CoreGrid Meeting
University of Dortmund
13,14 March 2008

Open Issues in SLA Aware Systems

Viktor Yarmolenko

Viktor Yarmolenko
viktor@cs.man.ac.uk
The Univercity of Manchester
School of Comuter Science
Kilburn Building, Oxford Road
Manchester M13 9PL
United Kingdom

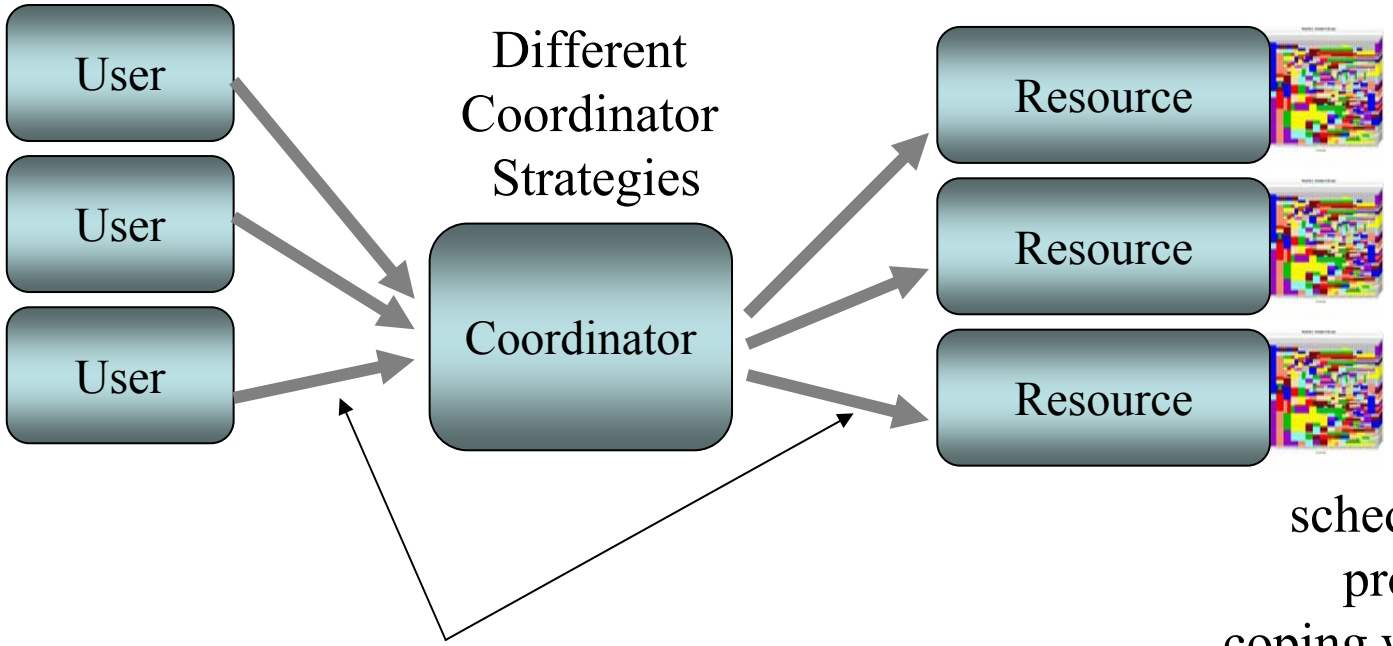
SLA Aware Systems

- Dynamic SLAs to tackle dynamic workflows
- Dynamic SLAs to improve negotiation
- Joint work with Michael (Barcelona)
- Dynamic SLAs and pricing models including the notion of trust (ongoing work with Cardiff)
- SLA Standards (WS-Agreement) to include workflows, preliminary discussions with Philipp (Dortmund)
- SLA Workload Generator

Simulation Example

Job Generator – producing different user behaviour, job workloads, *etc.*

Different number of Resources, each of different capacity, availability and other properties



Different negotiation protocols

Different scheduling algorithms, profit optimisations, coping with uncertainties, *etc.*

All this is in the context of Service Level Agreements.

SLA Representation: Traditional

T_S – *the earliest time the Job is allowed to start*

T_F – *the latest time the Job is allowed to finish*

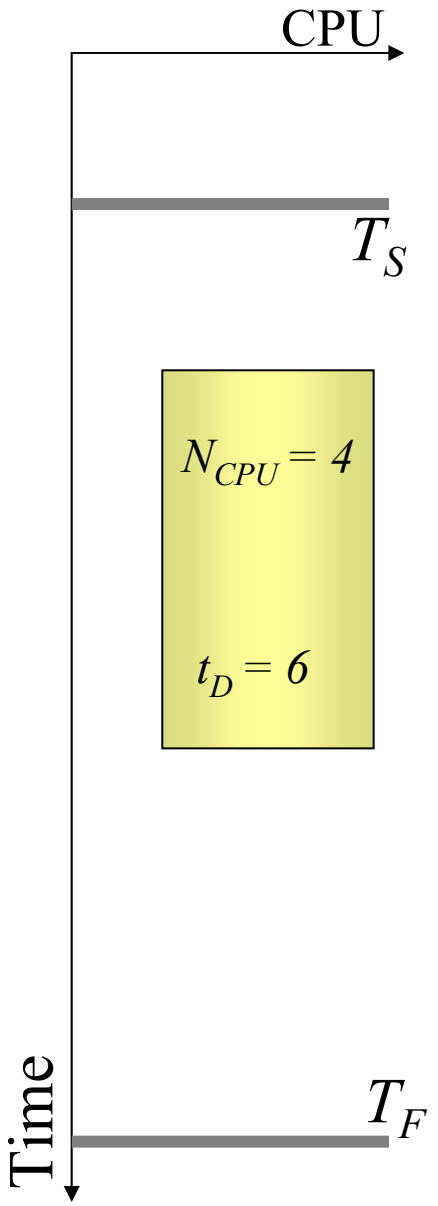
N_{CPU} – *number of nodes required for the Job*

t_D – *projected Job duration time for* N_{CPU} *nodes*

B_{job} – *projected traffic that Job creates*

V_{pr} – *the price for executing the Job*

V_{pn} – *the penalty for failing the Job*



SLA Representation: Expressive

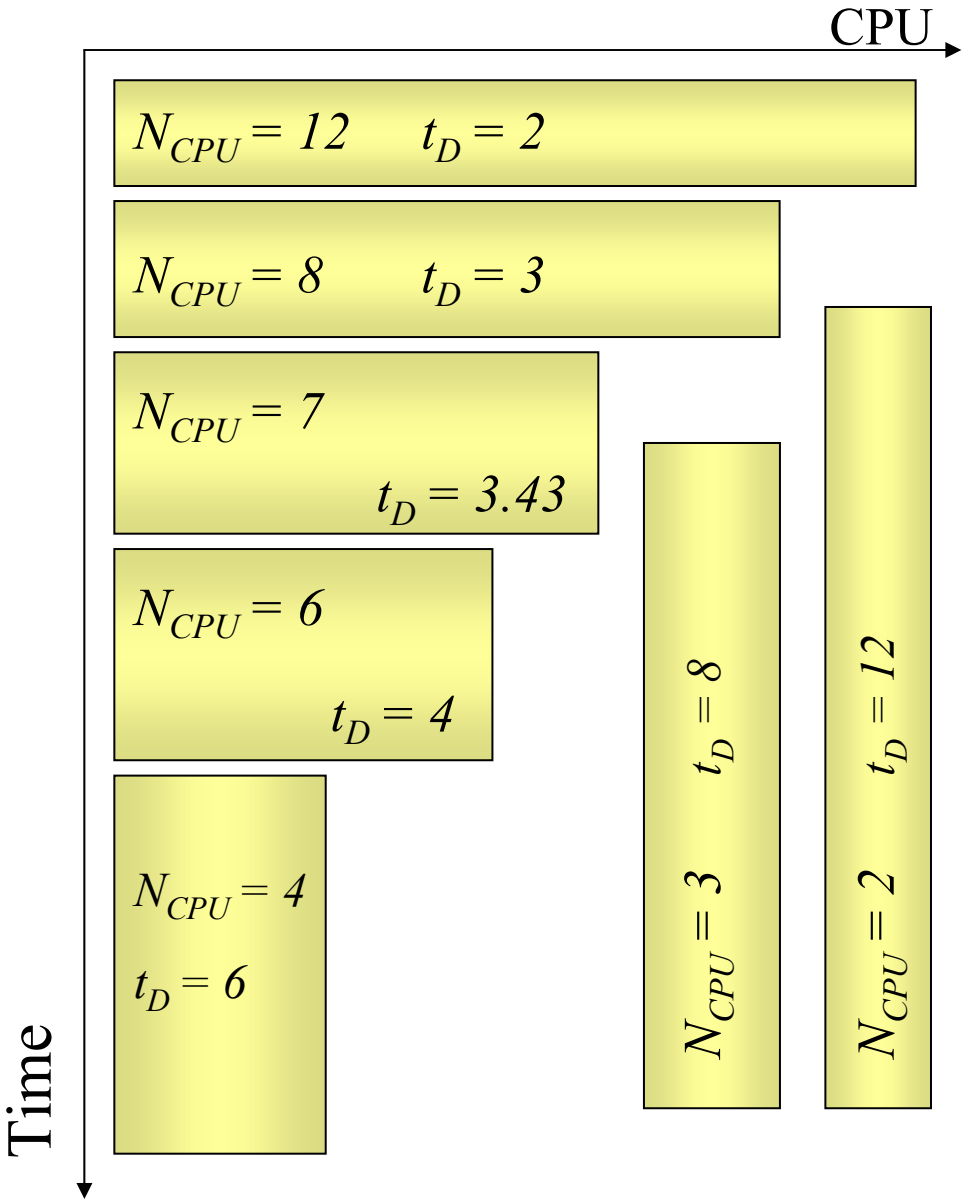
T_S, T_F, ... as before but …

$N_{CPU} = \{2,3,4,..\}$ is a range

$t_D = \dfrac{t_{UP}}{N_{CPU}}$ is a function

$t_{UP} = 24$, (CPU-hours) duration

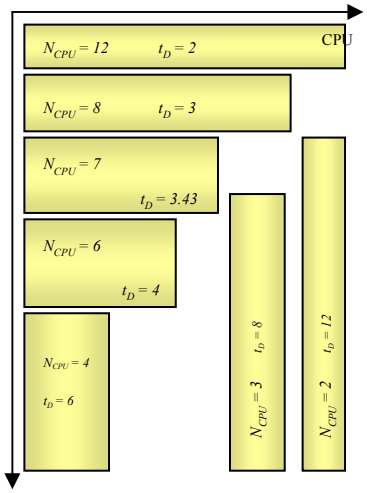
$V_{tot} = X\, t_D\, V_{pr}$ is a final value of the agreement (for example)



SLA Representation: Expressive

Same as before, but …

$B_{RES}(t_{curr})$, bandwidth provided by the Resource



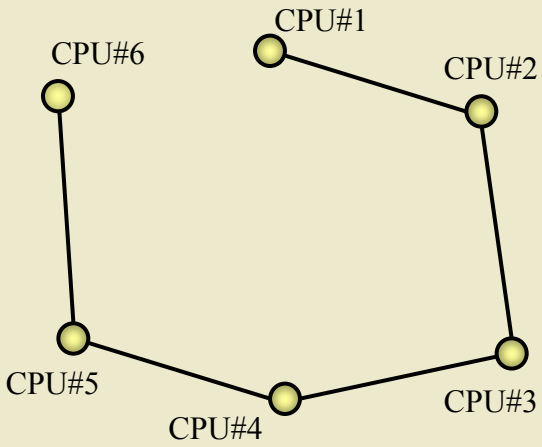
$d(n) = n+(n-1)+\ldots+2+1$, *a known expression*

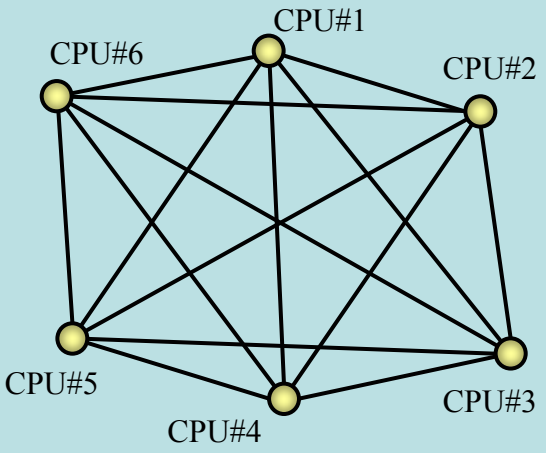
$B_{job} = B_0\, d(N_{CPU} - 1)$, traffic generated by the Job

$$t_D = \frac{B_{job}\, t_{UP}}{B_{RES}\, N_{CPU}} = \frac{B_0\, t_{UP}\, (N_{CPU} - 1)}{2B_{RES}}$$

$$t_D = \frac{B_{job}\, t_{UP}}{B_{RES}\, N_{CPU}} = \frac{B_0\, t_{UP}\, (N_{CPU} - 1)}{N_{CPU}\ B_{RES}}$$

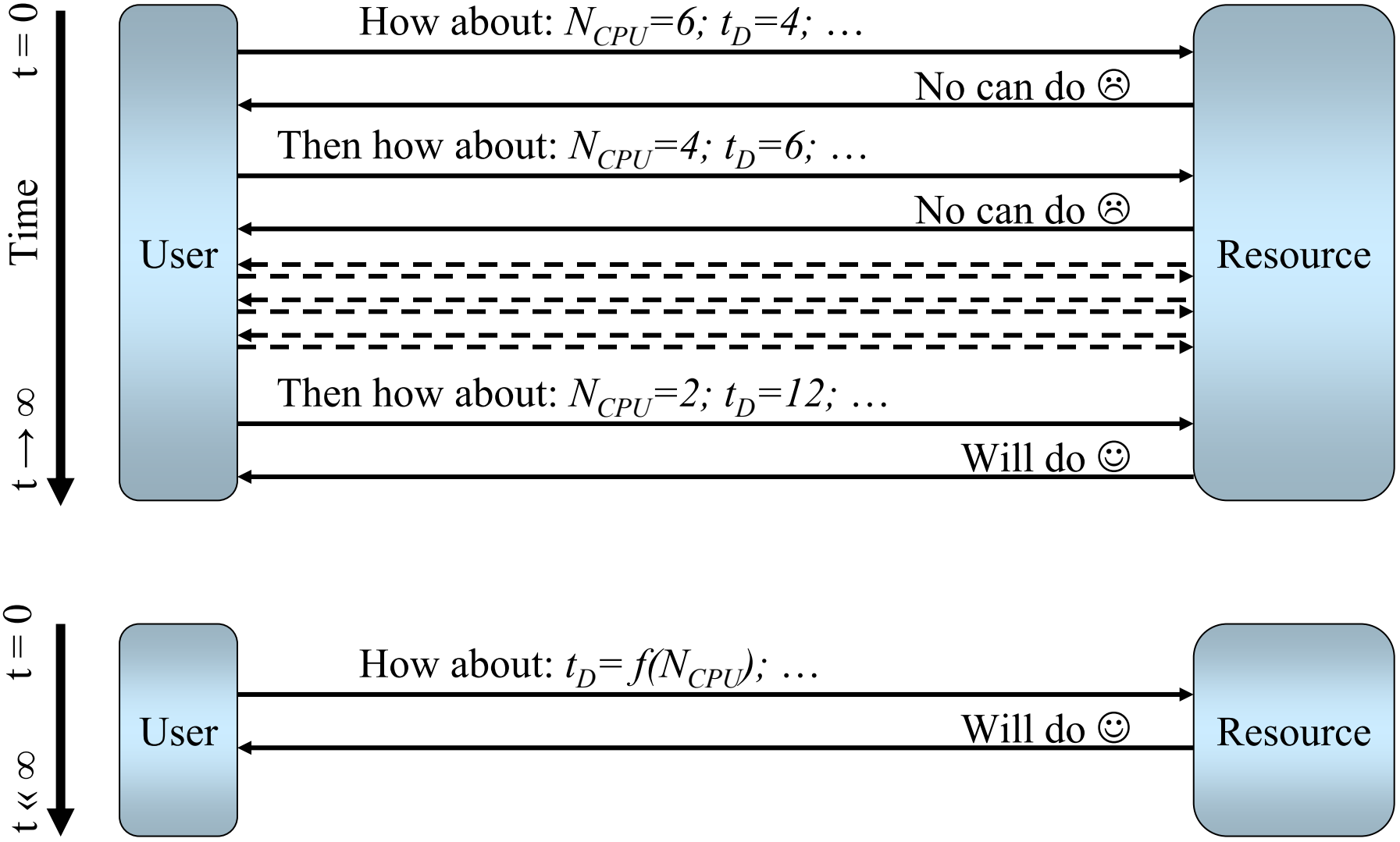
$B_{job} = B_0\, (N_{CPU} - 1)$





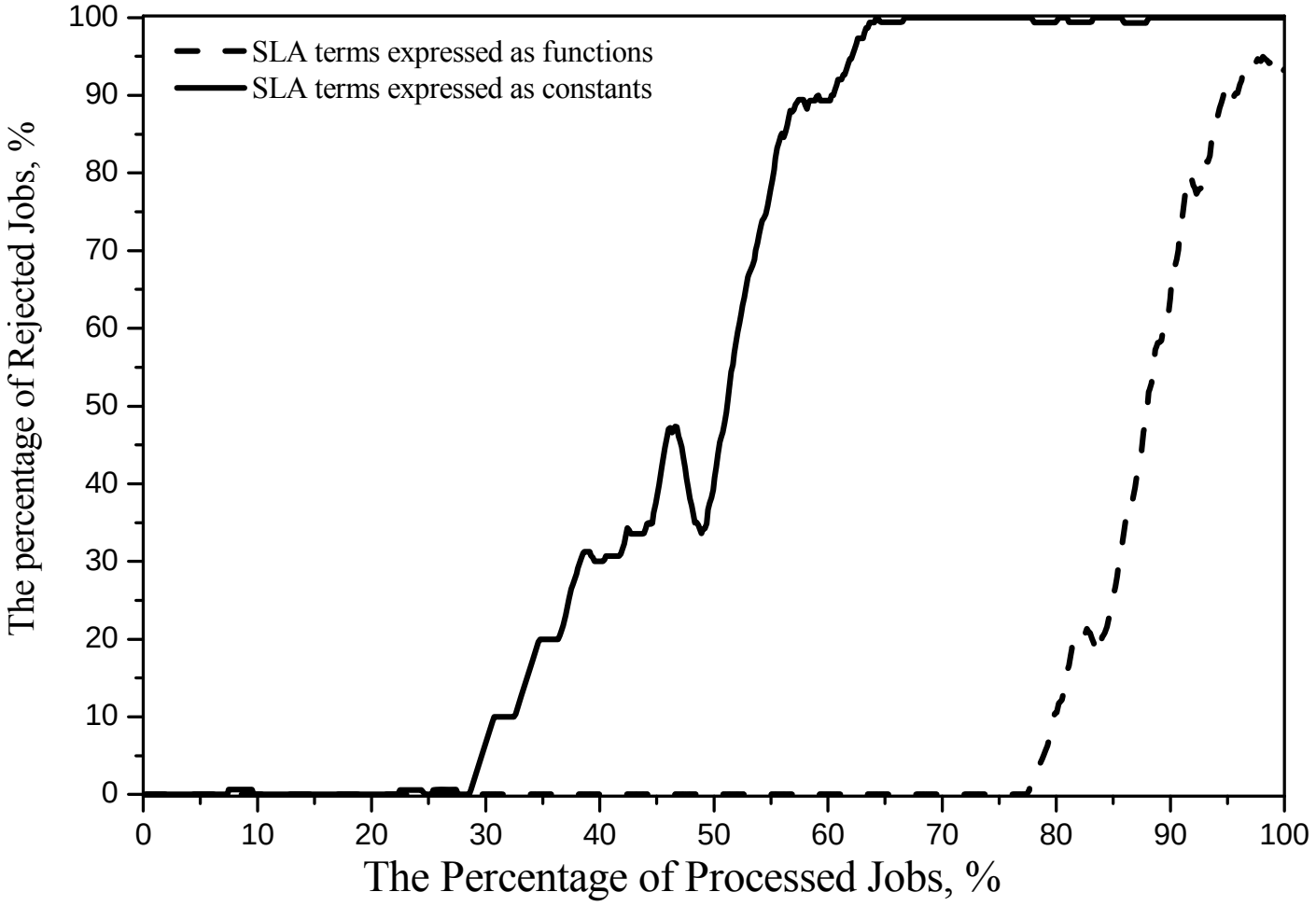
SLA Representation: Negotiation

Variable CPU Scenario (Traditional vs. Expressive SLA)



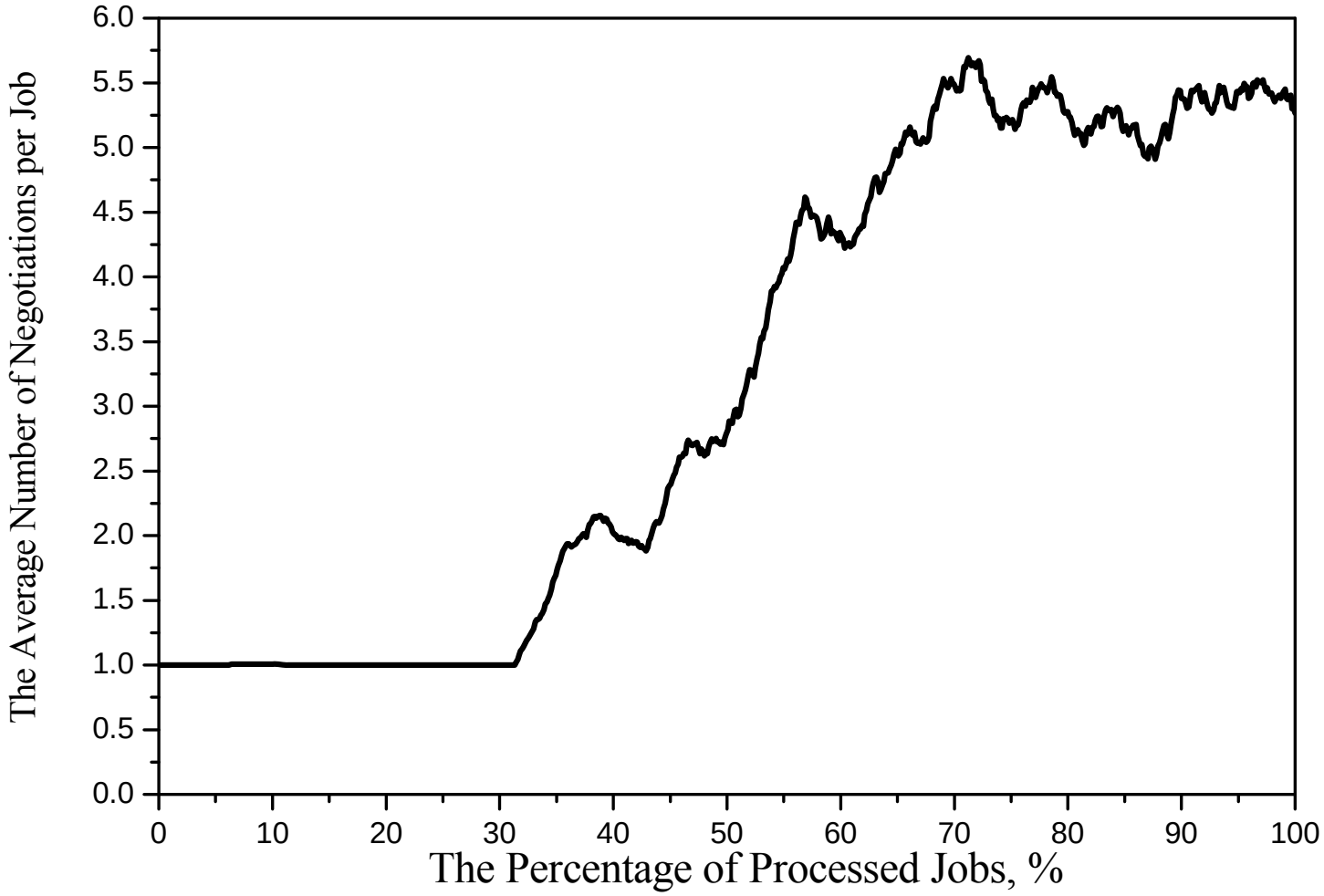
SLA Representation: Negotiation

Only Single Negotiation is Allowed



SLA Representation: Negotiation

Multiple Negotiations Allowed



Defining the Value of the Service

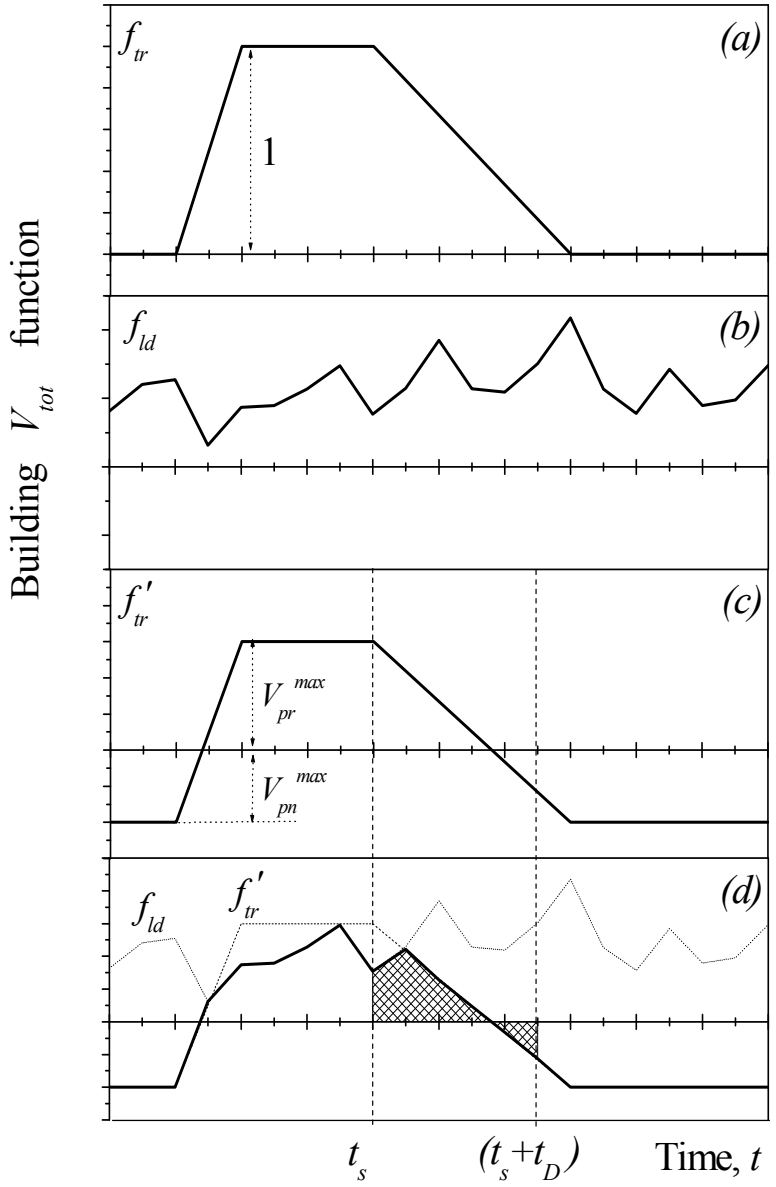
t_{curr}

$B_{RES}(t_{curr})$

$R_{ld}(t_{curr}) = f_{ld}$

$V_{tot} = f(R_{ld}, t_{D,} N_{CPU}, \ldots)$

Don't stop here, add fuzzy representation of SLA terms !!!



Dealing with Uncertainties

Challenges

- How to agree on something that you don't know?

- Even if the user has no idea about the execution time, can we still agree on price, for example?
- How to provide incentives to the user to "guess" the job duration.
- How provider can agree (SLA) on uncertain terms and guarantee QoS, be bound with penalties for not compliance (over-provisioning?) and still not go out of business

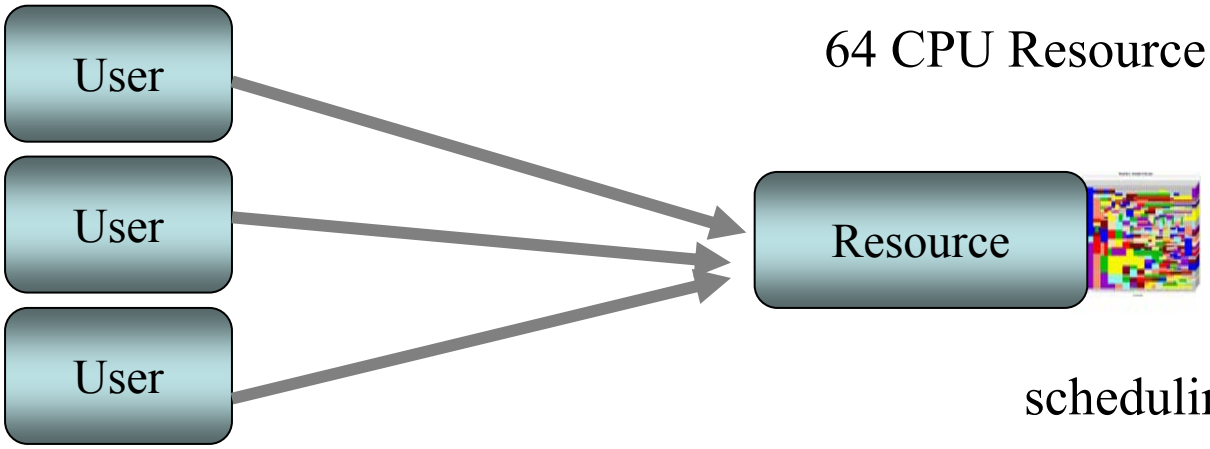
Dealing with Uncertainties

Solution Idea/Attempt

- We ask user to provide an estimate – for this user gets a cheaper price
- If user gets the timing right, he gets more discount
- If user gets it wrong user pays
- We want to make sure that user has the option to try and guess and gamble as well as to say, I have no idea, let me just reserve a job at a different rate.
- We want to make sure that the pricing function, which depends on user estimate of time duration (or any other SLA term) and its deviation from the actual value, encourages user to gamble. If user has even slghtest idea then it should be cheaper on average for him to try to guess.
- Provider than uses the information from the estimates, weighted by the confidence constant (or function) (how do we get this confidence?)
- The idea is that if such estimates are better than no estimates on average, the scheduler can utilise resources better, i.e. get more income for its resource, i.e. can either pass the savings to users in form of better discounts for guessing the job duration.
- Problems: getting the function right, with the penalties and discounts and the correct shape.

Dealing with Uncertainties

Simulation Environment

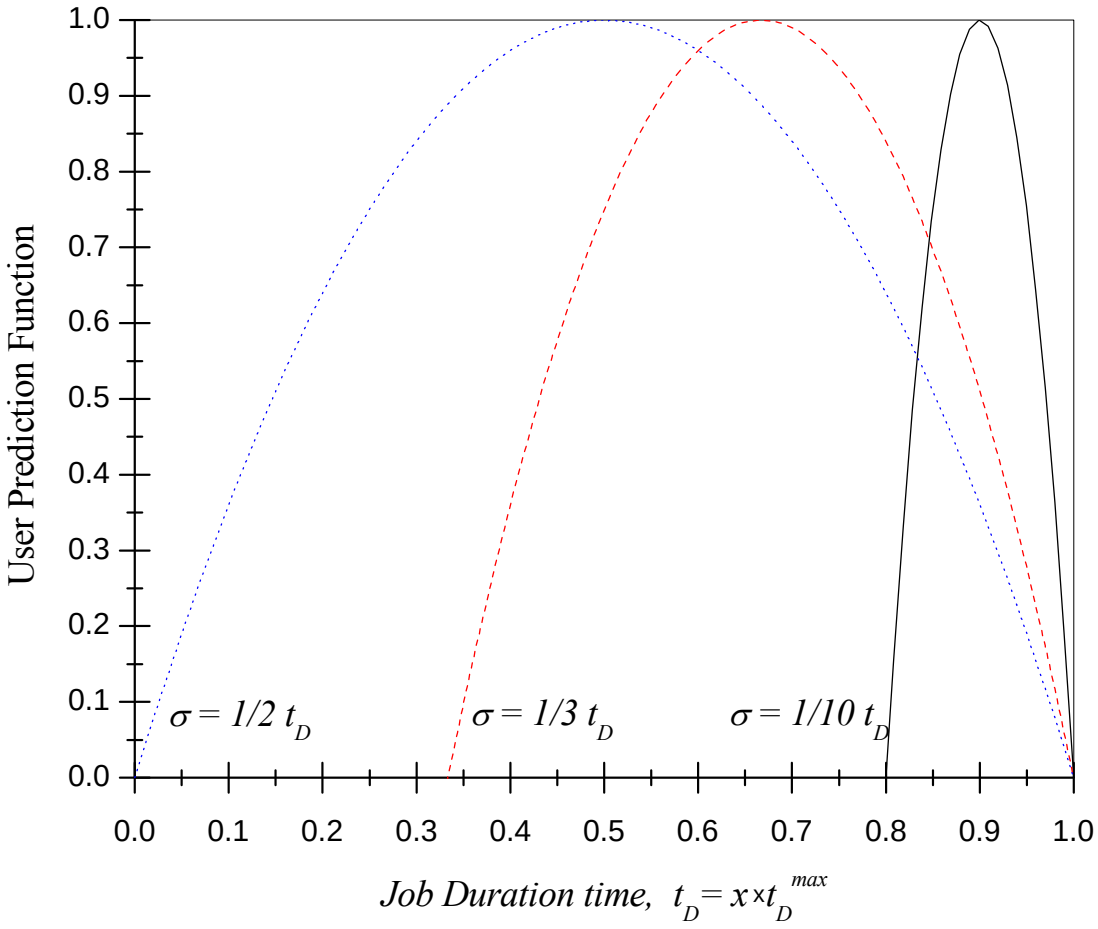


64 CPU Resource available for 200 hours

scheduling heuristics algorithms

Job Generator producing enough jobs, more than resource can handle: think of it as resource pulling jobs from the pool. Using combined models synthetic workload.

Simulating deviations from the estimated values in SLA



Distribution of runtimes, probability (in values of job runtime max, {0,1}

SLA Representation: Uncertain

T_S – *the earliest time the Job is allowed to start*

T_F – *the latest time the Job is allowed to finish*

N_{CPU} – *number of nodes required for the Job*

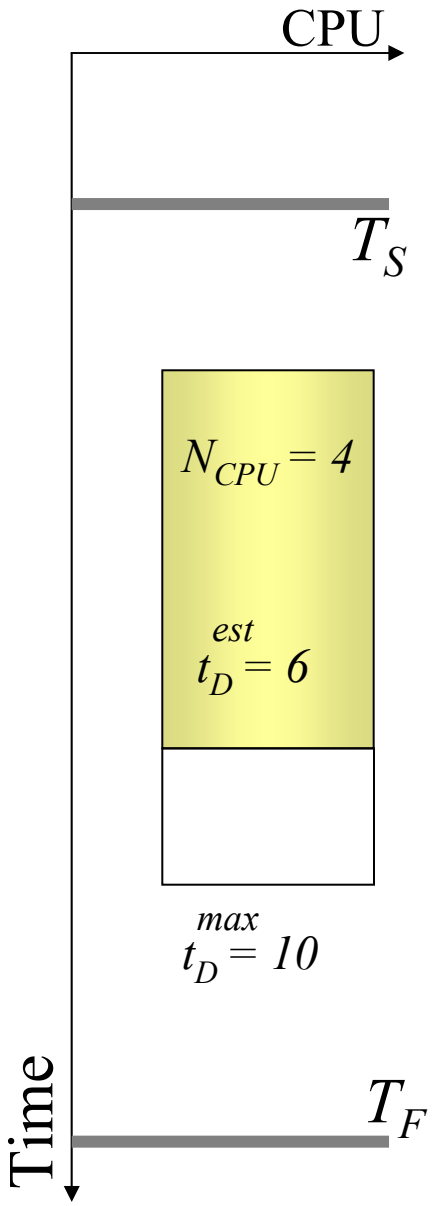
t_D^{est} – *projected Job duration time for* N_{CPU} *nodes*

t_D^{max} – *maximum allowed Job duration time*

.....

V_{pr} – *the price for executing the Job*

V_{pn} – *the penalty for failing the Job*



Dealing with Uncertainties

Preliminary results

- Utilisation using SLA that describes estimates (Uncertainty) on the same resource using the same workload and scheduling algorithm is:

20% better

Than SLA that simply uses max value for job duration.

- The amount of failed SLAs as a result of bad estimates for “uncertain” SLA is:

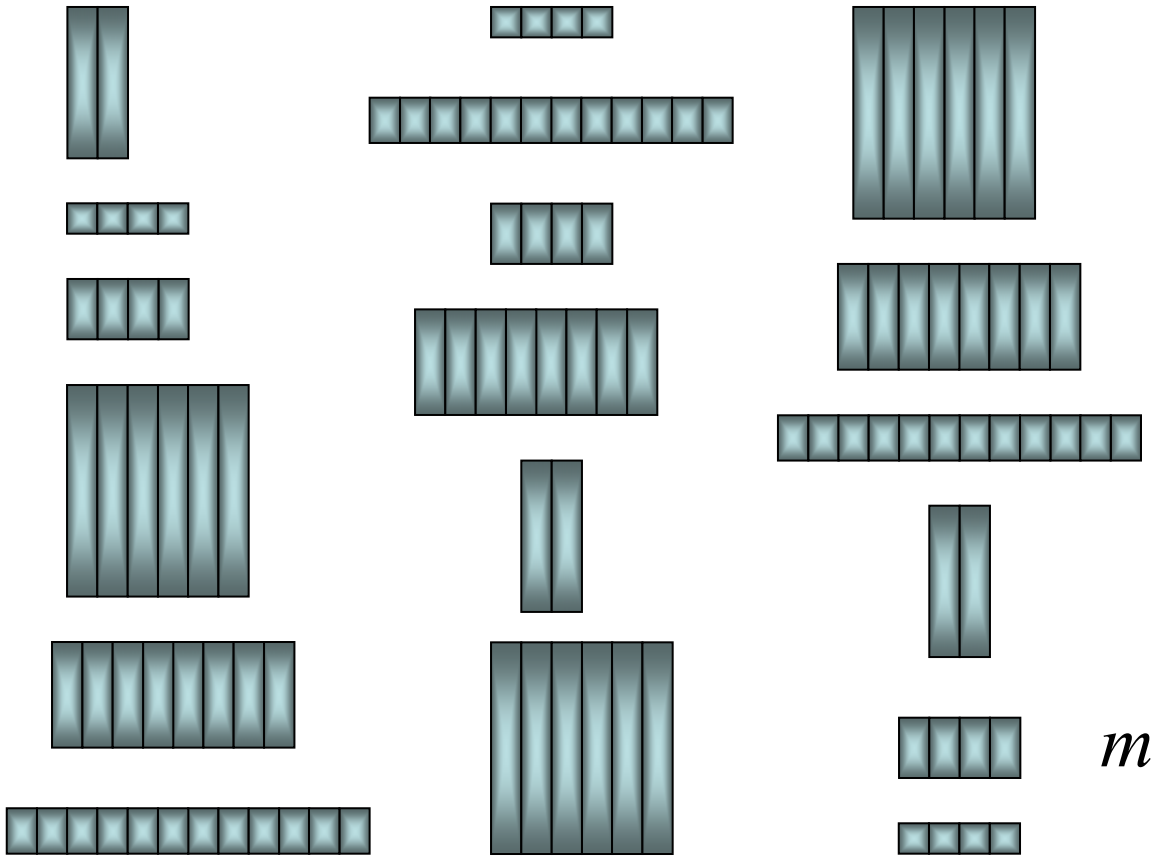
3-7%

Whereas conservative approach has no fails (we assume that resources never fail, so failures are due to bad scheduling)

Scheduling Heuristics

Prioritising Jobs

$min(N_{CPU})$ $min(t_D)$ $max(A)$



$min(T_S)$

$min(T_F)$

$min(t_L)$

$min(T_S + wt_D)$

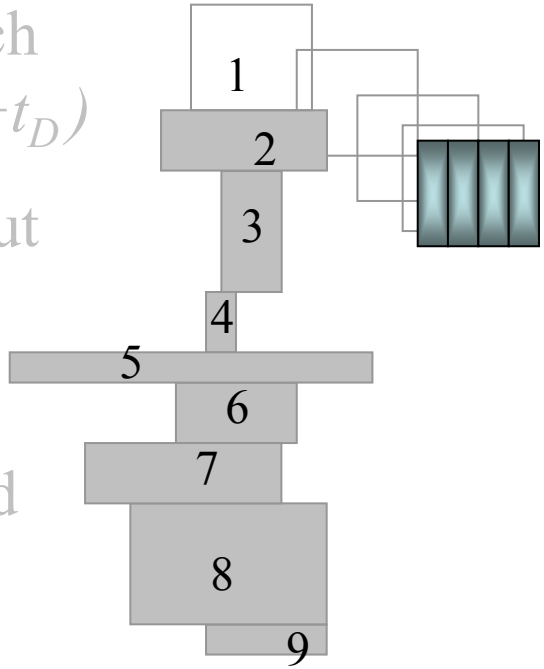
$max(T_F + wA)$

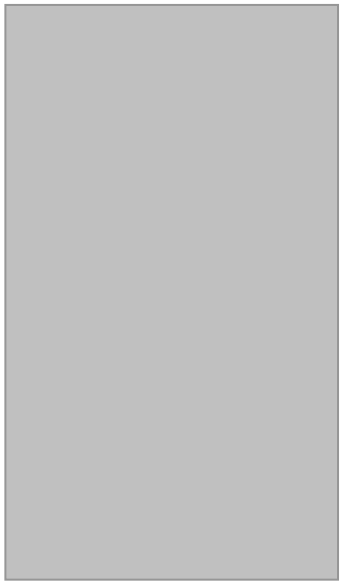
etc.

$min(T_F + w_1A + w_2t_L)$

Allocating Jobs

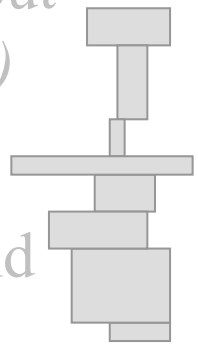
1. Pick up the next job on the list
2. Try to find N_{CPU} nodes which are available from T_S to (T_S+t_D)
3. If unsuccessful, try step 2 but with $(T_S + \Delta t)$ to $(T_S+\Delta t+t_D)$
4. Repeat steps 2 and 3 while $(T_S+\Delta t+t_D) < T_F$ or until find enough free nodes
5. If failed to find N_{CPU} nodes, reject the request.

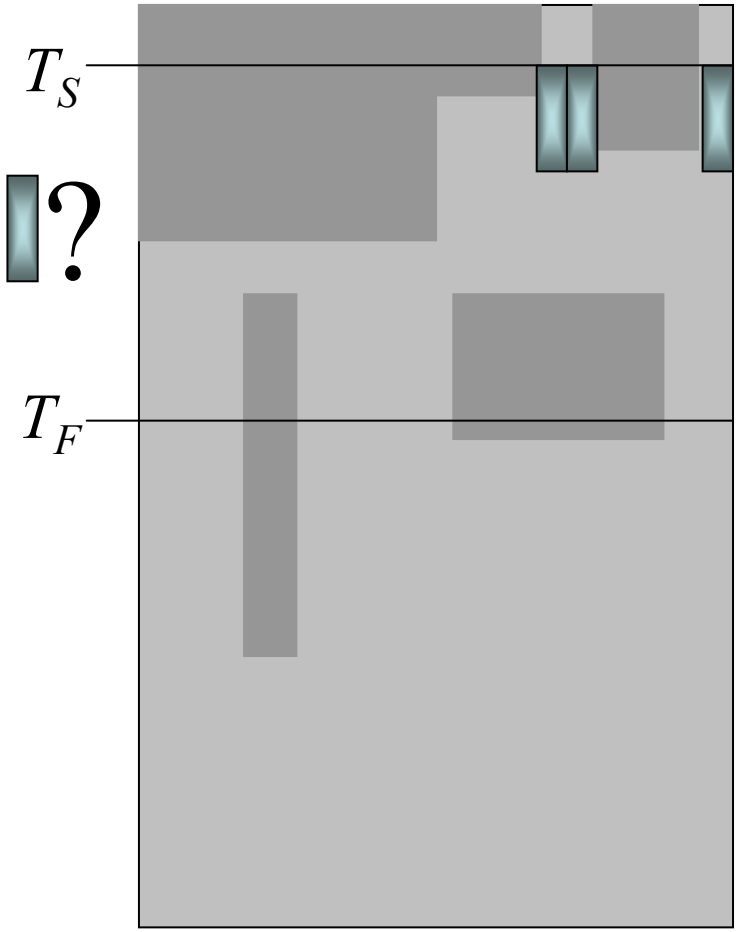




Allocating Jobs

1. Pick up the next job on the list
2. Try to find N_{CPU} nodes which are available from T_S to (T_S+t_D)
3. If unsuccessful, try step 2 but with $(T_S + \Delta t)$ to $(T_S+\Delta t+t_D)$
4. Repeat steps 2 and 3 while $(T_S+\Delta t+t_D) < T_F$ or until find enough free nodes
5. If failed to find N_{CPU} nodes, reject the request.

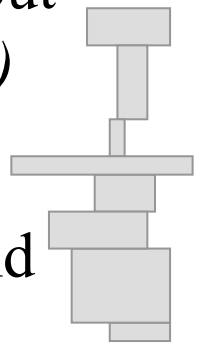


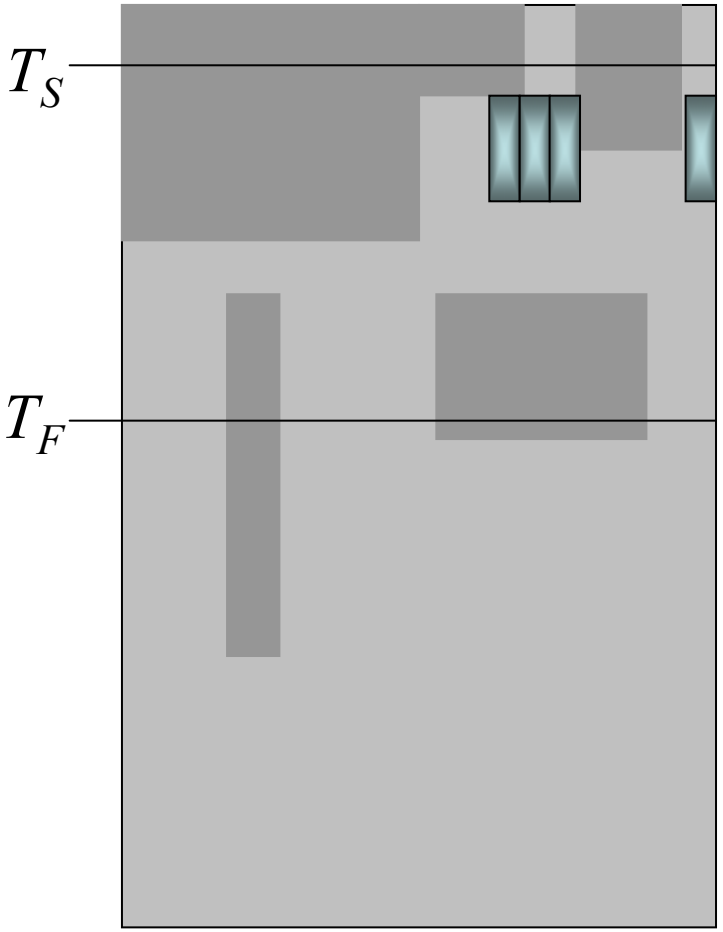


Scheduling Heuristics

Allocating Jobs

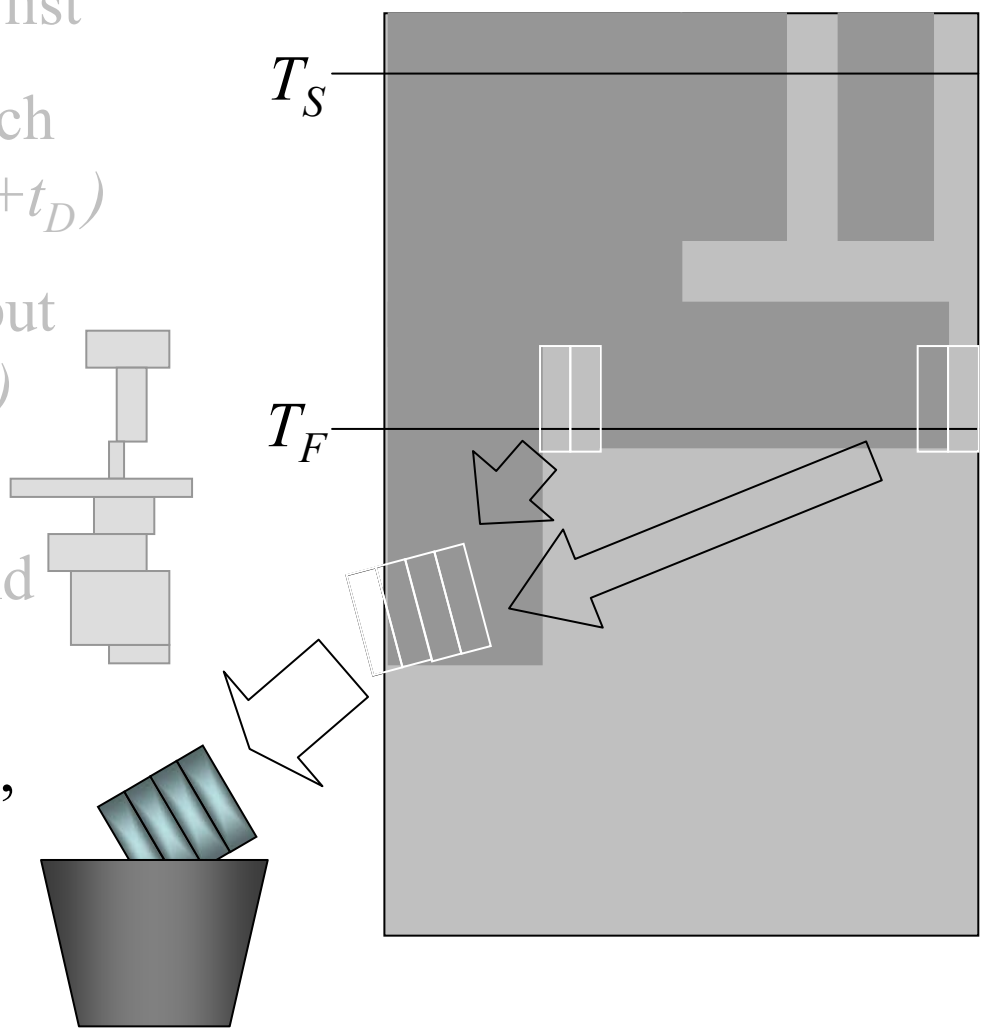
1. Pick up the next job on the list
2. Try to find N_{CPU} nodes which are available from T_S to (T_S+t_D)
3. If unsuccessful, try step 2 but with $(T_S + \Delta t)$ to $(T_S+\Delta t+t_D)$
4. Repeat steps 2 and 3 while $(T_S+\Delta t+t_D) < T_F$ or until find enough free nodes
5. If failed to find N_{CPU} nodes, reject the request.





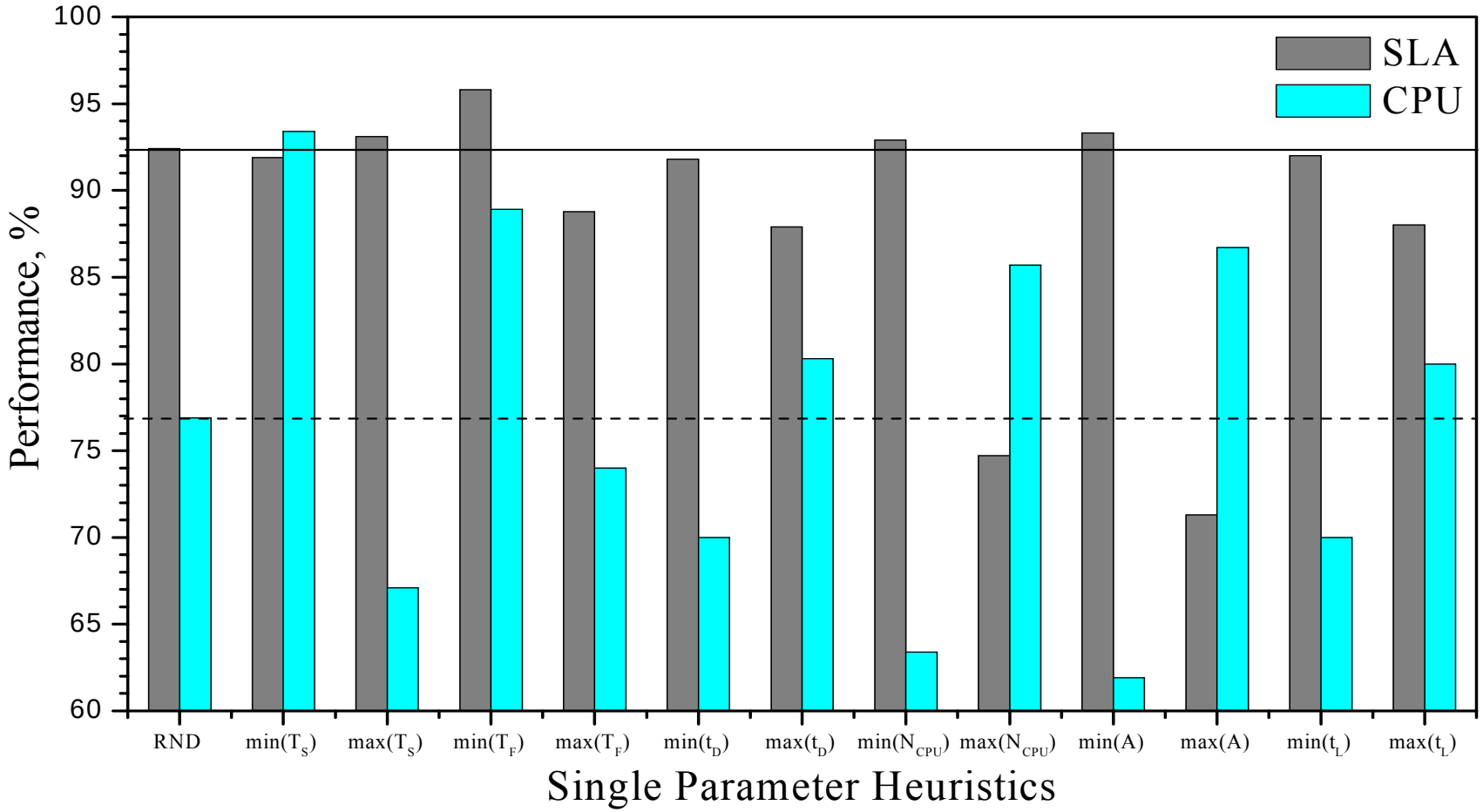
Allocating Jobs

1. Pick up the next job on the list
2. Try to find N_{CPU} nodes which are available from T_S to (T_S+t_D)
3. If unsuccessful, try step 2 but with $(T_S + \Delta t)$ to $(T_S+\Delta t+t_D)$
4. Repeat steps 2 and 3 while $(T_S+\Delta t+t_D) < T_F$ or until find enough free nodes
5. If failed to find N_{CPU} nodes, reject the request.

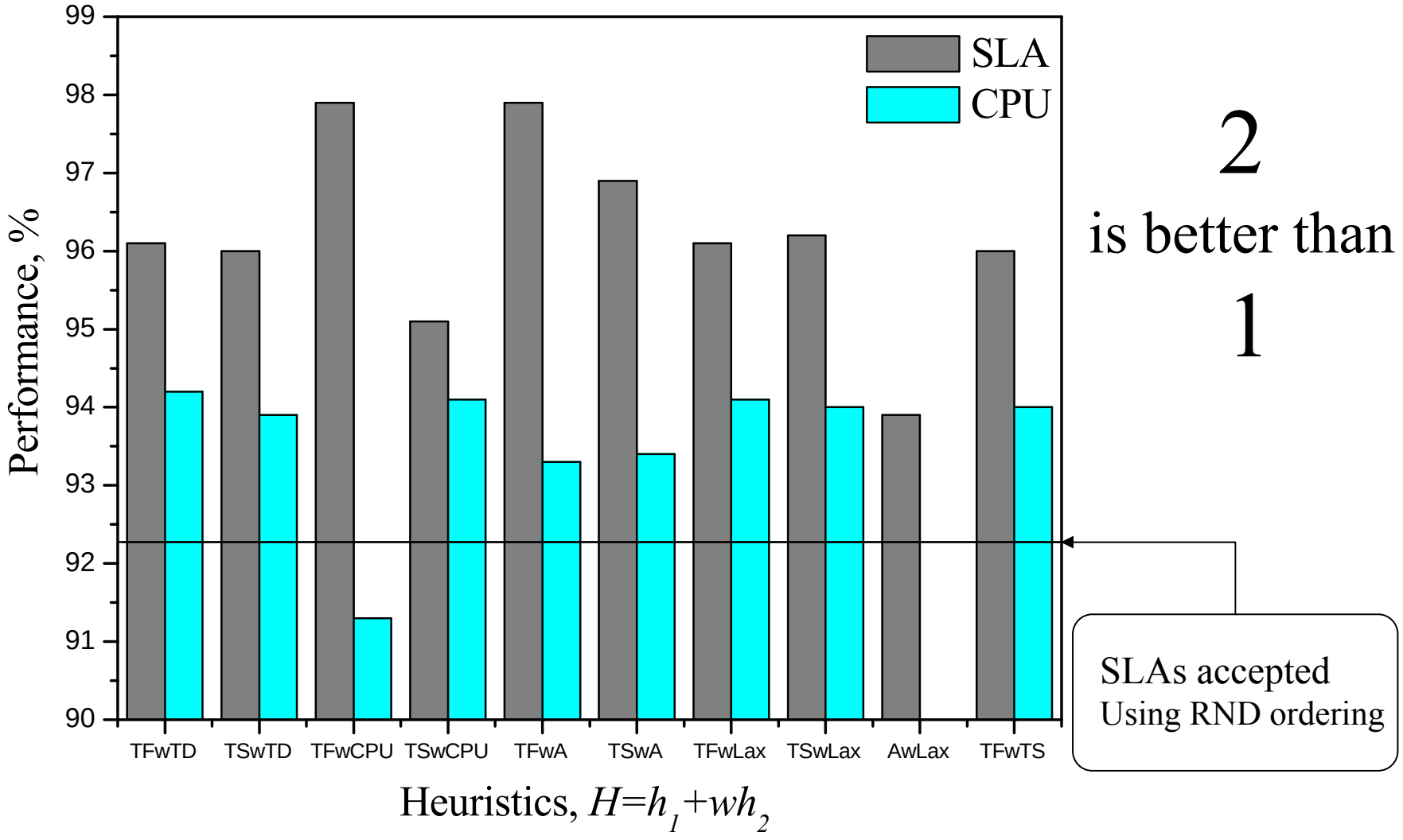


Results: Single Parameter Ordering

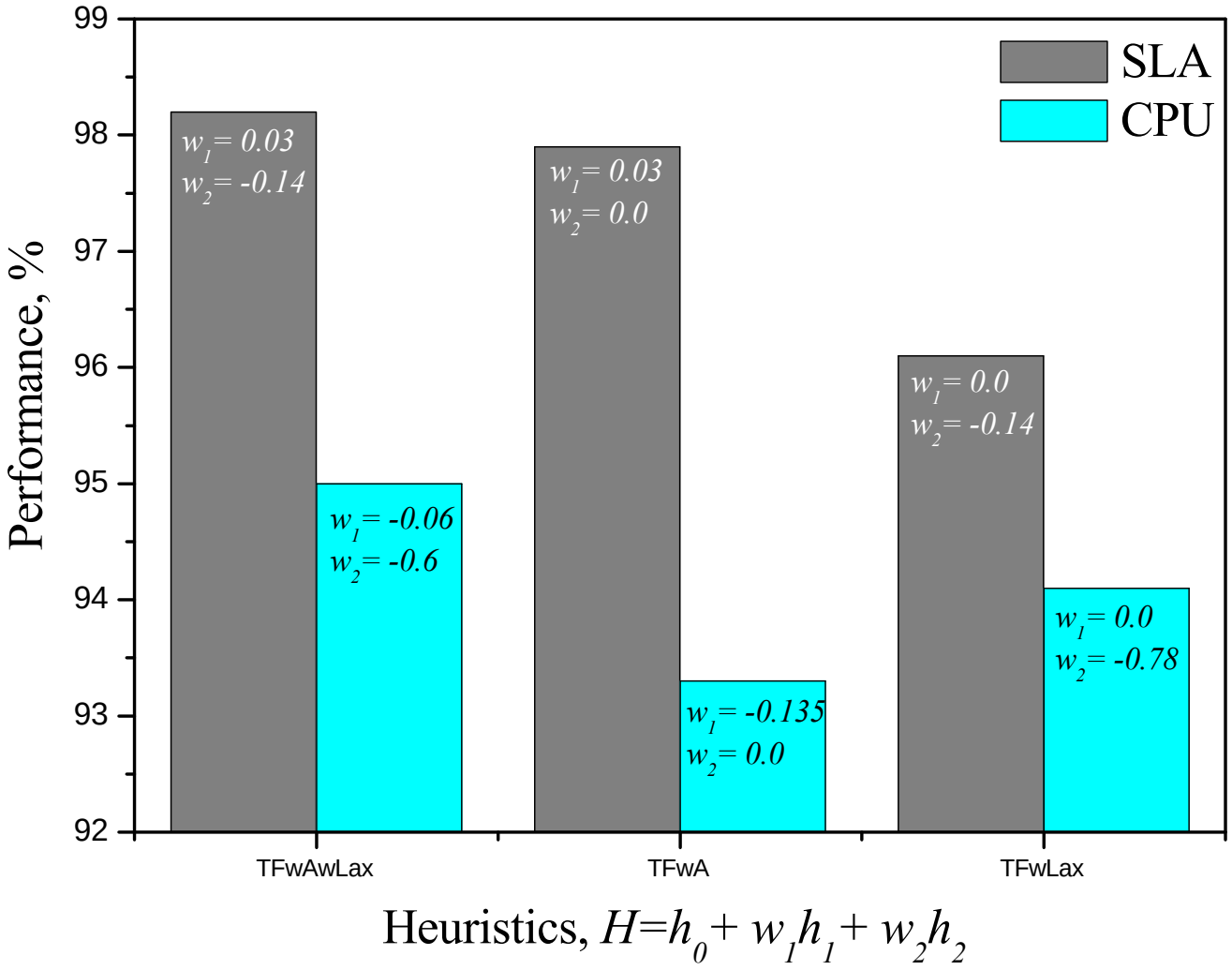
Order jobs in ascending (*min(H)*) or descending (*max(H)*) order of *H*.



Results: Two Parameter Ordering



Results: Three Parameter Ordering



3

is better than

2

$h_0 = T_F,$
$h_1 = A,$
$h_2 = t_L,$

Scheduling Heuristics

For best performance jobs must be always ordered by the lowest ⇩ or the highest ⇧ parameter first

Pricing	T_F	T_S	t_D	t_L	A	N_{CPU}
SLA	⇩	⇩	⇩	⇩	⇩	⇩
CPU	⇩	⇩	⇧	⇧	⇧	⇧

Other pricing policies were explored:

Viktor Yarmolenko, Rizos Sakellariou, AN EVALUATION OF HEURISTICS FOR SLA BASED PARALLEL JOB SCHEDULING, *Proceedings of the 3rd High Performance Grid Computing Workshop (HPGC)* (in conjunction with IPDPS 2006), Rhodes, Greece (April 2006), IEEE Computer Society Press

SLA Representation: Extended

T_S – *the earliest time the Job is allowed to start*

T_F – *the latest time the Job is allowed to finish*

N_{CPU} – *number of CPU nodes required for the Job*

t_D – *projected Job duration time for* N_{CPU} *nodes*

.....

t_{UP} – *uniprocessor Job duration time (CPU-hours)*

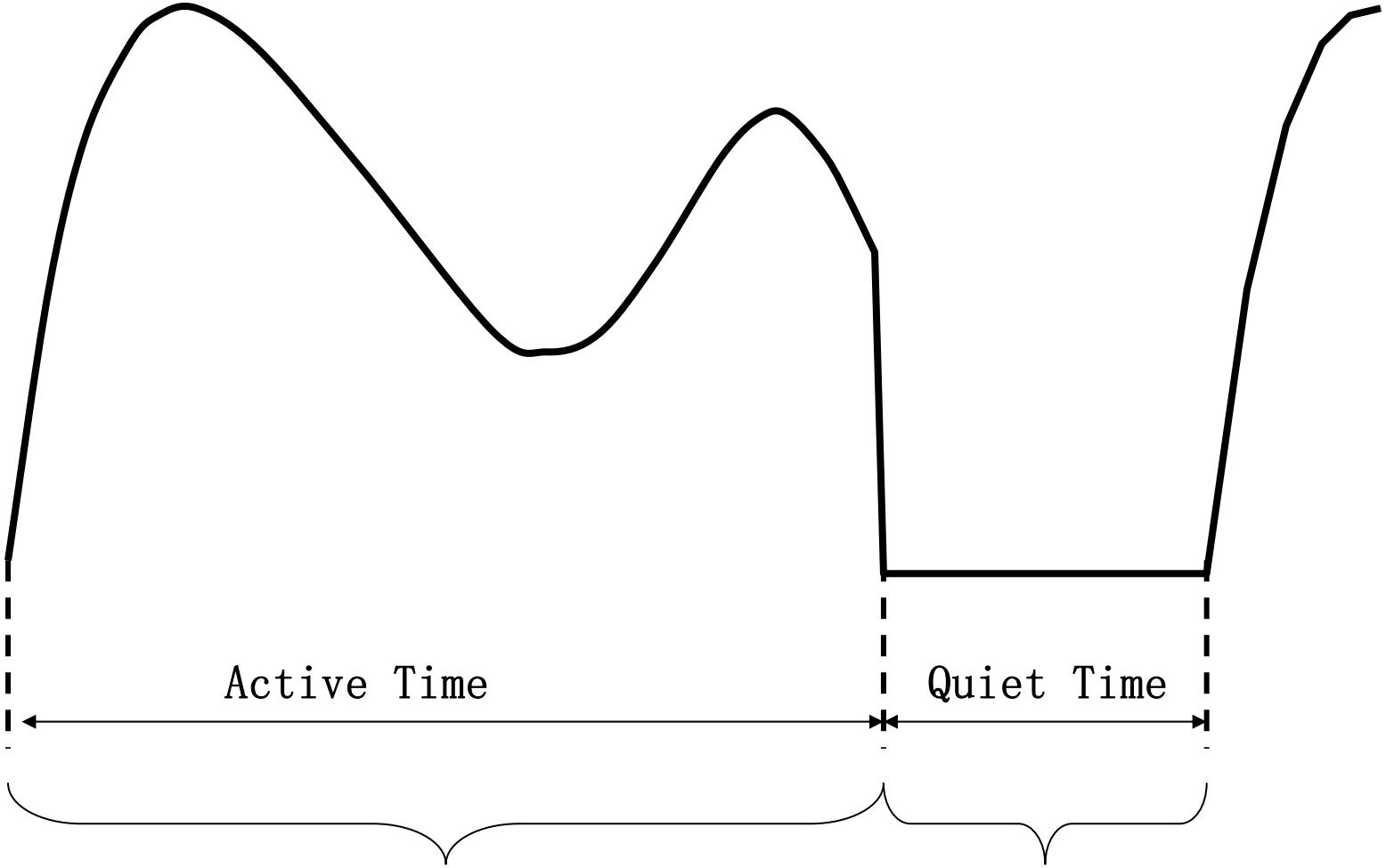
B_{job} – *projected traffic that Job creates*

V_{pr} – *the price for executing the Job*

V_{pn} – *the penalty for failing the Job*

V_{tot} – *final value of the agreement (optional)*

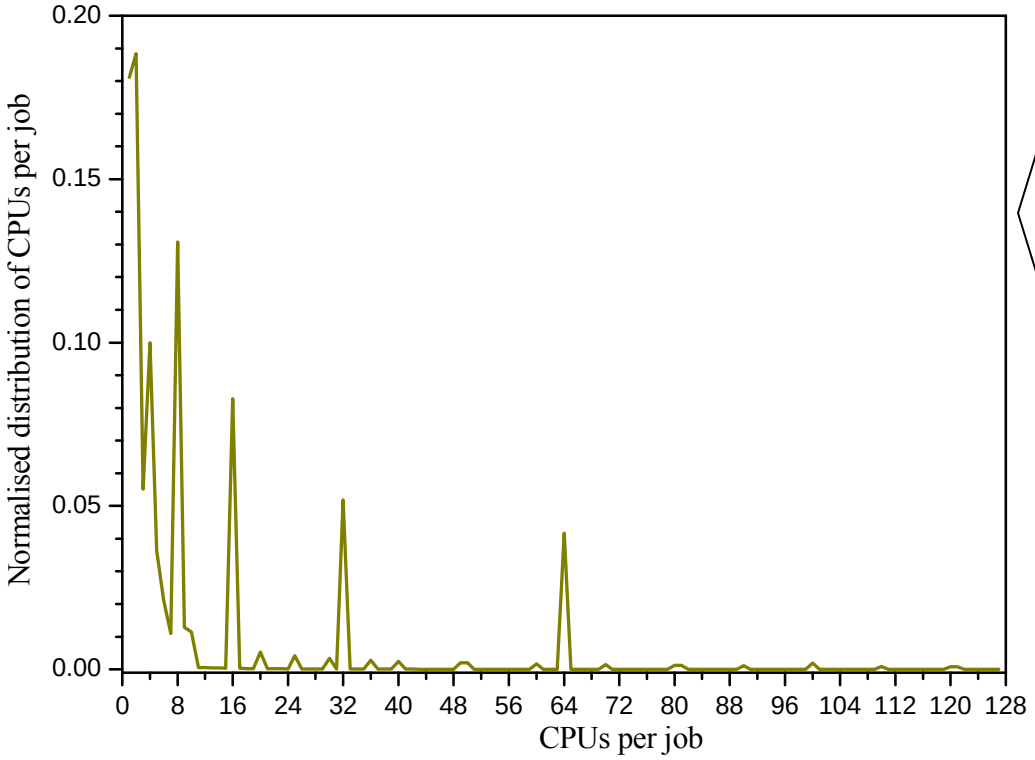
Modelling Variation in User Demand: Related Work



Calzarossa & Serrazi Model (1985) models the daily cycle of job submittals, with peaks in the morning and afternoon, and a drop at lunchtime.

Extended to include a night time, currently *const*, but can be modelled by any function *f(t)*.

Modelling Job Characteristics: Related Work



The Feitelson Model (1996)

Degrees of Parallelism: harmonic distribution of order 1.5, with the most popular sizes of powers of two.

Repeated Executions: the same job is likely to be executed again, $n^{-2.5}$,where n is the job duration.

Arrival Process: used in this model is Poisson.

Correlation of Runtime with Parallelism

Modelling SLA Constraints: Paper in Preparation

The constraint is represented as tightness:

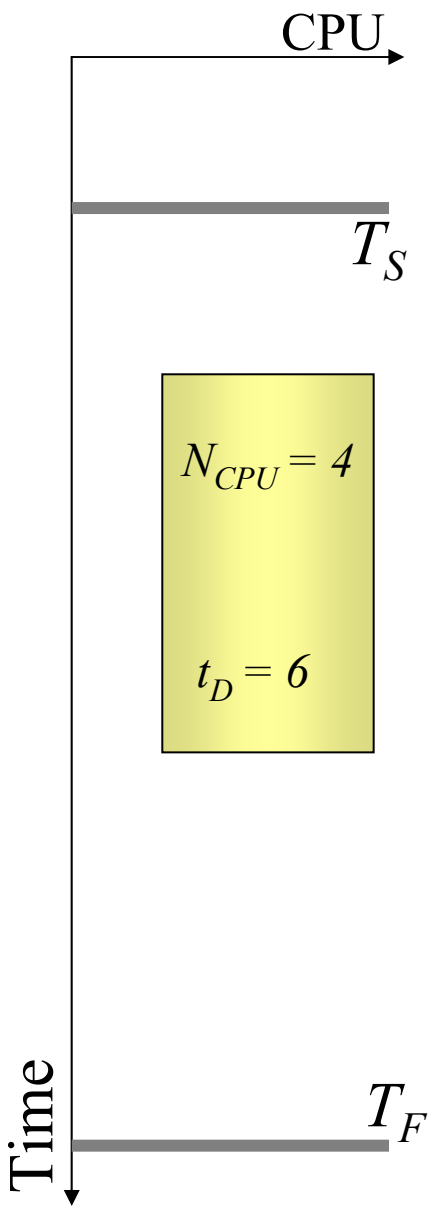
$$t_T = \frac{t_D}{T_F - T_S}$$

The distribution of tightness in the workload is described by a specific function $f_{TT}()$

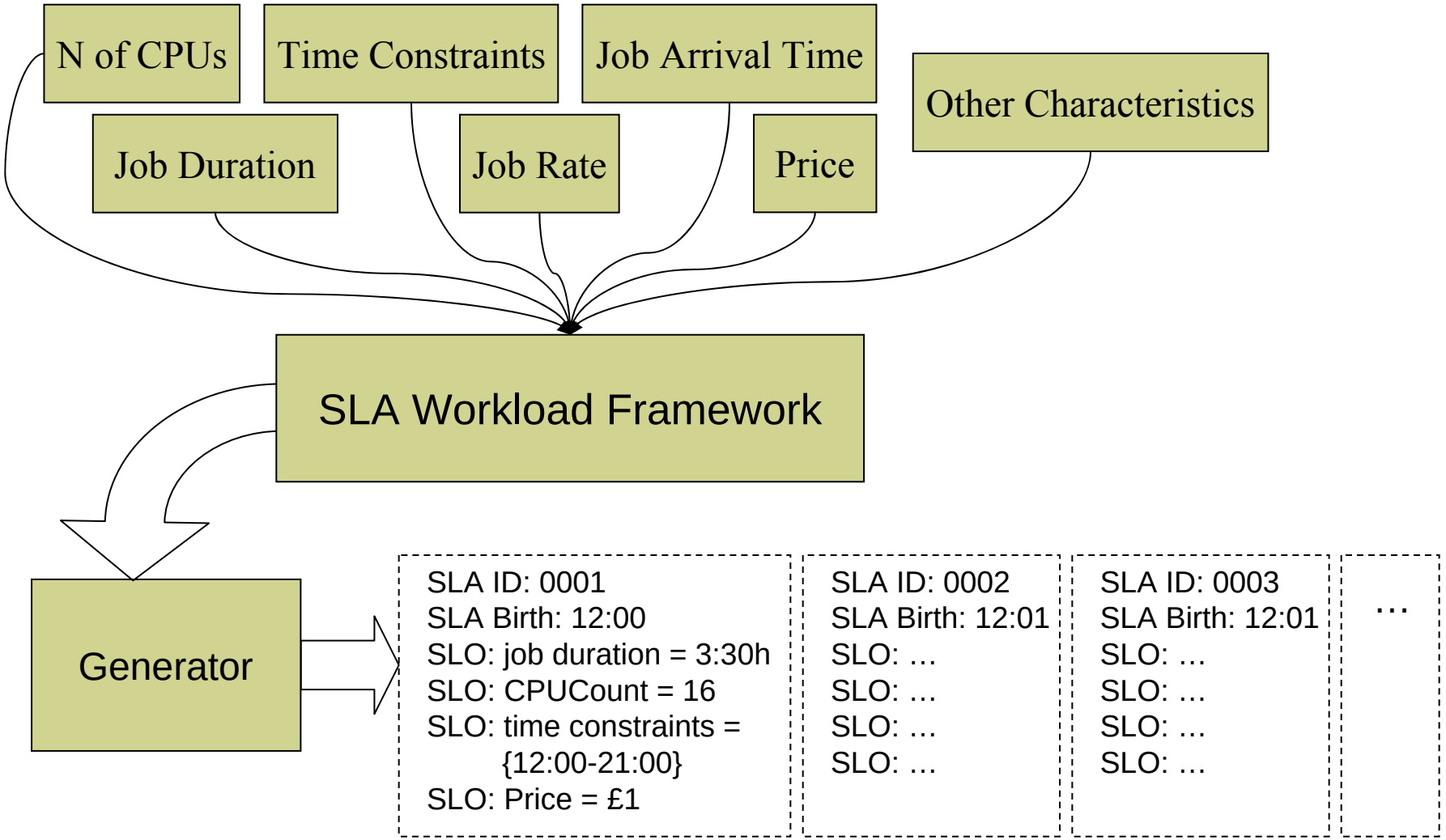
Current state of grid is such that this function is a discrete distribution with non-zero values for:

$t_T = 0$ – no constraints, deadlines

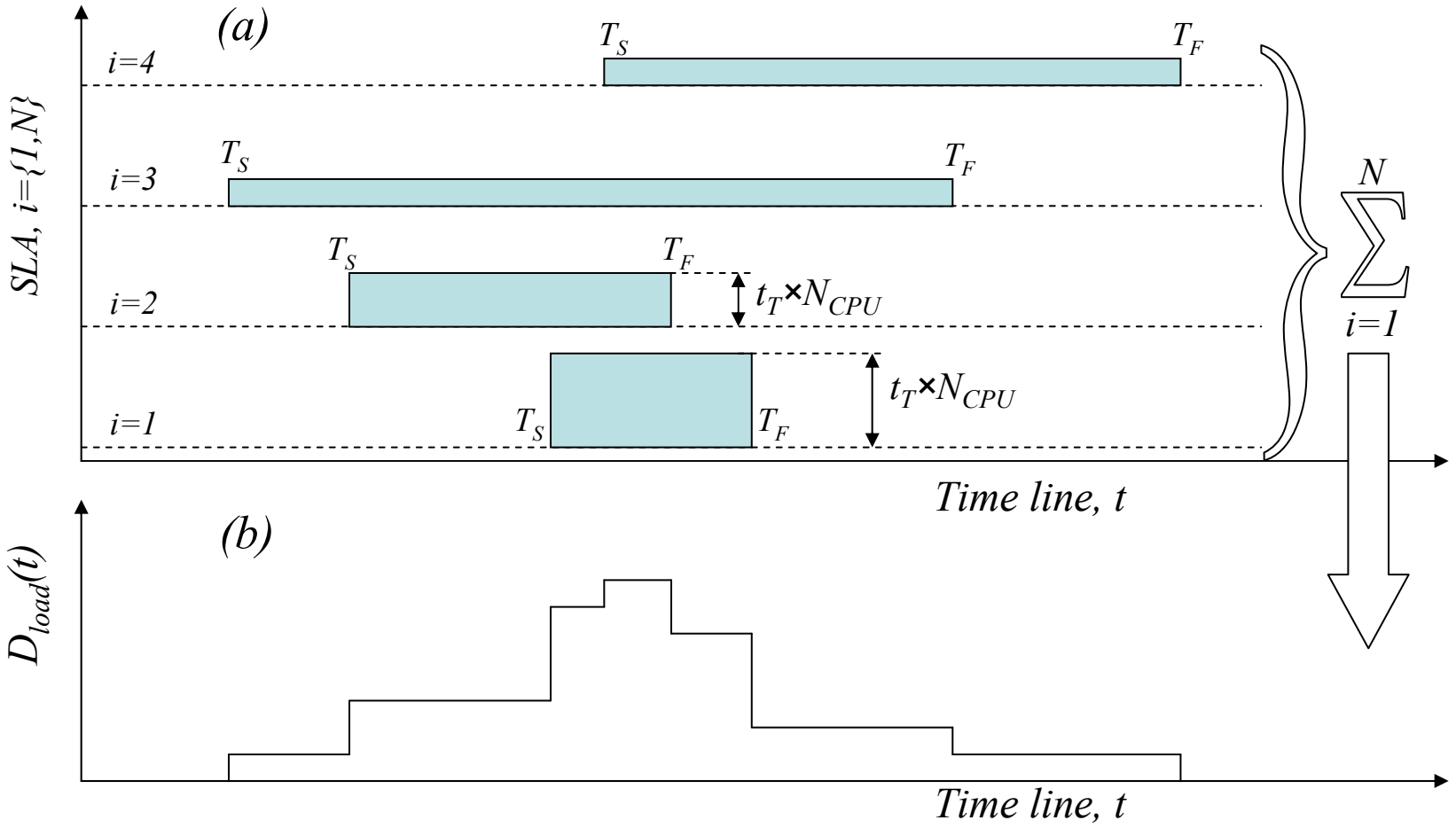
$t_T = 1$ – Advance Reservation



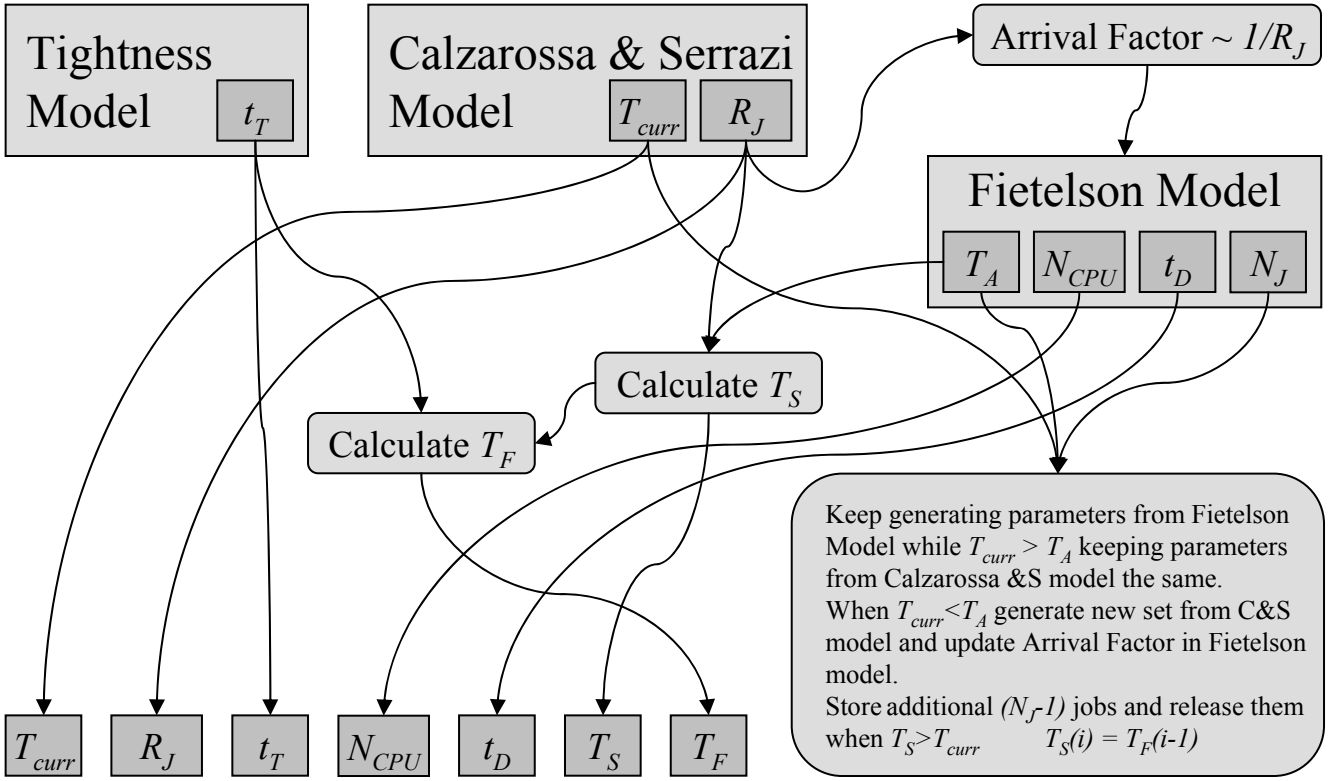
Workload Generator Framework: The Concept (paper in preparation)



Workload Generator Framework: What Metrics (paper in preparation)

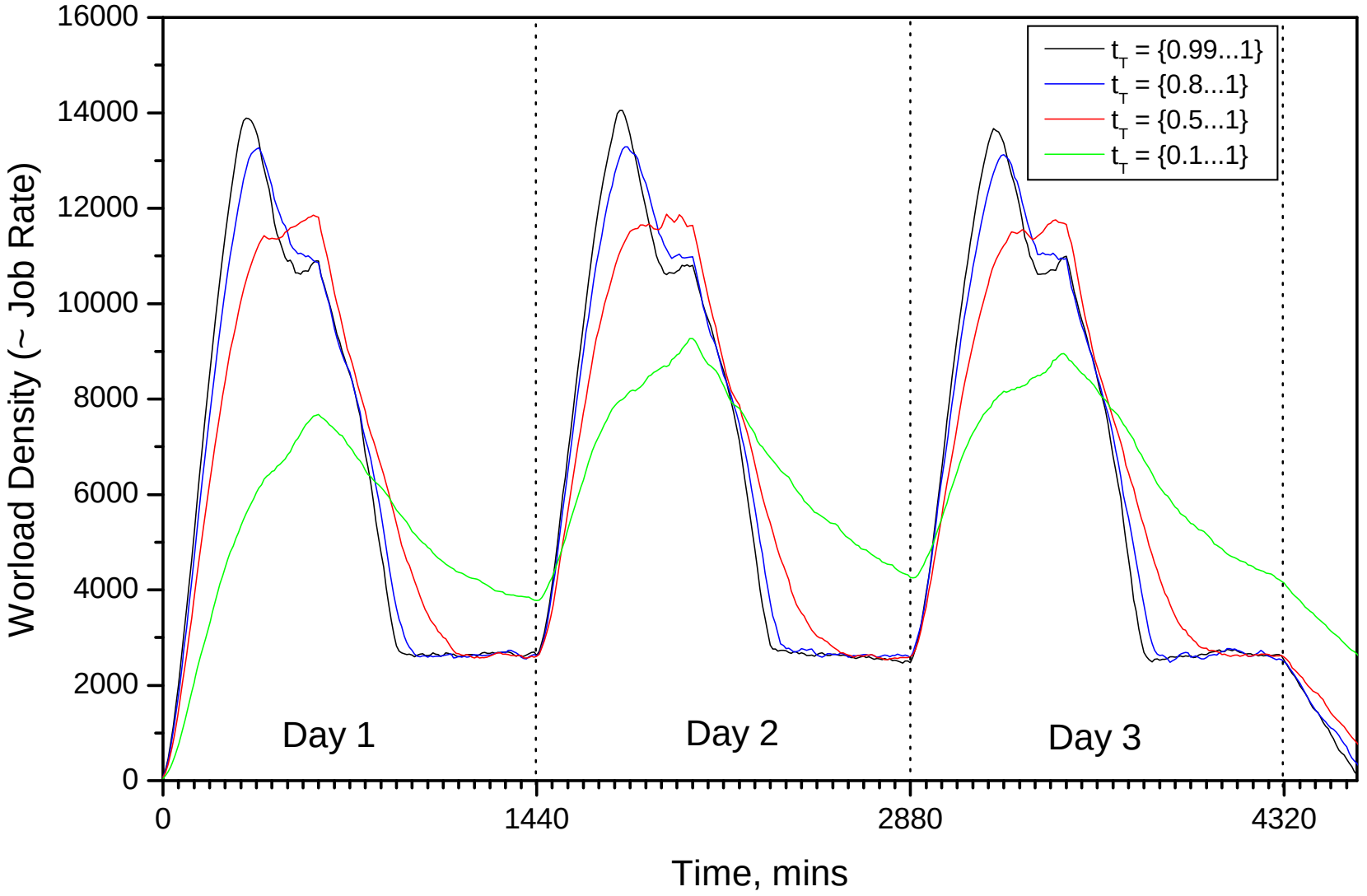


Workload Generator Framework: Example (paper in preparation)

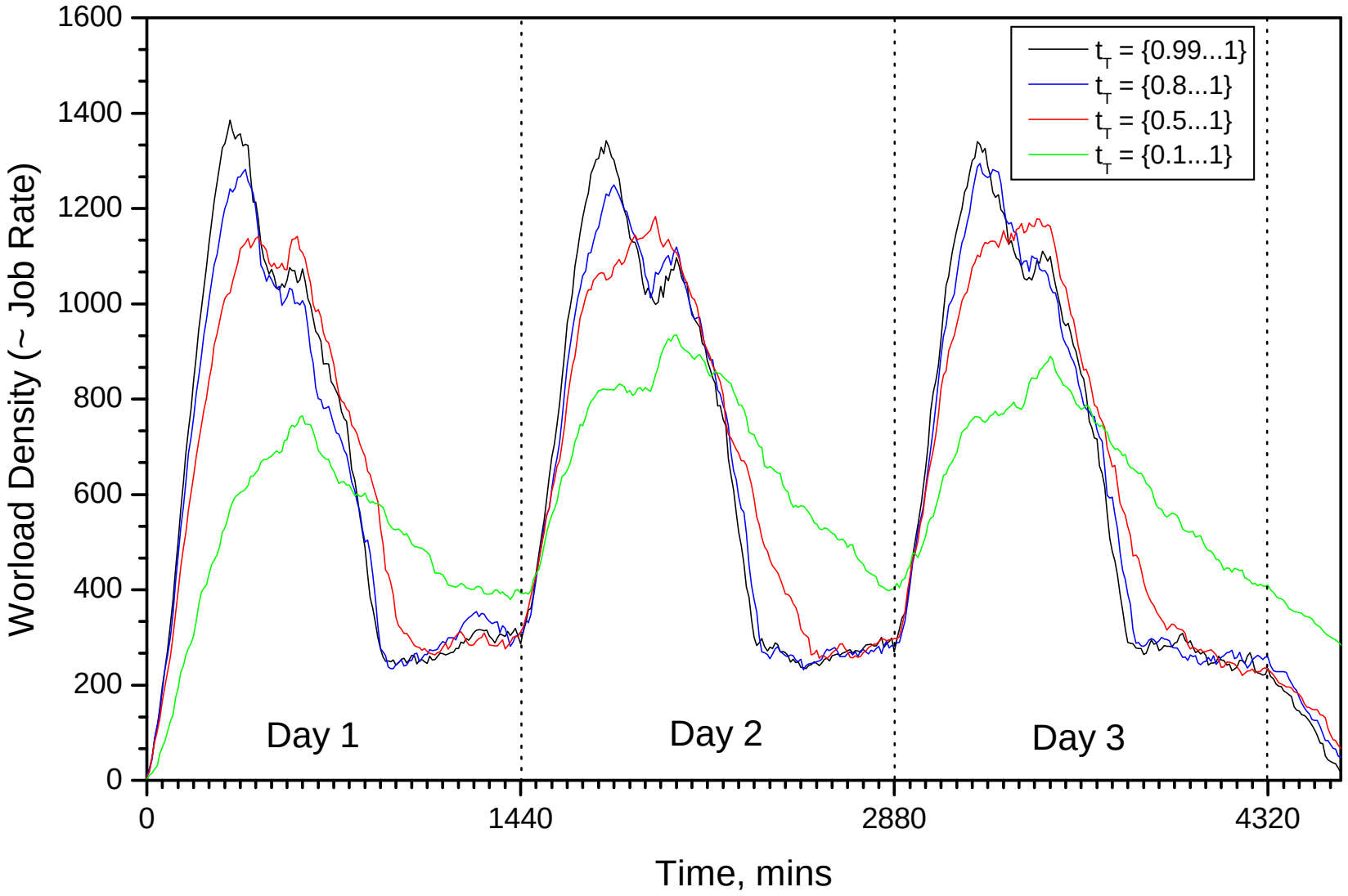


Demo available from:
http://www.gridscheduling.org

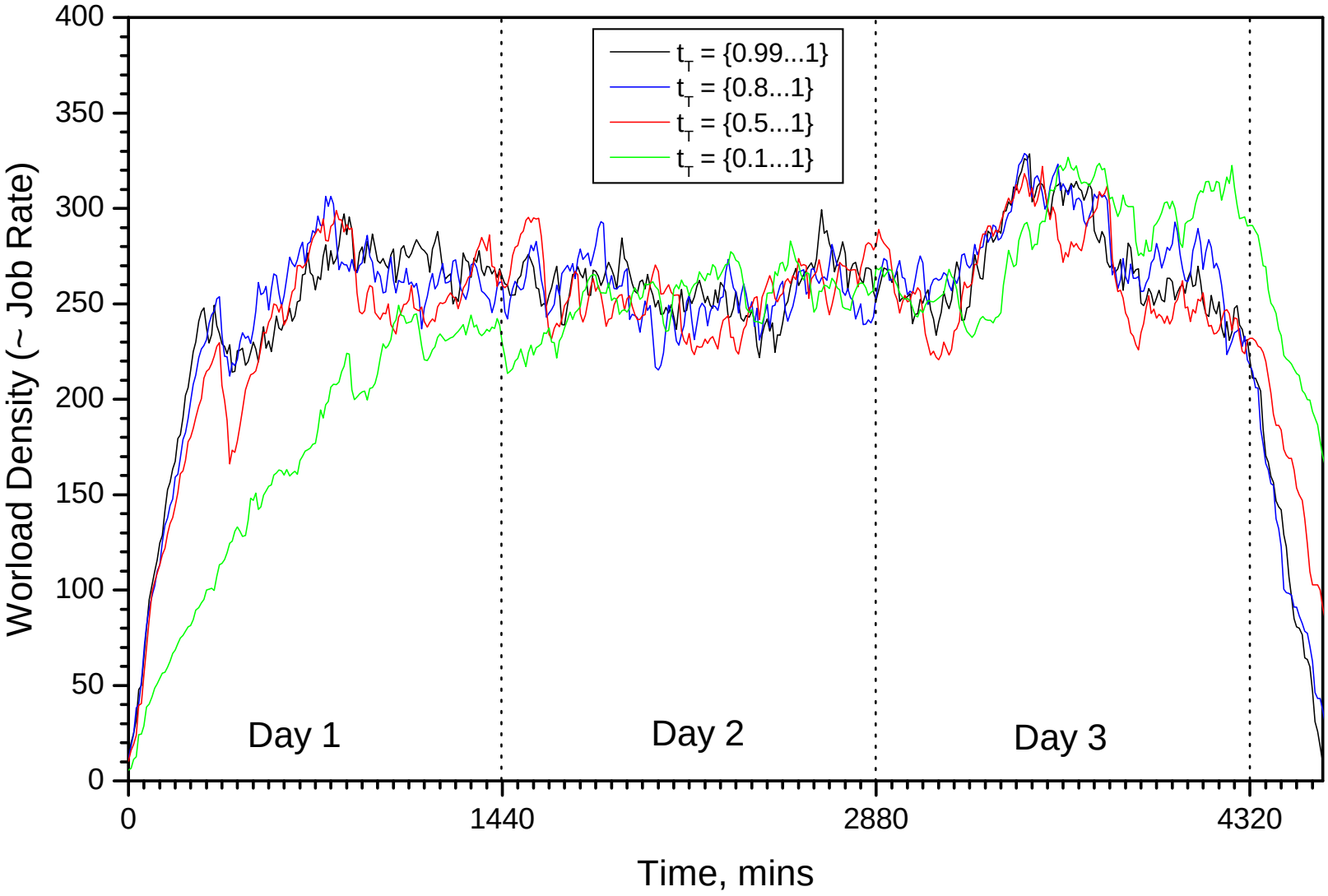
Workload Generator Framework: Output



Workload Generator Framework: Output



Workload Generator Framework: Output



Future Work and Work in Progress

- Dynamic SLAs for DAGs
- Dynamic SLAs for dynamic workflows
- SLA workload models and Generator
- Scheduling Heuristics
- Dynamic SLAs for Market Models

References

Thanks!

In preparation: SLA Workload Generator

Rizos Sakellariou, Viktor Yarmolenko, JOB SCHEDULNG ON THE GRID: TOWARDS SLA-BASED SCHEDULING, in L.Grandinetti (ed.), *High Performance Computing and Grids in Action*, IOS Press, 2008

Viktor Yarmolenko, Rizos Sakellariou, TOWARDS INCREASED EXPRESSIVENESS IN SERVICE LEVEL AGREEMENTS, *Concurrency and Computation: Practice and Experience*, vol.19, 1975-1990, 2007

Viktor Yarmolenko, Rizos Sakellariou, AN EVALUATION OF HEURISTICS FOR SLA BASED PARALLEL JOB SCHEDULING, *Proceedings of the 3rd High Performance Grid Computing Workshop (HPGC)* (in conjunction with IPDPS 2006), Rhodes, Greece (April 2006), IEEE Computer Society Press

Viktor Yarmolenko, Rizos Sakellariou, Djamila Ouelhadj, Jonathan M Garibaldi, SLA BASED JOB SCHEDULING: A CASE STUDY ON POLICIES FOR NEGOTIATION WITH WITH RESOURCES, *Proceedings of the All Hands Meeting AHM'05*, Nottingham, UK (September 2005)

http://www.gridscheduling.org