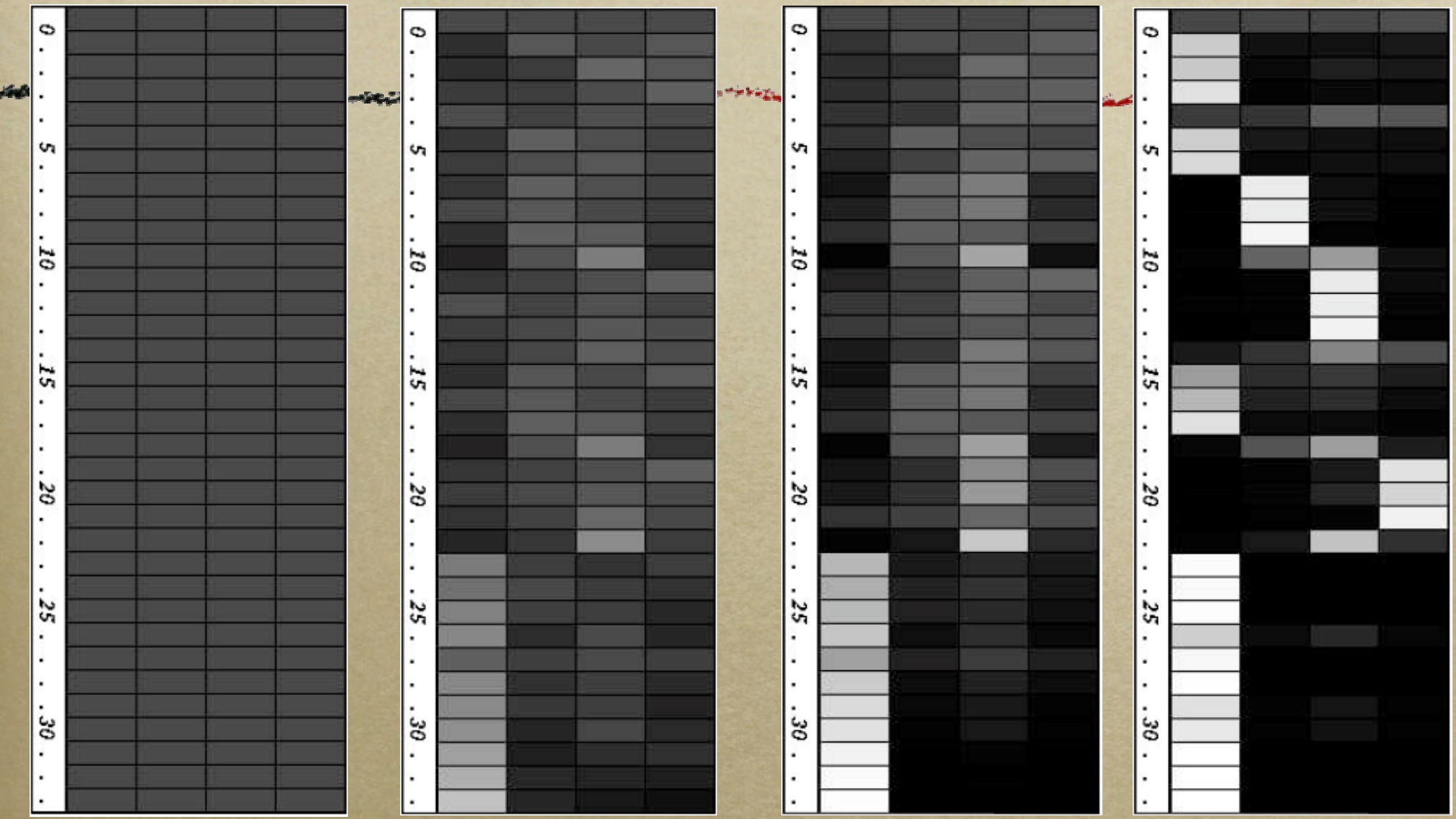
- *identifier*
- *deadline*
- *execution time estimation*
- *benchmark score*
- *input data*
- *number and type of the CPUs*
- *licences*

Convergent scheduling

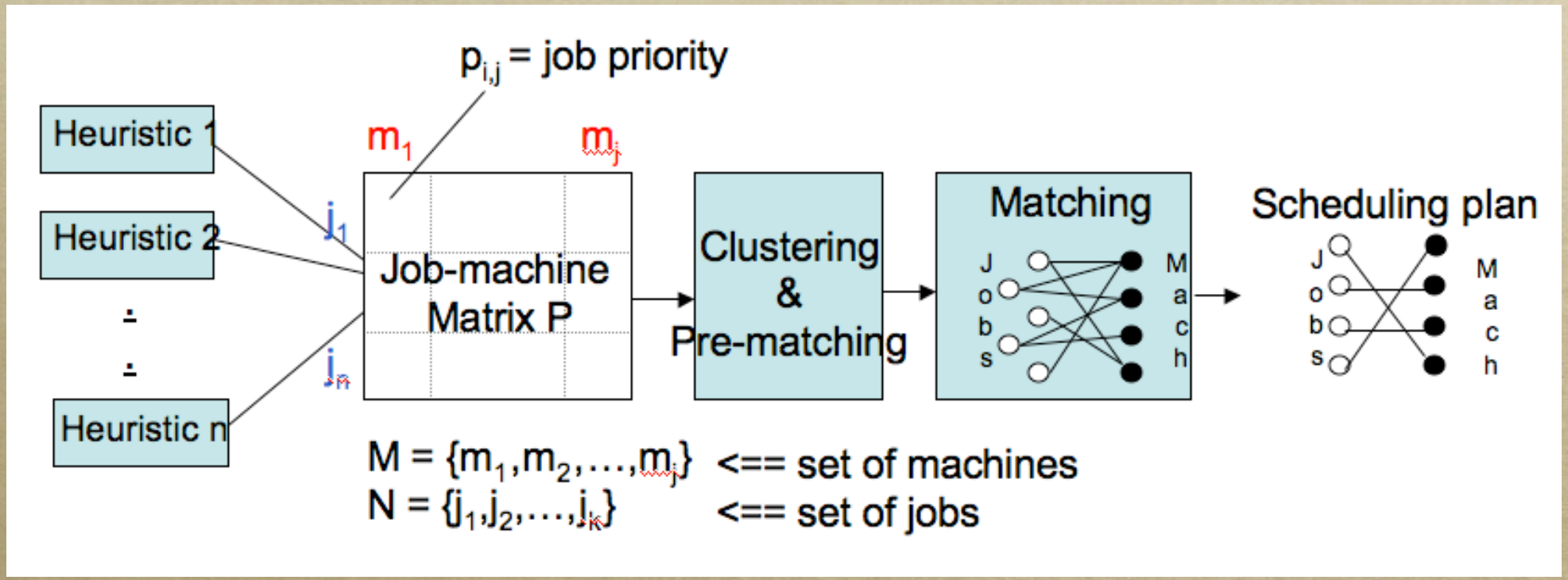


$P^{|N|x|M|}$ is the job machine matrix

Degree of preference of a job *i* for a machine *m*



Structure of the CS framework



Heuristics

Heuristic	Goal
Minimal requirements	To select the set of m_j with the features required to execute j_i
Deadline	To execute the job by respecting their deadline
Licenses	To respect the constraint on the sw licenses by optimizing their usage
Input	To reduce the cost due to I/O files transfer
Wait minimization	To minimize the job response time
Overhead minimization	To reduce the overhead due to job execution stop-resume.
Anti-aging	To avoid large response time for jobs without a deadline

Minimal requirements

- *This heuristics fixes the associations job-machines.*
- *It selects the set of machines* M_{Affi} *that has the computational requirements suitable to perform a job i.*
- *In our tests, we considered only two requirements: number of processors and floating licenses (i.e. number and type).*

Deadline

- *Updates the job priorities to execute jobs respecting their deadline.*
- *The job-machine matrix entries of "late" jobs are increased proportionally to the proximity (delay time) of their deadline.*
- *Jobs closer to their deadline get a boost (f_1) that gives them an advantage to be scheduled onto the more powerful machines.*

Deadline [1]

- *Boost function*

$$f_1 = \begin{cases} Min & \text{if } Extime_{i,m} \leq t_{i,m} \\ a(Extime_{i,m} - t_{i,m}) + Min & \text{if } t_{i,m} < Extime_{i,m} \leq deadline_{i,m} \\ Min & \text{if } Extime_{i,m} > deadline_{i,m} \end{cases}$$

- *Time required to complete a job execution*

$$t_{i,m} = Deadline_{i,m} - k \cdot Nxtime_{i,m}$$

$$Nxtime_{i,m} = (estimated_i \times (1 - progress_i)) \cdot \frac{BM_{\bar{m}}}{BM_m}$$

- *Evaluation of the time at which a job will end its execution*

$$Extime_{i,m} = Now + Nxtime_{i,m}$$

Deadline [2]

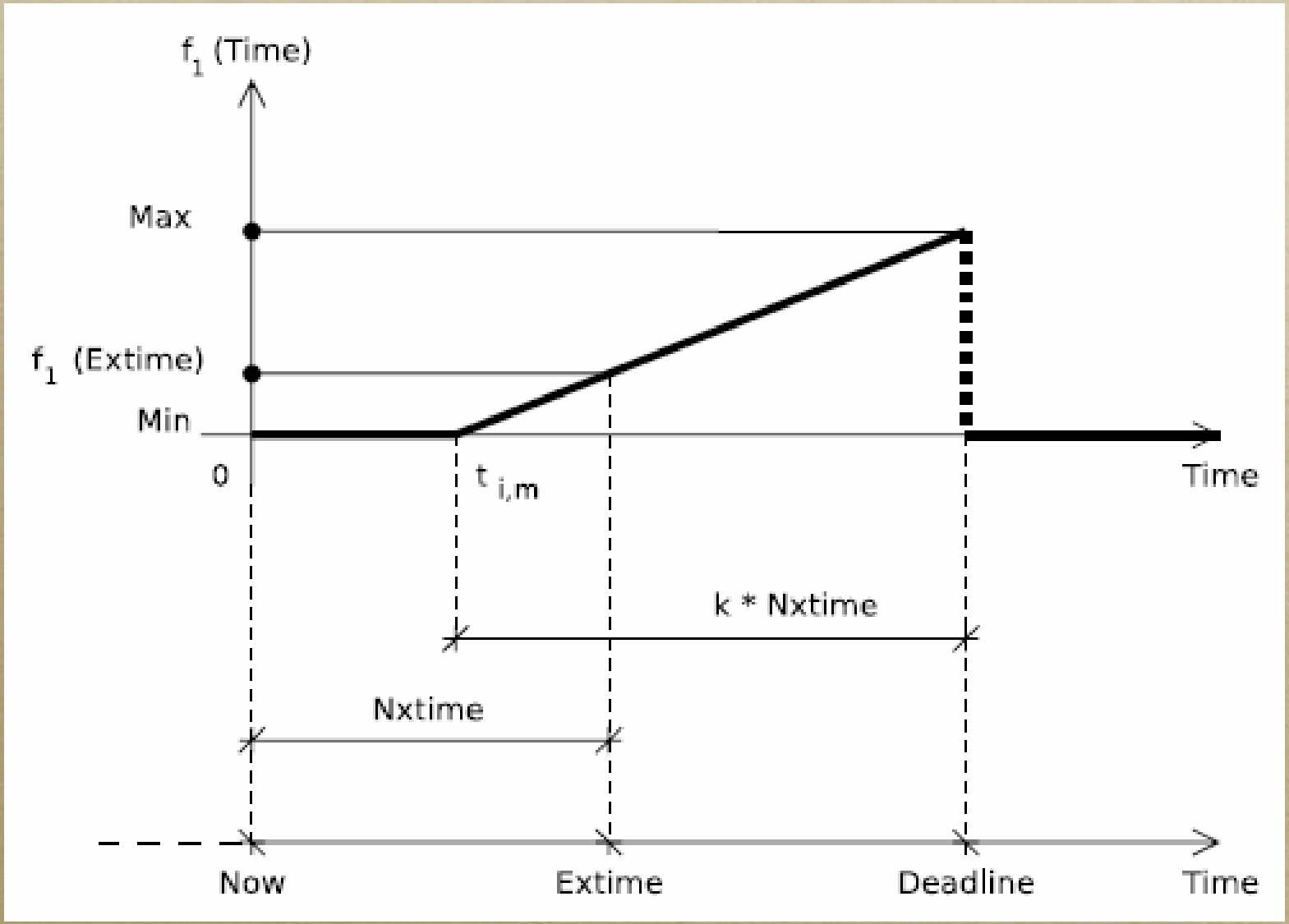
The values to update the job-machines entries are computed according to the following expressions:

$$\Delta_{p_{i,m}} = \left(\sum_{k=1}^{|M_{Aff_i}|} f_1(Extime_{i,k}) \right) \cdot \frac{\bar{f}_1(Extime_{i,m})}{d_i}$$

$$\bar{f}_1(Extime_{i,m}) = Max - f_1(Extime_{i,m})$$

$$d_i = \sum_{m=1}^{|M_{Aff_i}|} \bar{f}_1(Extime_{i,m})$$

Deadline [4]


f1 (Time)
Max
f1 (Extime)
Min
0
t i,m
Time
k * Nxtime
Nxtime
Now
Extime
Deadline
Time

Licenses

- *Critical licenses are those with a request number greater than availability*
- *We give higher score to jobs requesting a high number of critical licenses*

Licenses [1]

- *The heuristic updates the machine-job preference degree according to the requested critical licenses*
- *Definition of critical license*

$s_{j,l} \in \{0,1\} \wedge s_{j,l} = 1$ *iff* $<j \in J$ *ask for* $l \in L>$

$\forall l \in L,\ \rho_l = (\sum_{\forall j \in J} s_{j,l}) \times (a_l)^{-1}$

$\sum_{\forall j \in J} s_{j,l}$ = *number of jobs requesting a sw license*

a_l = *number of sw license copies that can be simultaneously active*

l is critical iff $\rho_l > 1$

Licenses [2]

- *In order to compute the score we introduce the function w() to assign a weight to a job according to its sw licenses usage.*

$$w(j) = \sum_{\forall l \in L} s_{j,l} \cdot \rho_l$$

- *An example*
  - *We assume to dispose of 3 different types of licenses: l_0, l_1, l_2.*
  - *Let j_0, j_1, j_2 three jobs, which need the sw licenses (l_2) and (l_1, l_2), (l_0, l_1, l_2), respectively*
  - *l_0, l_1, l_2 are usable on all the three available machines.*
  - *l_0 and l_1 activable in a copy at time, l_2 in four copies.*

$$\rho(l_0) = \sum_{i=0}^{2} s_{i,0}/a_0 = \frac{1}{1} = 1.00$$

$$\rho(l_1) = \sum_{i=0}^{2} s_{i,1}/a_1 = \frac{2}{1} = 2.00$$

$$\rho(l_2) = \sum_{i=0}^{2} s_{i,2}/a_2 = \frac{3}{4} = 0.75$$

Licenses [3]

- *Concerning the example, w() is computed as:*

$$w(j_0) = 0 \cdot \rho_0 + 0 \cdot \rho_1 + 1 \cdot \rho_2 = 0.75$$
$$w(j_1) = 0 \cdot \rho_0 + 1 \cdot \rho_1 + 1 \cdot \rho_2 = 2.75$$
$$w(j_2) = 1 \cdot \rho_0 + 1 \cdot \rho_1 + 1 \cdot \rho_2 = 3.75$$

- *According to the computed score (i.e. w()), the next phase will select j_2 and will discard j_1 obtained* $\forall l \in L\ \rho_l \leq 1$
- *It permits us to run first the job asking for a larger number of critical licenses, reducing the contention on such licenses due to new job submissions.*
- *It makes it easier to meet the deadline of such new jobs.*

Input

The goal of this heuristics is to update the matrix entries according to the cost due to the transfer of the job input data on the machines candidate to run it.

The input data transfer time for a job i is computed as:

$$Transf_i(m) = \sum_{\forall m \in M_{Aff_i} \wedge \bar{m} \neq m} \frac{input_{i,\bar{m}}}{b_{\bar{m},m}} \quad \text{with } i \in N$$

The updated value is computed as an inverse proportion of the data transfer time:

$$\Delta_{p_{i,m}} = max \cdot \left(1 - \frac{Transf_i(m)}{\sum_{\forall m \in M_{Aff_i} \wedge \bar{m} \neq m} Transf_i(m)}\right)$$

Wait Minimization

- *The aim of this heuristics is to minimize the average time that jobs spend waiting to complete their execution.*

$$\Delta_{p_{i,m}} = max \cdot \left(1 - \frac{remaining_{i,m}}{sup_{remaining}}\right)$$

$$remaining_{i,m} = estimated_{i,\bar{m}} \cdot (1 - progress_i) \cdot \frac{BM_{\bar{m}}}{BM_m}$$

Overhead Minimization

- *The heuristic aims to contain the overhead due to the interruption of a job execution on a machine and its resuming on another one, or on the same one at a different time.*

- *This heuristic checks the current job-machine matching and tries to minimize the job migrations by preserving the scheduling made in the previous step.*

Anti-aging

•The goal of this heuristic is to avoid that a job remains, for a long time, waiting to start or progress its execution.

$$\Delta_{p_i,m} = (wallclock - submit_i) \cdot age\_factor \quad \forall m \in M_{Aff_i}$$

clustering & pre-matching

- *This phase is executed after the job-machine matrix was updated by all the heuristics.*
- *All the jobs requiring critical licenses ($\rho_l > 1$) are clustered by putting in the same cluster the jobs asking for the same license/s, and with each job belonging to only a cluster.*
- *When a job asks for more than one critical license, the related clusters are merged, and whenever a next job asks for such licenses, it is put in such merged cluster.*

clustering & pre-matching [1]

- *The MKP (Multidimensional 0-1 Knapsack Problem) optimization method is applied to each cluster to find the subset of jobs that could be simultaneously performed, without violating the constraint on the licenses usage.*
- *The remaining subset will be discarded, i.e. their entries are cleared by the job-machine matrix.*
- *The resulting job-machine matrix will be passed to the matching phase.*
- *Adopted solution based on the Eilon's algorithm.*

Matching

- *Starting from the matrix, it computes the new scheduling plan.*
- *Goal:*
  - Find a matching $MTC \subset N \times M$ that maximizes
    - the cardinality of the job-machine associations $|MTC|$
    - the sum of the selected associations $\Sigma p_{j,m} \in MTC$
  - According to the following condition
    - $(j_1,m_1) \in MTC \wedge (j_2,m_2) \in MTC \Leftrightarrow j_1 \neq j_2 \wedge m_1 \neq m_2$
  - $|M| = |MTC| \Rightarrow$ *system usage* = 1

Matching 2: Maximum flow search

We compute a more effective matching by means of standard Max Flow Search algorithm

- *MFS solution permits us to carry out a job-machine association set with max cardinality and max cost*
- *Algorithm complexity $O(n^4)$*
- *Golden standard for our comparison*

Matching 3: Incremental maximum flow search

- *We sped the process up by an incremental matching*
- *It can be computed in $O(n^3)$*
- *Exploitable when the matching process is not fast enough to provide fresh data for the scheduling.*

Experimental Simulation

- *Our discrete simulator tracks the usage of machines (single or multi proc.), the job execution, the number of licenses*
- *Main metrics: average usage, failed/respected deadlines, average delay (SlowDown), scalability,*
- *Fully configurable for jobs, machines, licenses*

Experimental Simulation [1]

The evaluation was conducted by simulations using different streams of jobs.

- *Estimated [500 ÷ 3000]*
- *Deadline [25 ÷ 150]*
- *Num. CPUs [1 ÷ 8]*
- *Benchmark [200 ÷ 600].*
- *Input [100 ÷ 800]*

- *License ratio [55% ÷ 65%] maximum number of copies concurrently usable*
- *License suitability 90%*
- *License needed by a job 20%*

Experimental Simulation [2]

We evaluated three different versions of CS:

CS-incr

CS-nopre

CS-noincr

Compared with:

EASY Backfilling

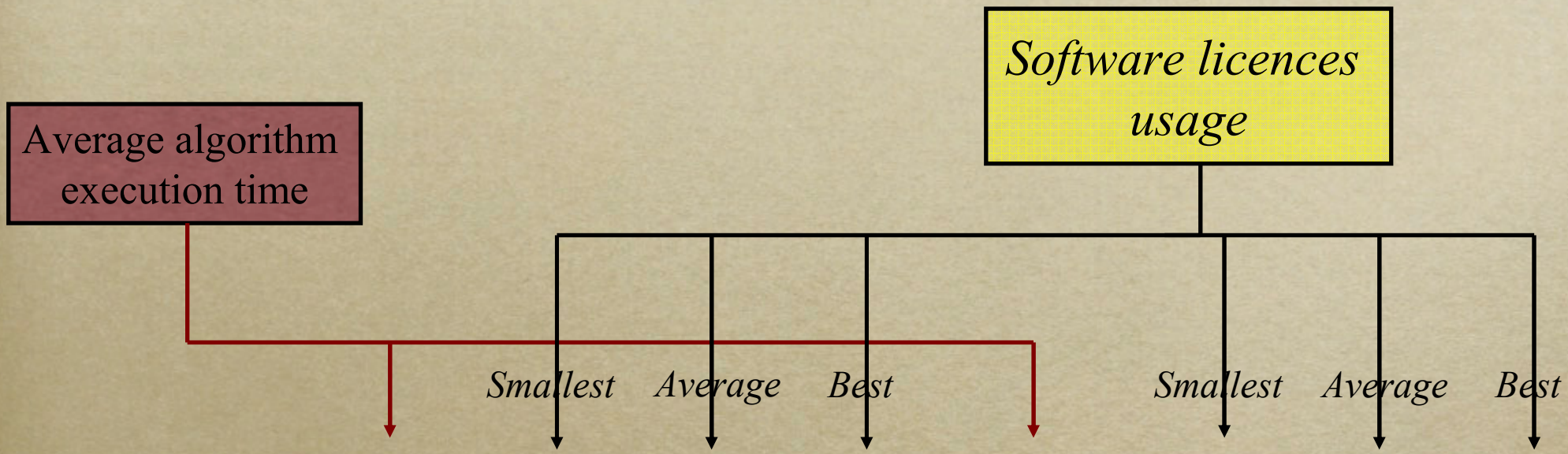
Flexible Backfilling

Early Deadline First

Bf-rm

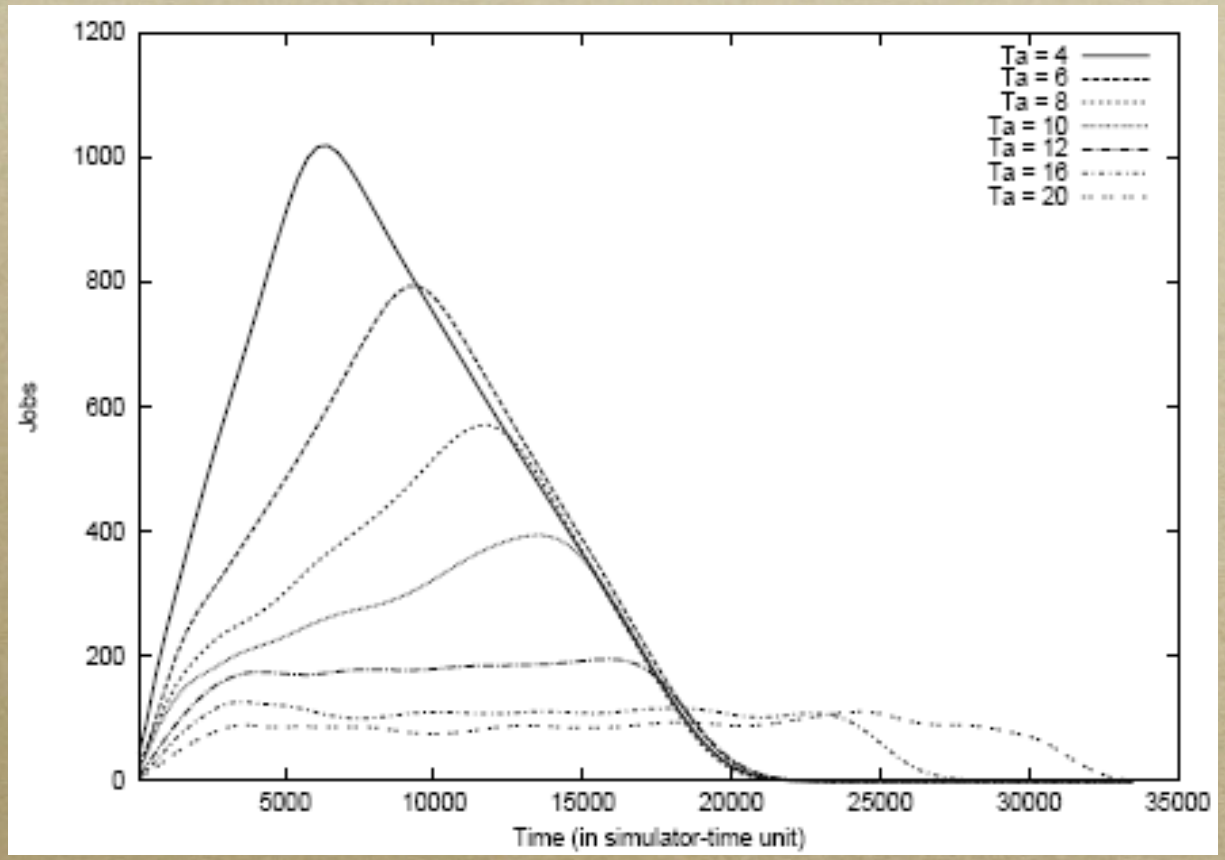
Bf-unrm

Experimental Simulation [3]



License ratio	Branch and bound				MKP			
	Time (ms)	Sml (%)	Avg (%)	Bst (%)	Time (ms)	Sml (%)	Avg (%)	Bst (%)
$[40 \div 45]$	496.53	0.57	0.87	1.00	11.23	0.54	0.82	1.00
$[45 \div 50]$	353.51	0.62	0.88	1.00	10.40	0.59	0.84	1.00
$[50 \div 55]$	331.28	0.64	0.89	1.00	10,42	0.62	0.87	1.00
$[55 \div 60]$	556.96	0.64	0.88	1.00	11.90	0.63	0.87	1.00

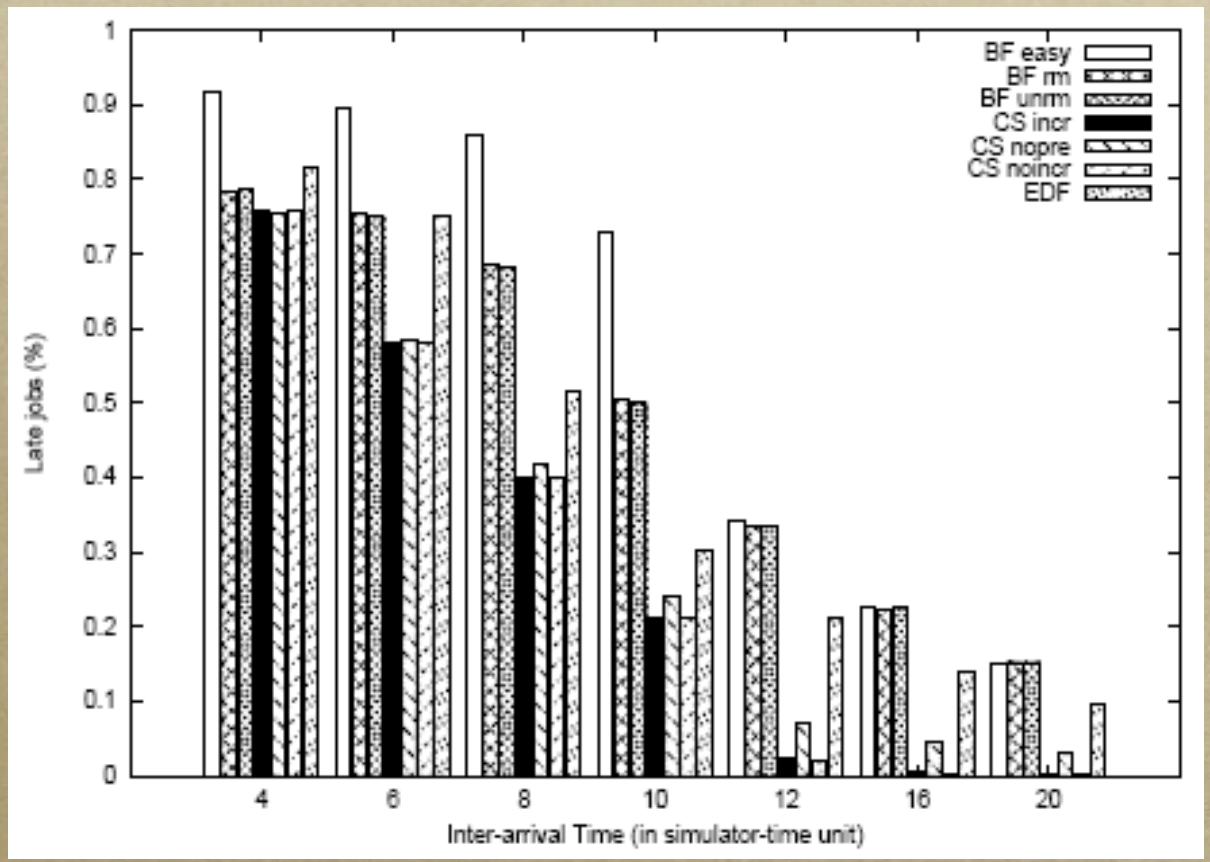
Experimental Simulation [4]



System workload through simulation

The closer jobs inter-arrival time is, the higher the contention in the system is.

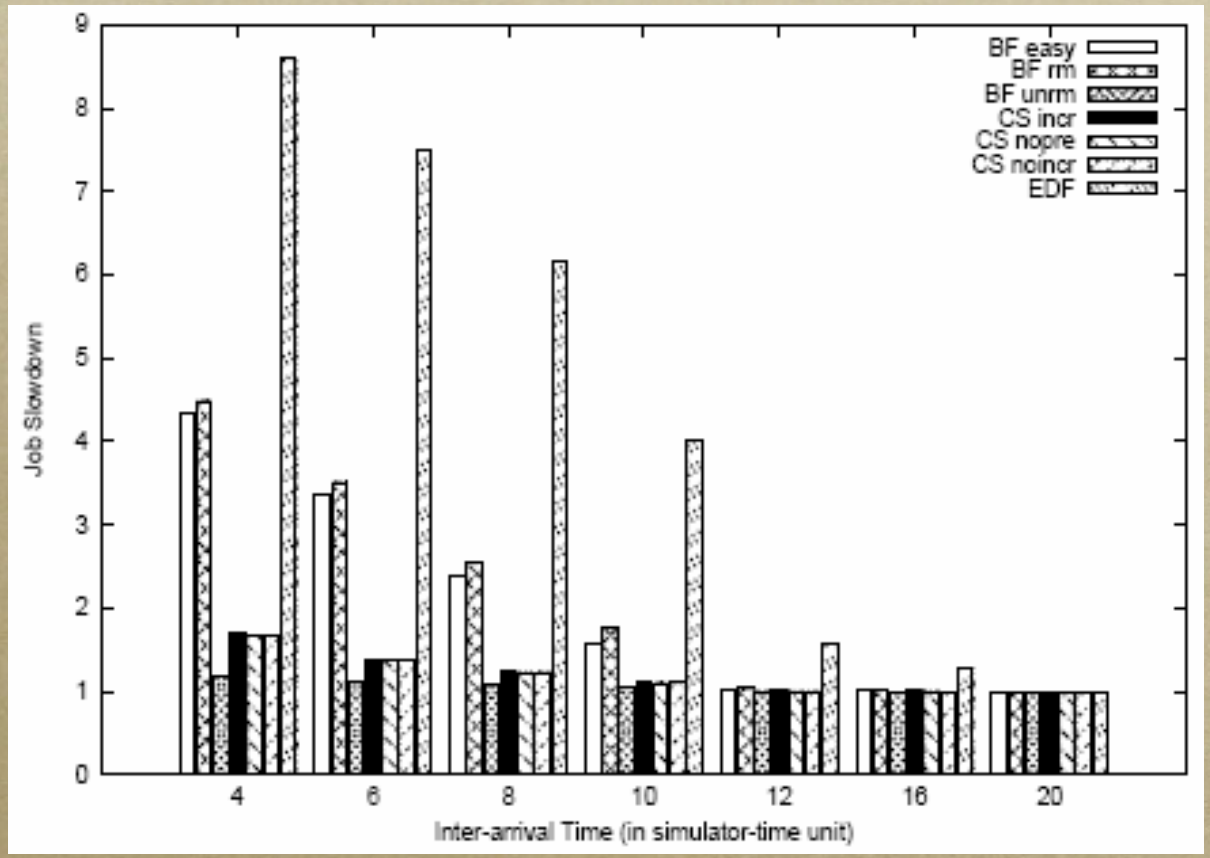
Experimental Simulation [5]



Percentage of jobs executed do not respecting the deadline constraint

The smaller the job inter-arrival time is,
the greater the job competition in the system is,
and consequently the number of late jobs improves.

Experimental Simulation [6]

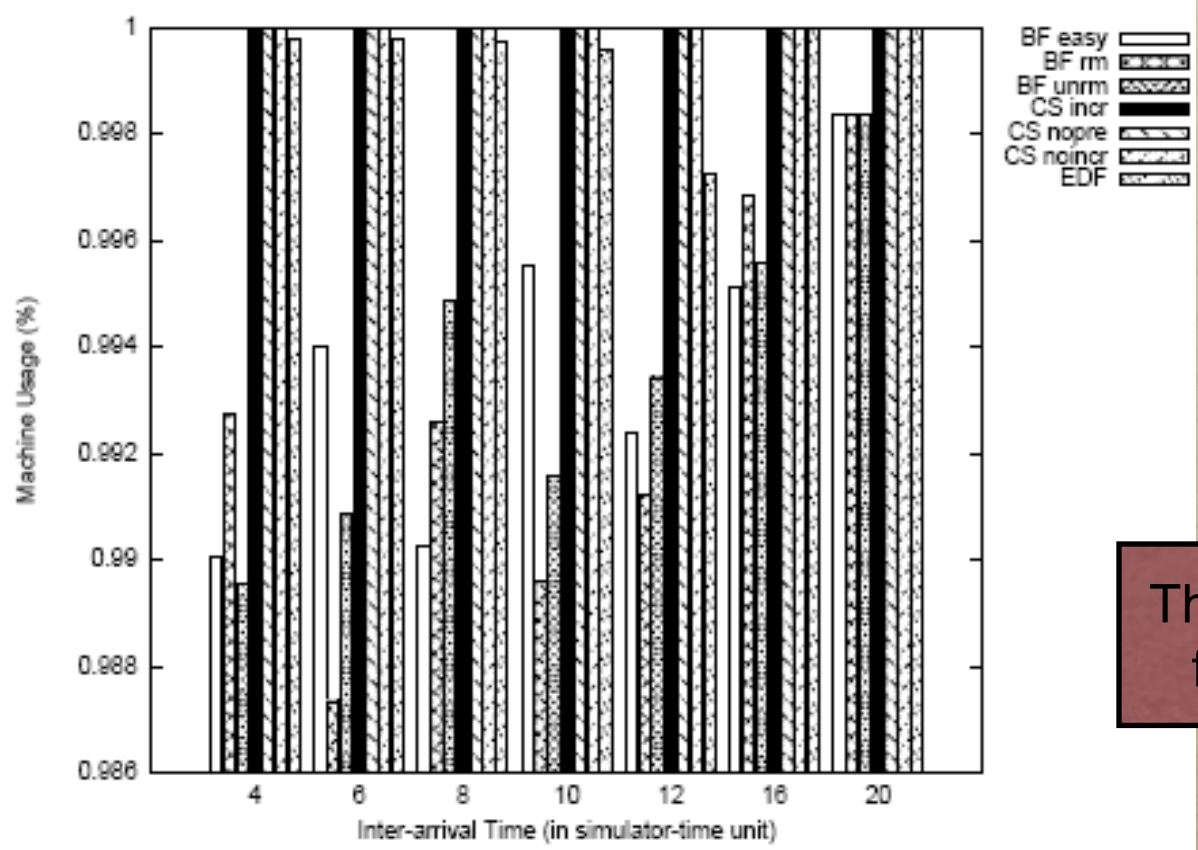


Slowdown of jobs submitted without deadline.

$$\textbf{Slowdown} = (T_w + T_e)/T_e$$

This parameter gives us as the system load delays the execution of such jobs.

Experimental Simulation [7]

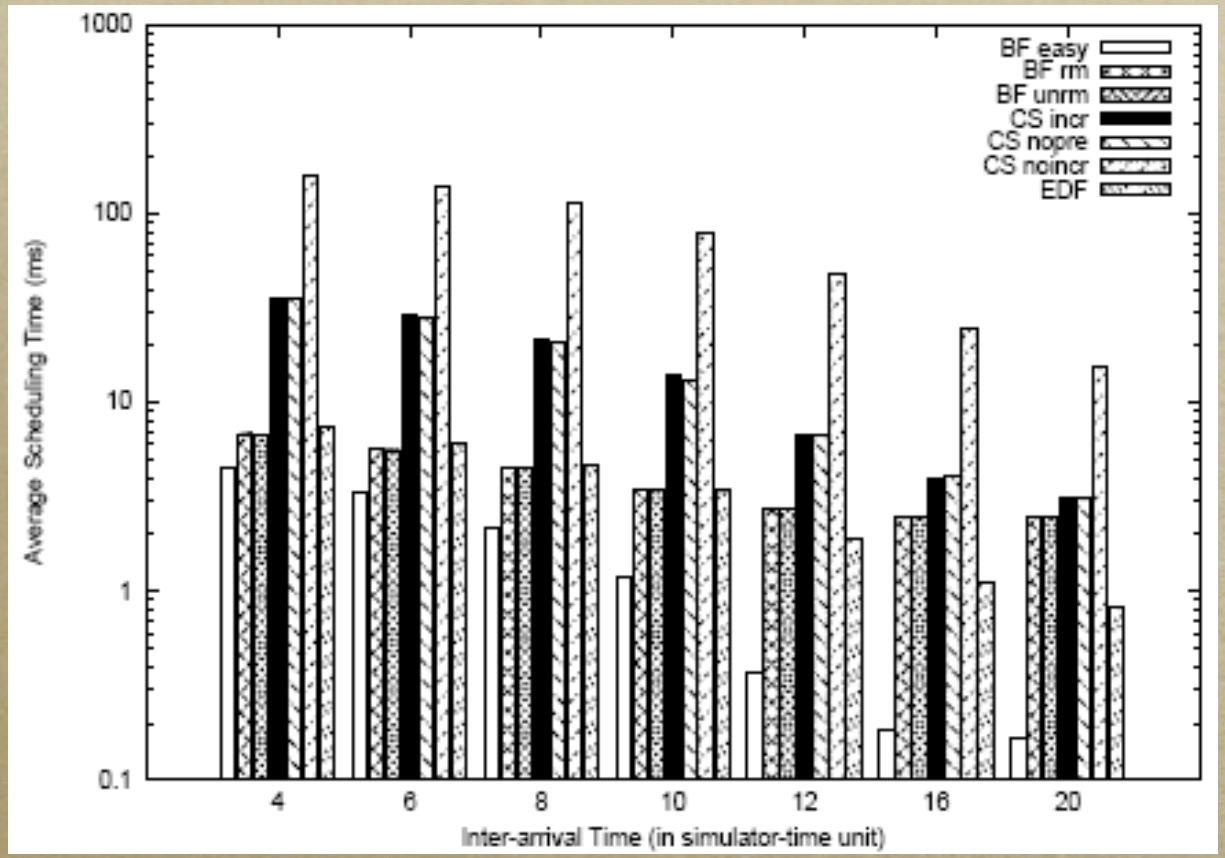


Percentage of machines usage.

This measure permits us to roughly figure out the system utilization.

$$System\_Usage = \frac{\#\,active\ \ machines}{min(\#\,available\ \ machines,\ \#\,jobs\ \ within\ \ the\ \ system)}$$

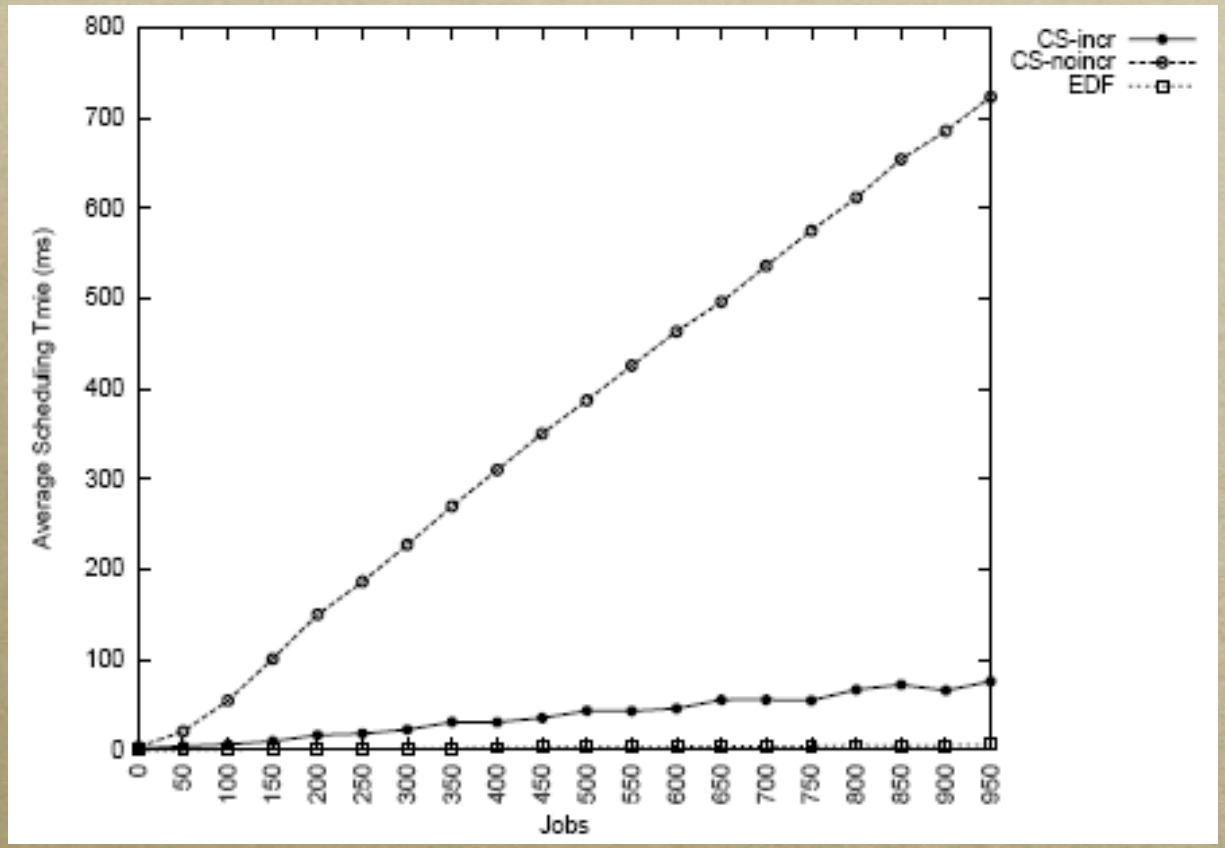
Experimental Simulation [8]



Average scheduling times.

As expected, the CS not incremental version requires more execution time.

Experimental Simulation [9]



Algorithm scalability

It was evaluated measuring the time need to carry out a new scheduling plan increasing the number of jobs.

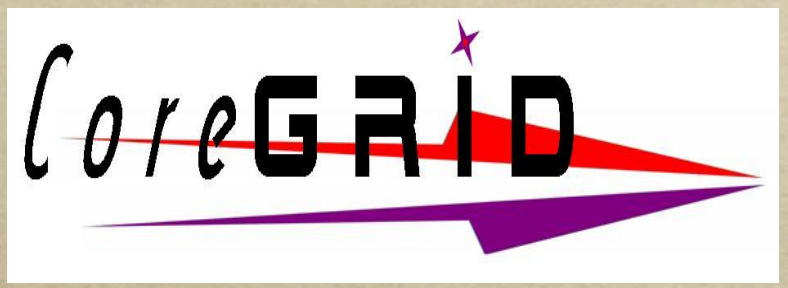
Conclusion

- *We have two algorithms:*
  - *Max Flow $O(n^4)$*
  - *Incr. Max Flow $O(n^3)$*
- *All reach very good usage (97% to 100%)*
- *Similar number of late jobs*

Coming soon

- *The sequence, number and weights of heuristics can be determined by a learning system*
  - *It gets feedback from chosen metrics*
  - *It can evolve while running!!*
- *With real data we can simulate and evolve a very precise scheduler*
- *More heuristics and parameters (costs, revenues) can be easily added*
- *Possibly, multi-level scheduling*

Acknowledgement



This work will appear to the Supercomputing conference '07. November 10-16, 2007. Reno, Nevada, USA