SLA Aware Systems

SORMA-Project Meeting
University of Karlsruhe
11,12 March 2008

Recent Research Activities

Viktor Yarmolenko

People involved:

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

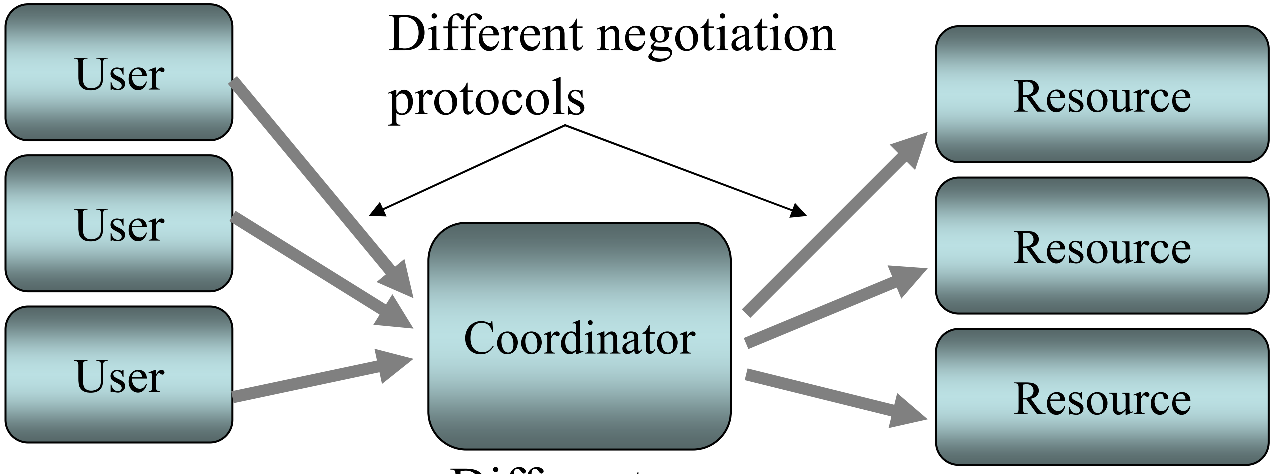
Main directions

- Dynamic Service Level Agreements
- SLA Aware Scheduling
- Synthetic SLA Workloads
- Other work in Progress

Simulation Example

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

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



User

User

User

Different negotiation protocols

Coordinator

Resource

Resource

Resource

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

Different Coordinator Strategies

Viktor Yarmolenko, Rizos Sakellariou, Djamila Ouelhadj, Jonathan M Garibaldi, “SLA Based Job Scheduling: A Case Study on Policies for Negotiation With Resources”, *Proceedings of the All Hands Meeting AHM'05*, Nottingham, UK (September 2005)

All this is in the context of Service Level Agreements.

SLA Representation: Traditional

SLA Example: Traditional Approach

T_S – the earliest time the Job is allowed to start

T_F – the latest time the Job is allowed to finish

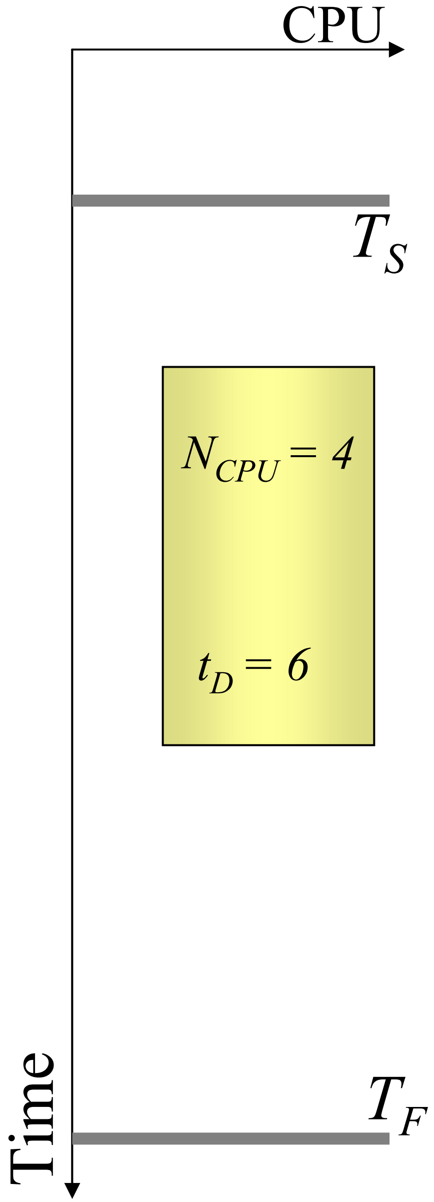
N_{CPU} – number of nodes required for the Job

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

SLA Example: Our Approach

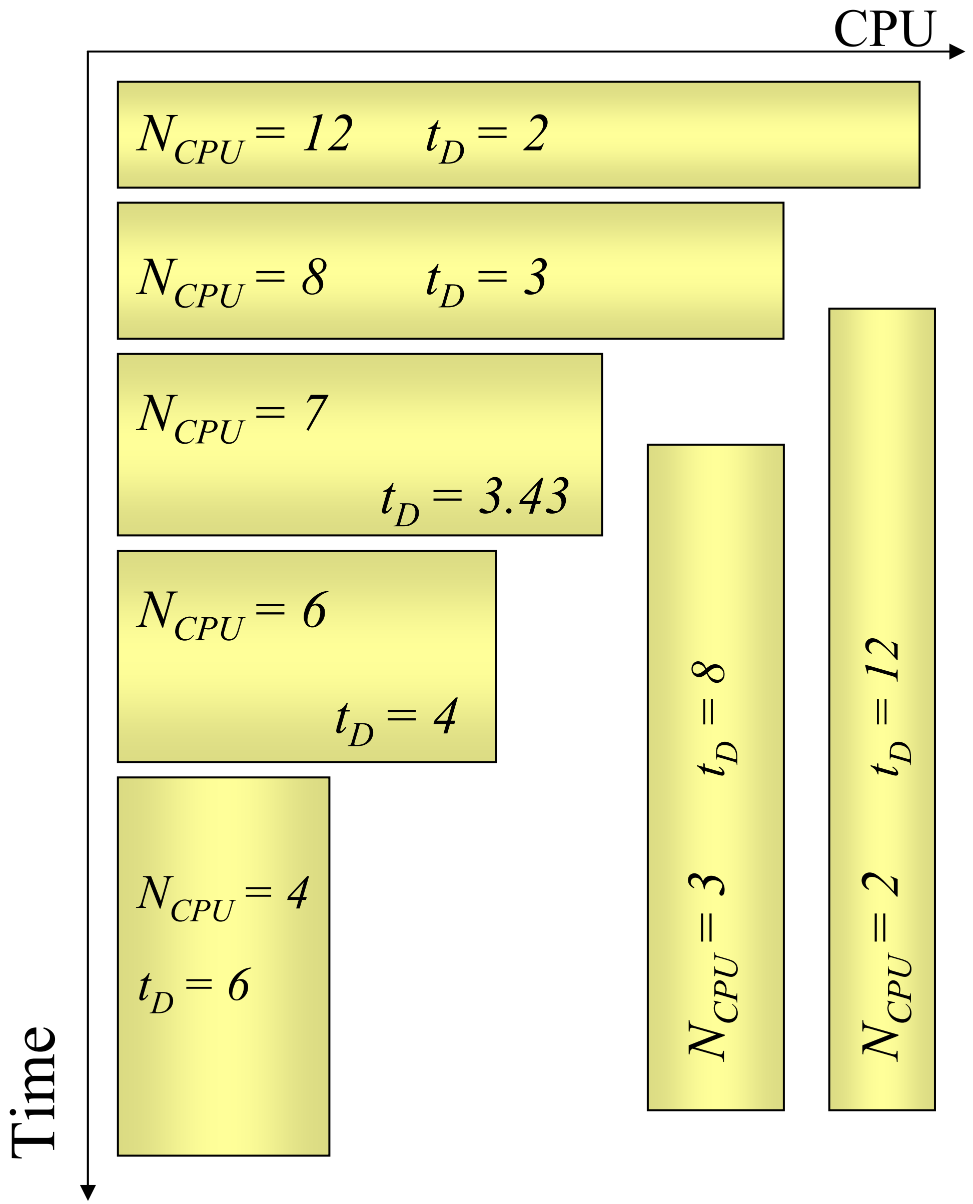
T_S, T_F , … as before but …

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

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

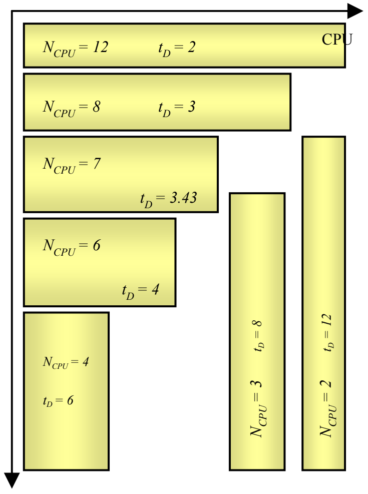
$t_{UP} = 24$, (CPU-hours) duration

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



SLA Representation: Expressive

SLA Example: Same as before, but …



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

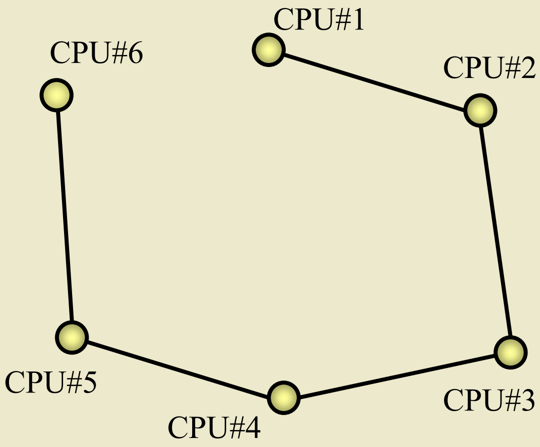
$d(n) = n+(n\text{-}1)+\ldots+2+1$, *a known expression*

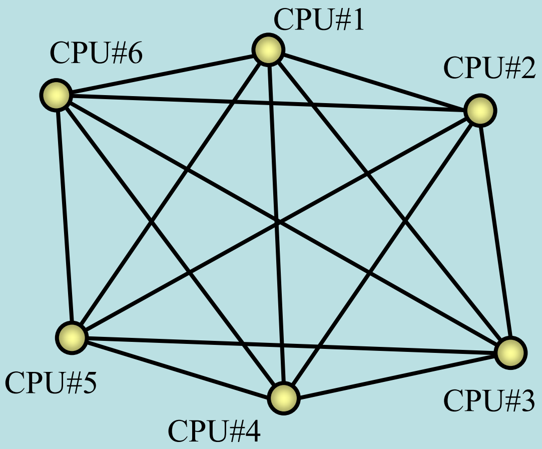
$B_{job} = B_0\, d(N_{CPU} - 1)$, traffic generated by the Job

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

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

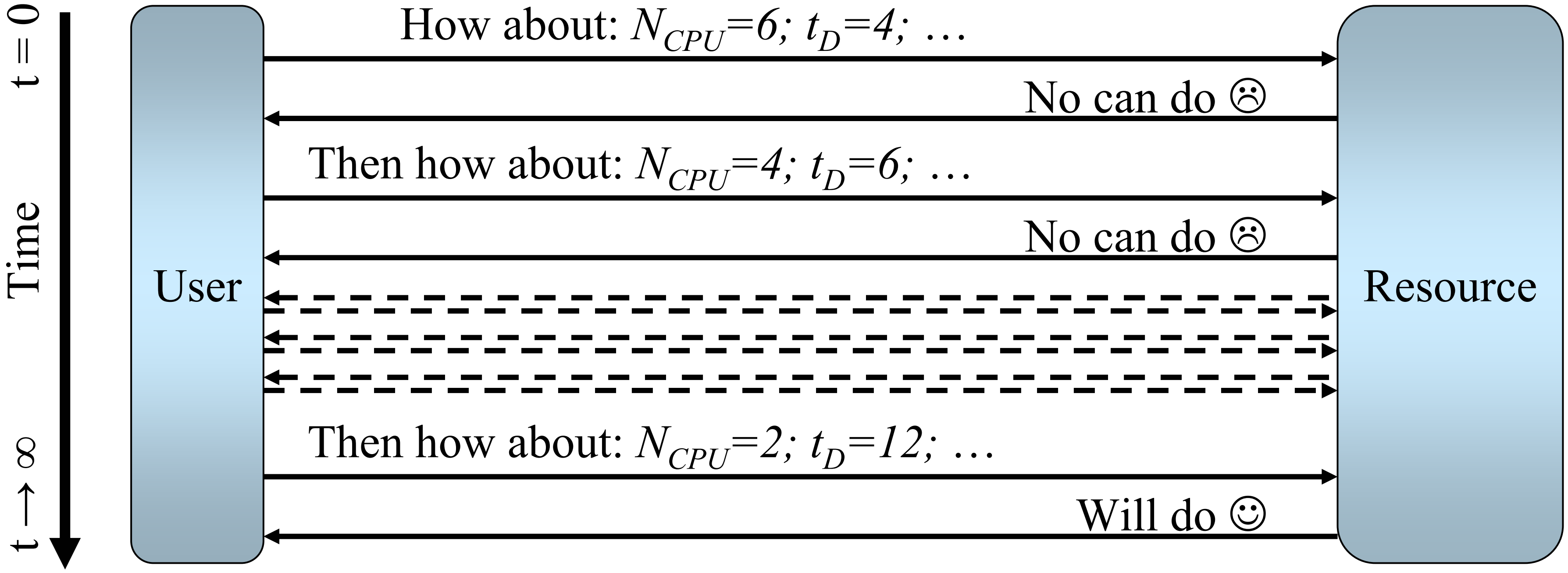
$$B_{job} = B_0\, (N_{CPU} - 1)$$





SLA Representation: Negotiation

Variable CPU Scenario (Traditional vs. Expressive SLA)



t = 0

Time

t → ∞

User

How about: $N_{CPU}=6; t_D=4;$ …

No can do ☹

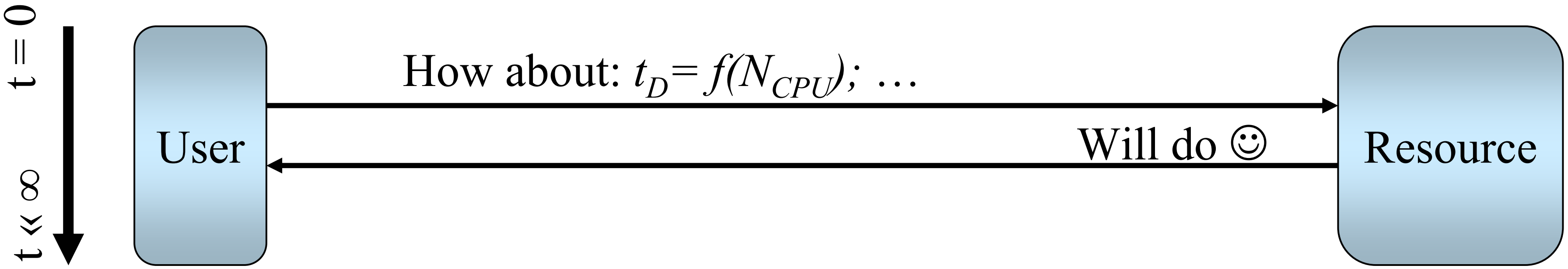
Then how about: $N_{CPU}=4; t_D=6;$ …

No can do ☹

Then how about: $N_{CPU}=2; t_D=12;$ …

Will do ☺

Resource



t = 0

t ≪ ∞

User

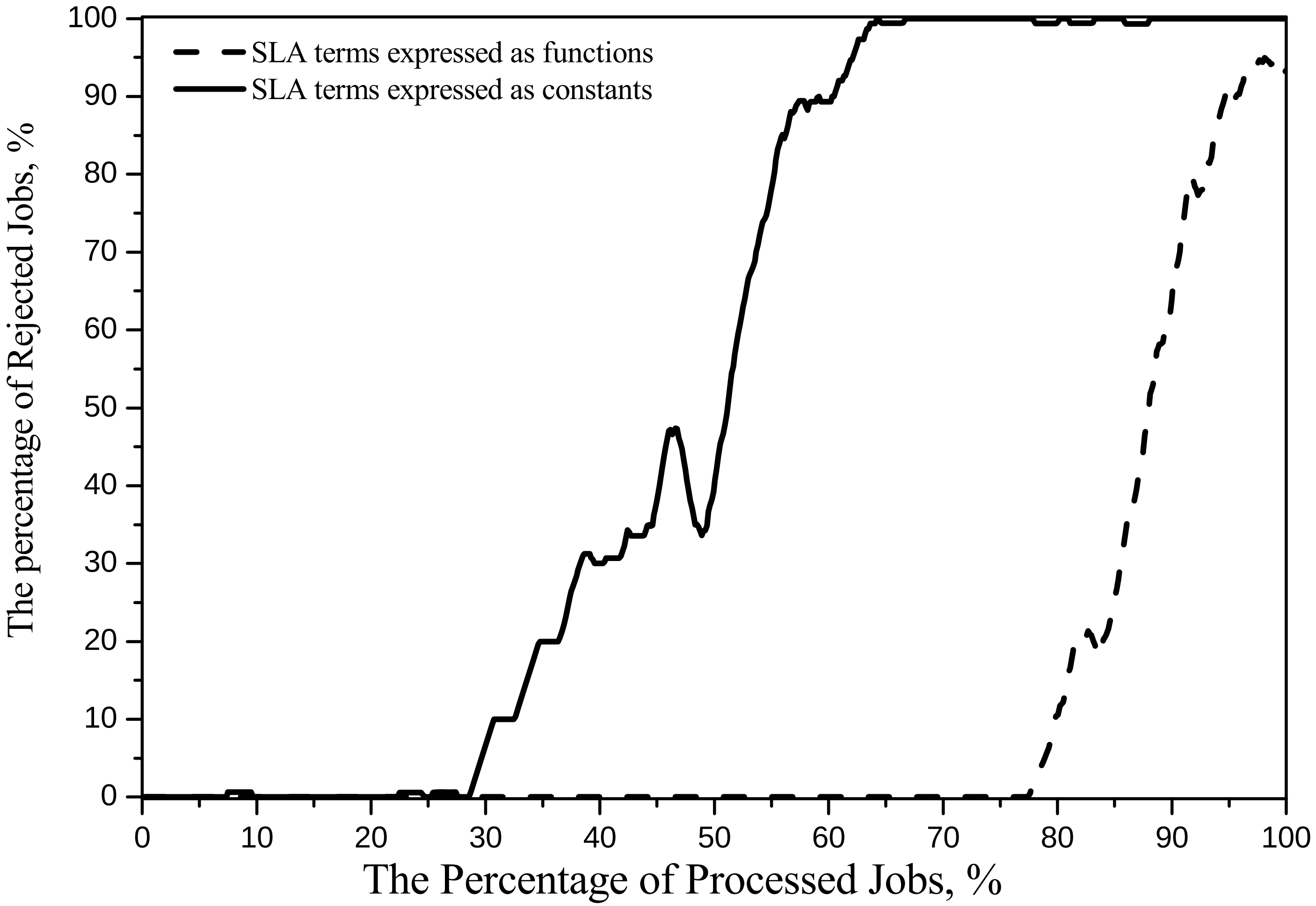
How about: $t_D = f(N_{CPU});$ …

Will do ☺

Resource

SLA Representation: Negotiation

Only Single Negotiation is Allowed



SLA Example: Defining the Price of the Service as Function

t_{curr}

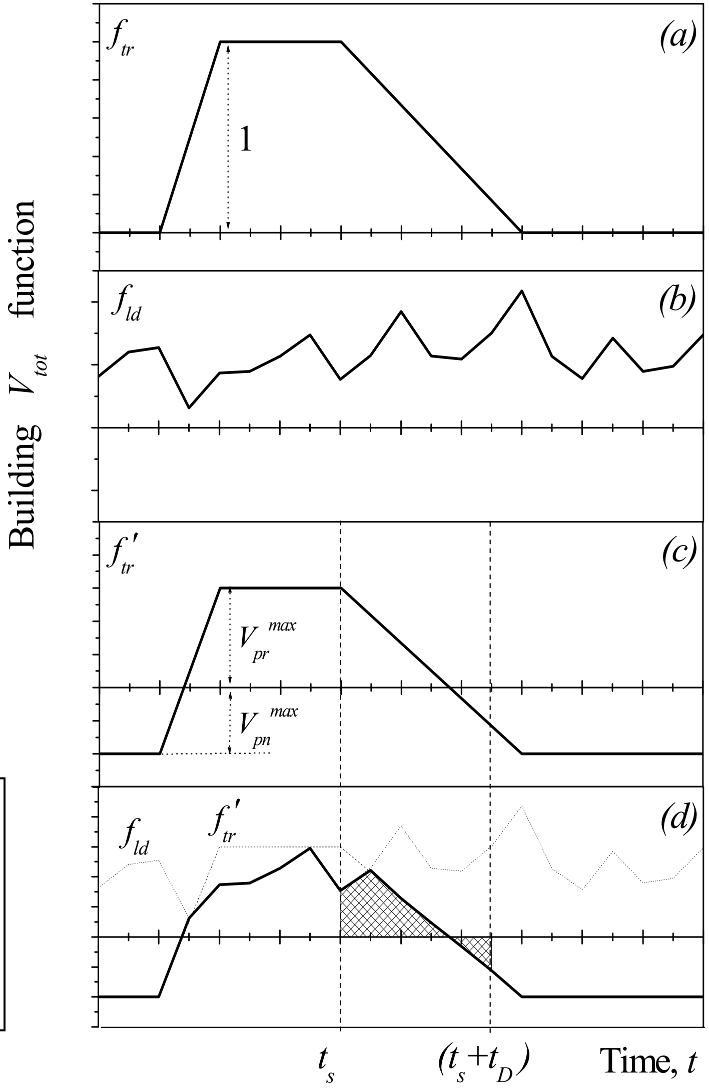
$B_{RES}(t_{curr})$

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

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

Don’t stop here, add more functions!!!

Viktor Yarmolenko, Rizos Sakellariou, “Towards Increased Expressiveness in Service Level Agreements”, *Concurrency and Computation: Practice and Experience*, vol.19, 1975-1990 (2007)



Scheduling Heuristics

The Problem

- When client pays money, client wants guarantees, QoS, promises, etc

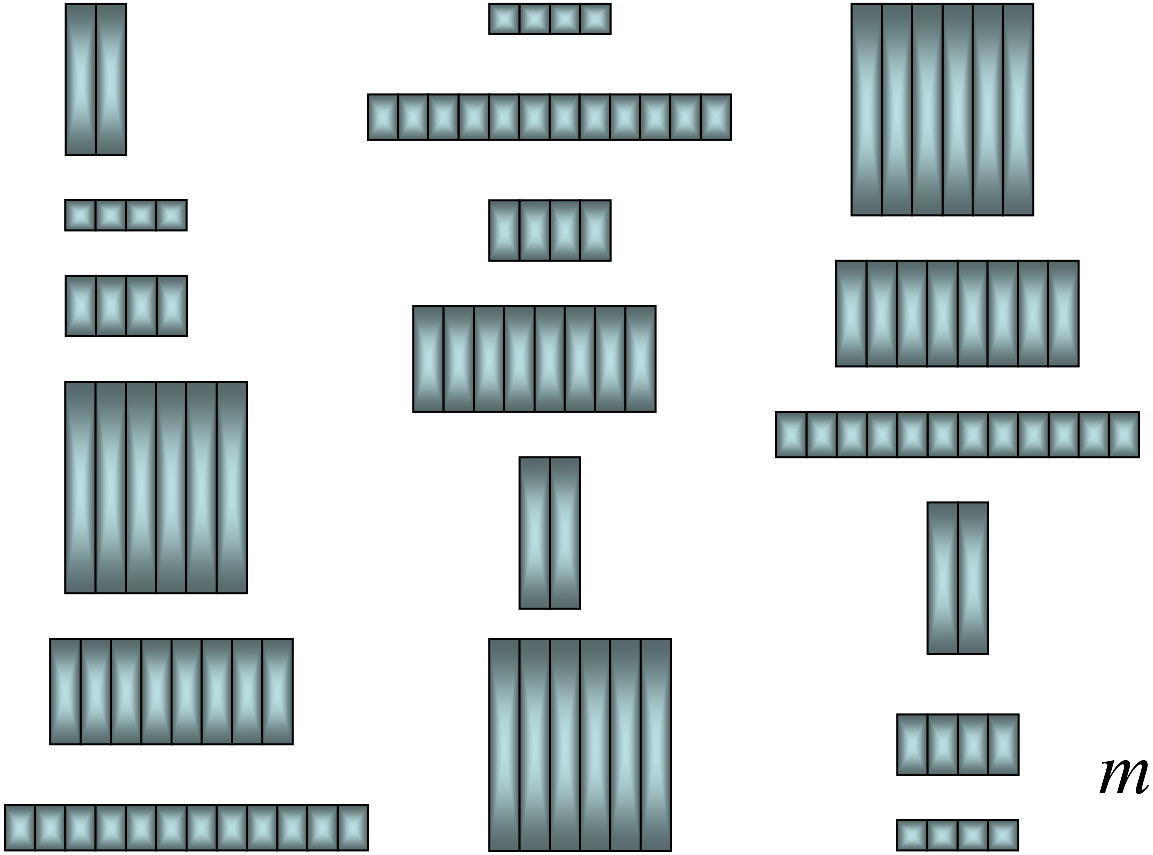
- These can be defined in Service Level Agreements, which preferably are legally binding – contracts.

- Once provider agreed to the terms described in SLA, provider better keep the agreement. How? If failures occur, what to do? Which SLAs to brake? How to schedule more efficiently? How to schedule to generate more income? …

Scheduling Heuristics

Simple and Fast Heuristics: Step 1 - Prioritising Jobs

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



$min(T_S)$

$min(T_F)$

$min(t_L)$

$min(T_S + wt_D)$

$max(T_F + wA)$

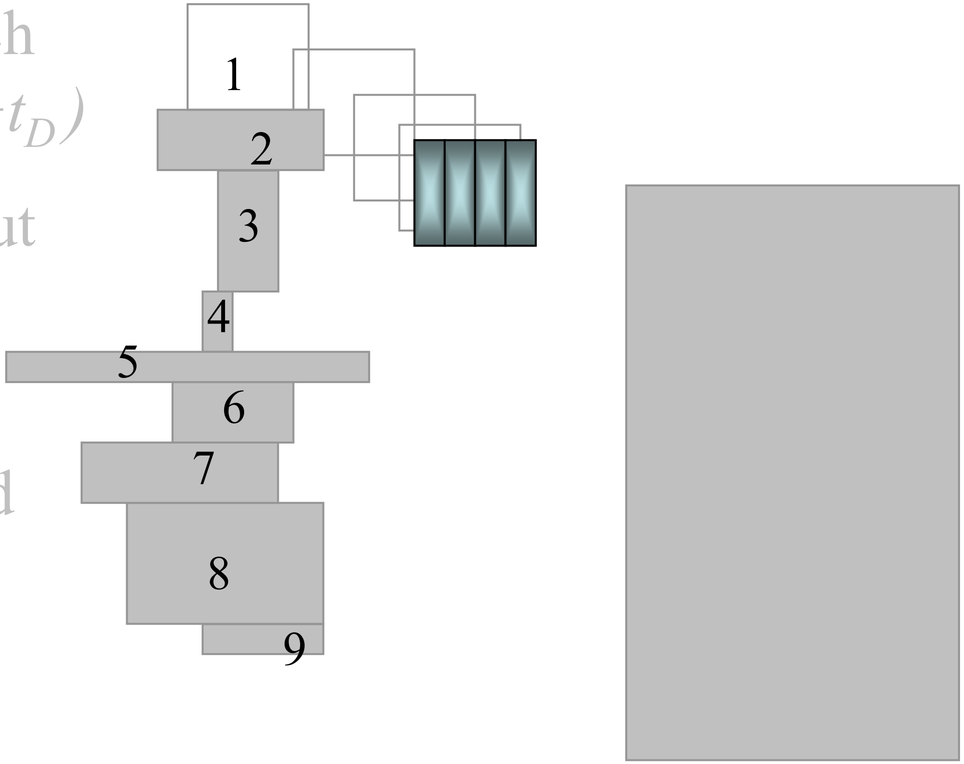
etc.

$min(T_F + w_1A + w_2t_L)$

Scheduling Heuristics

Simple and Fast Heuristics: Step 2 - Allocating Jobs

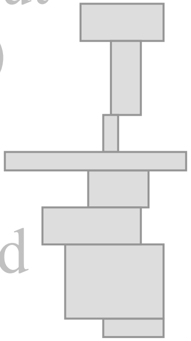
1. Pick up the next job on the list
2. Try to find N_{CPU} nodes which are available from T_S to (T_S+t_D)
3. If unsuccessful, try step 2 but with $(T_S + \Delta t)$ to $(T_S+\Delta t+t_D)$
4. Repeat steps 2 and 3 while $(T_S+\Delta t+t_D) < T_F$ or until find enough free nodes
5. If failed to find N_{CPU}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

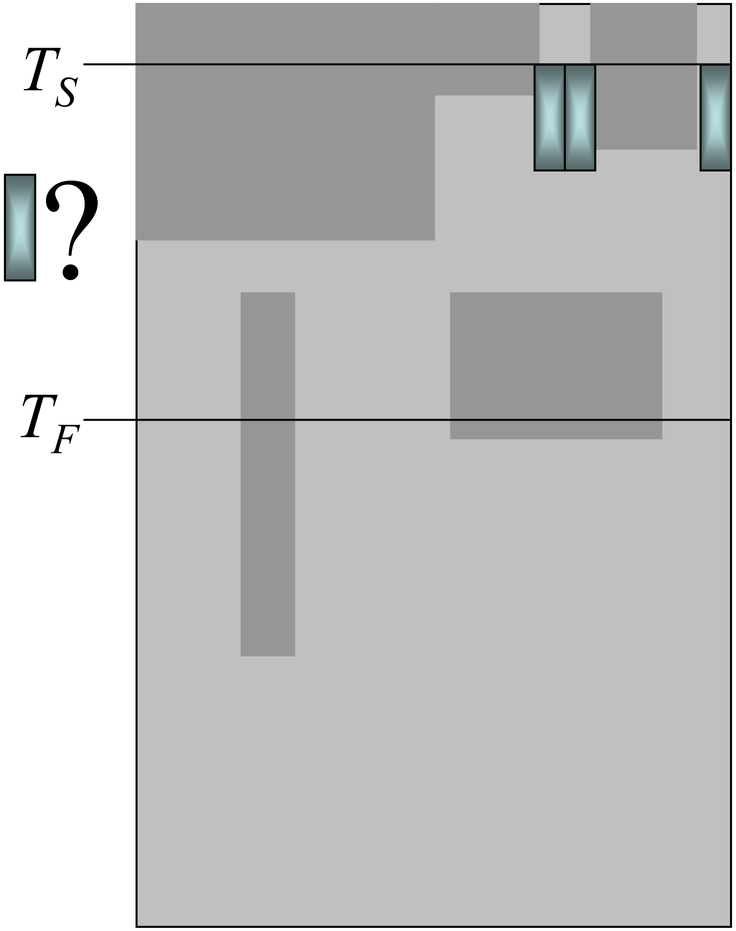


Scheduling Heuristics

Simple and Fast Heuristics: Step 2 - 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.

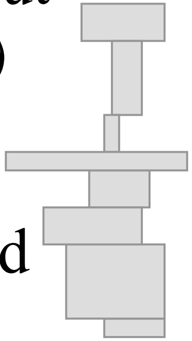


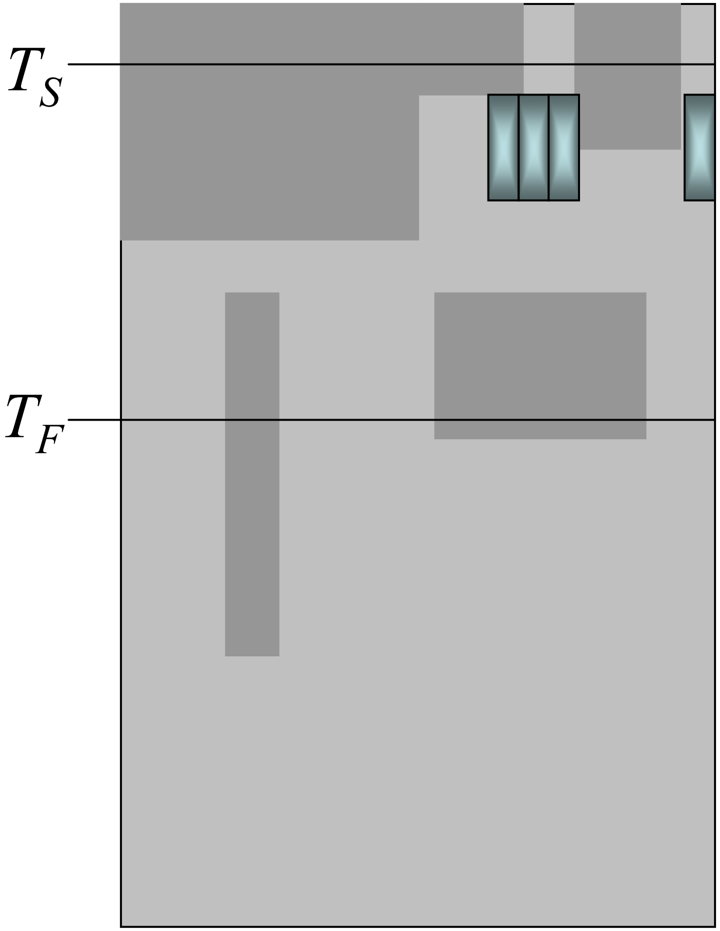


Scheduling Heuristics

Simple and Fast Heuristics: Step 2 - Allocating Jobs

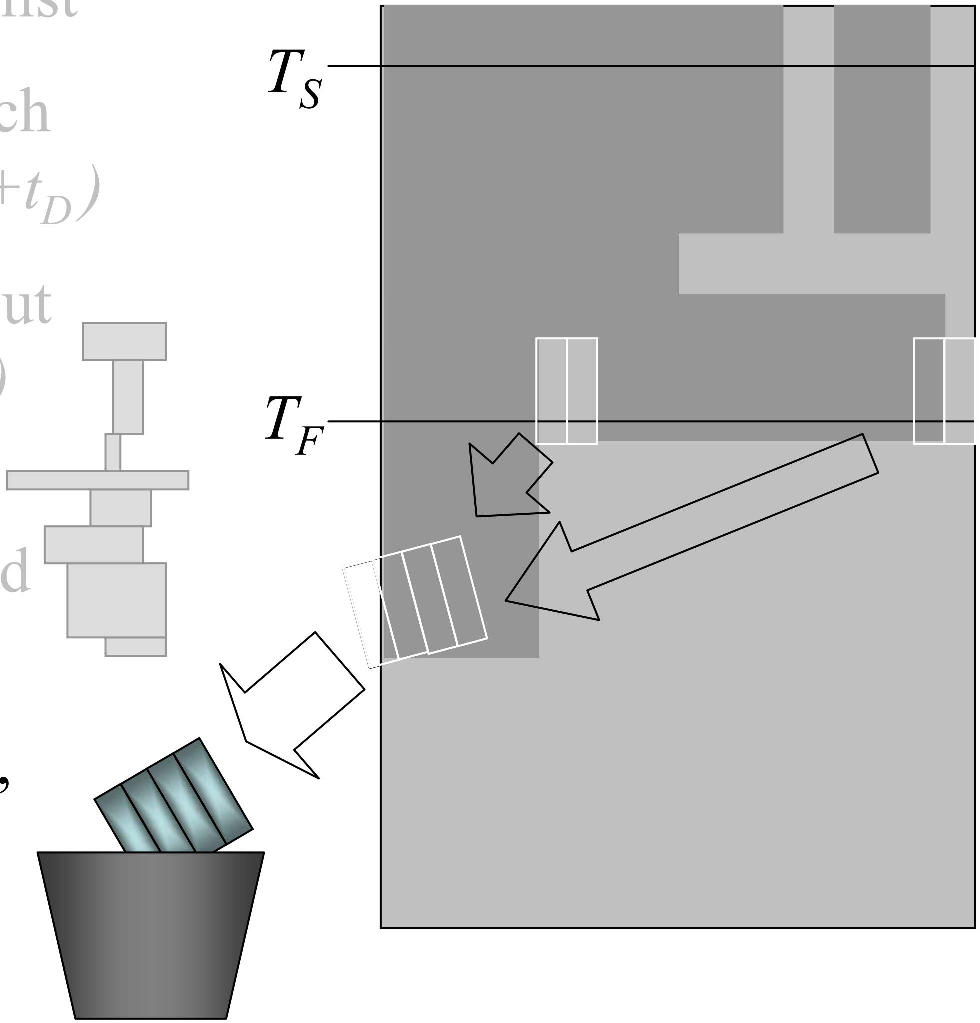
1. Pick up the next job on the list
2. Try to find N_{CPU} nodes which are available from T_S to (T_S+t_D)
3. If unsuccessful, try step 2 but with $(T_S + \Delta t)$ to $(T_S+\Delta t+t_D)$
4. Repeat steps 2 and 3 while $(T_S+\Delta t+t_D) < T_F$ or until find enough free nodes
5. If failed to find N_{CPU}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





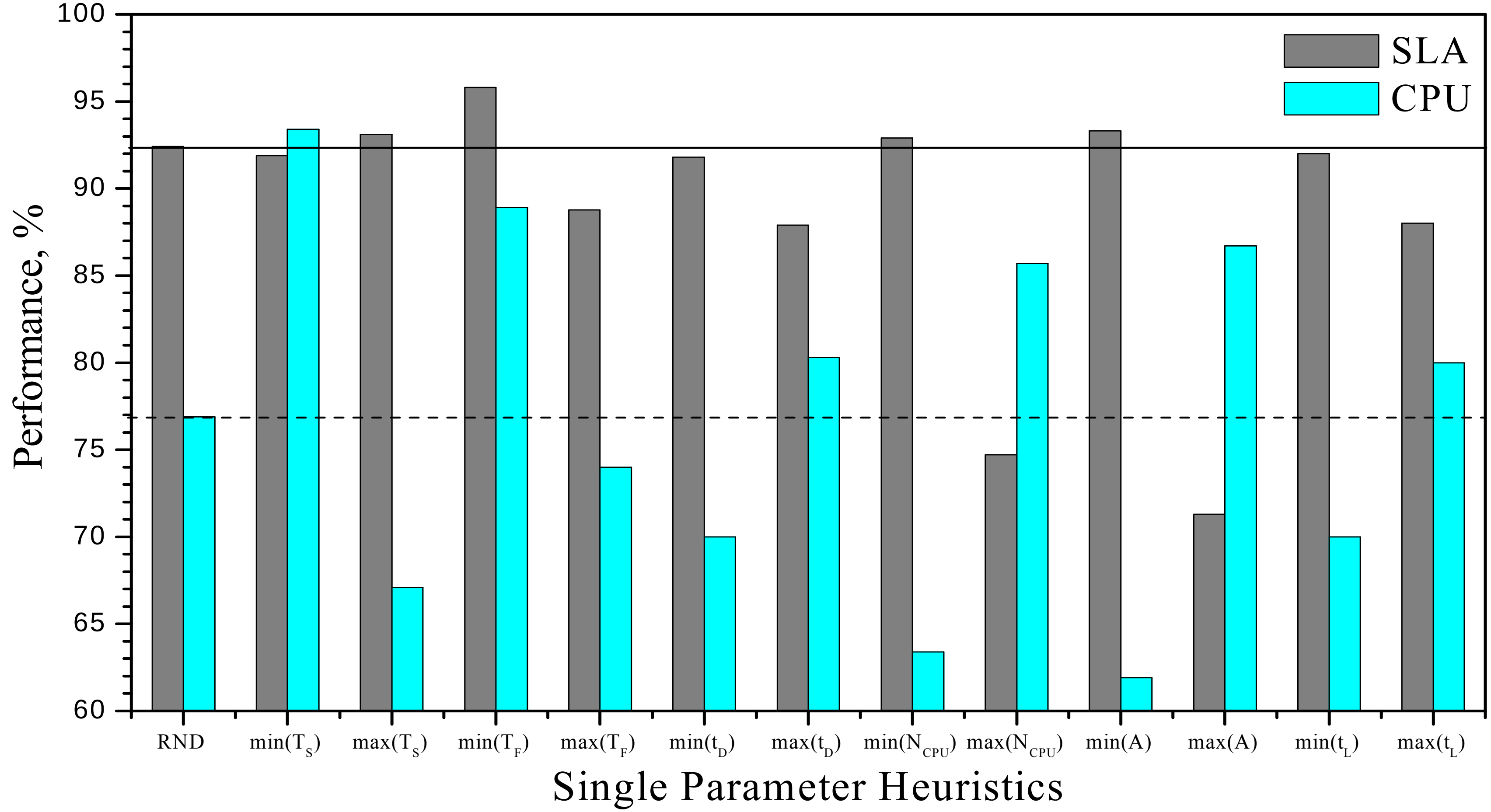
Simple and Fast Heuristics: Step 2 - Allocating Jobs

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.

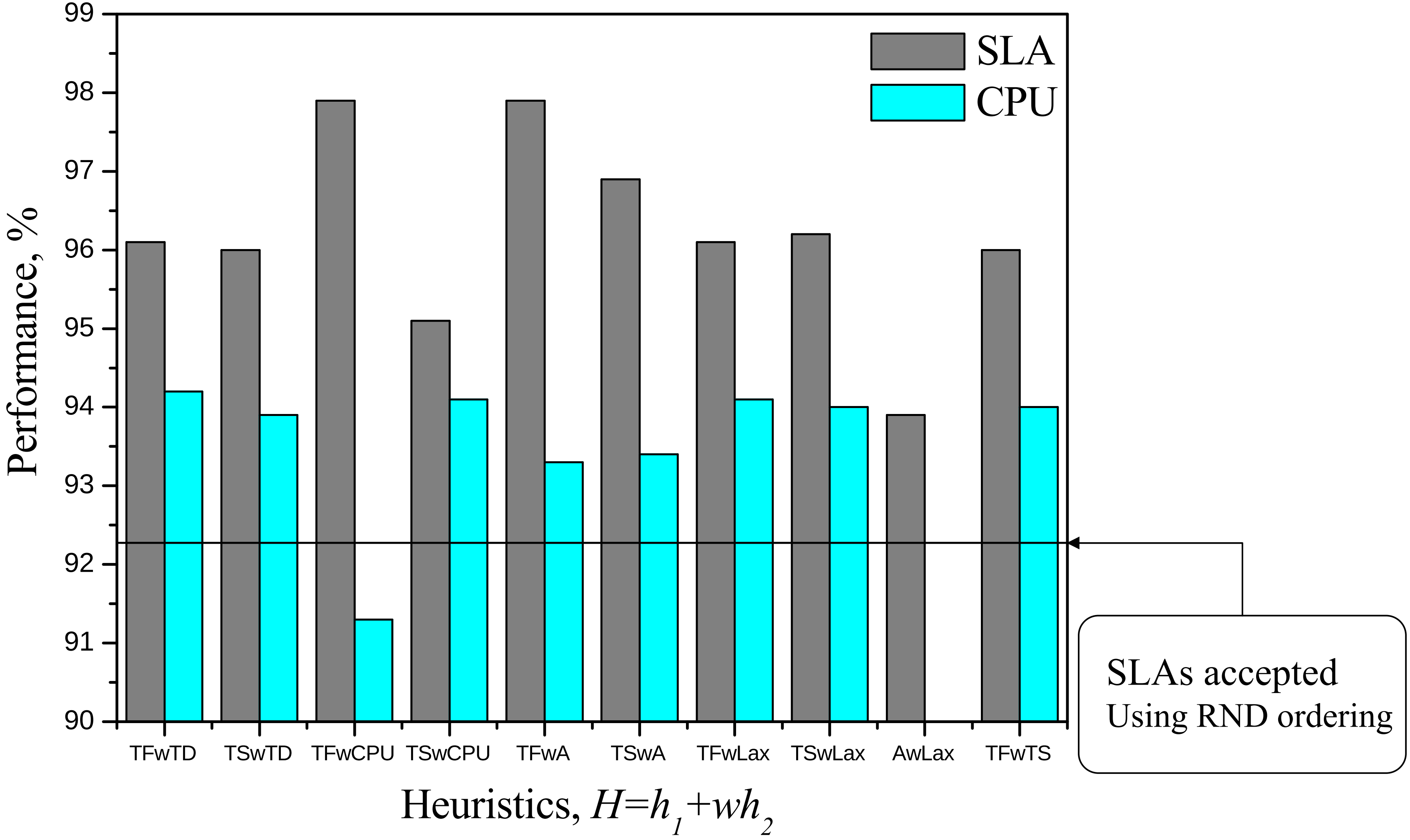


Results: Single Parameter Ordering

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

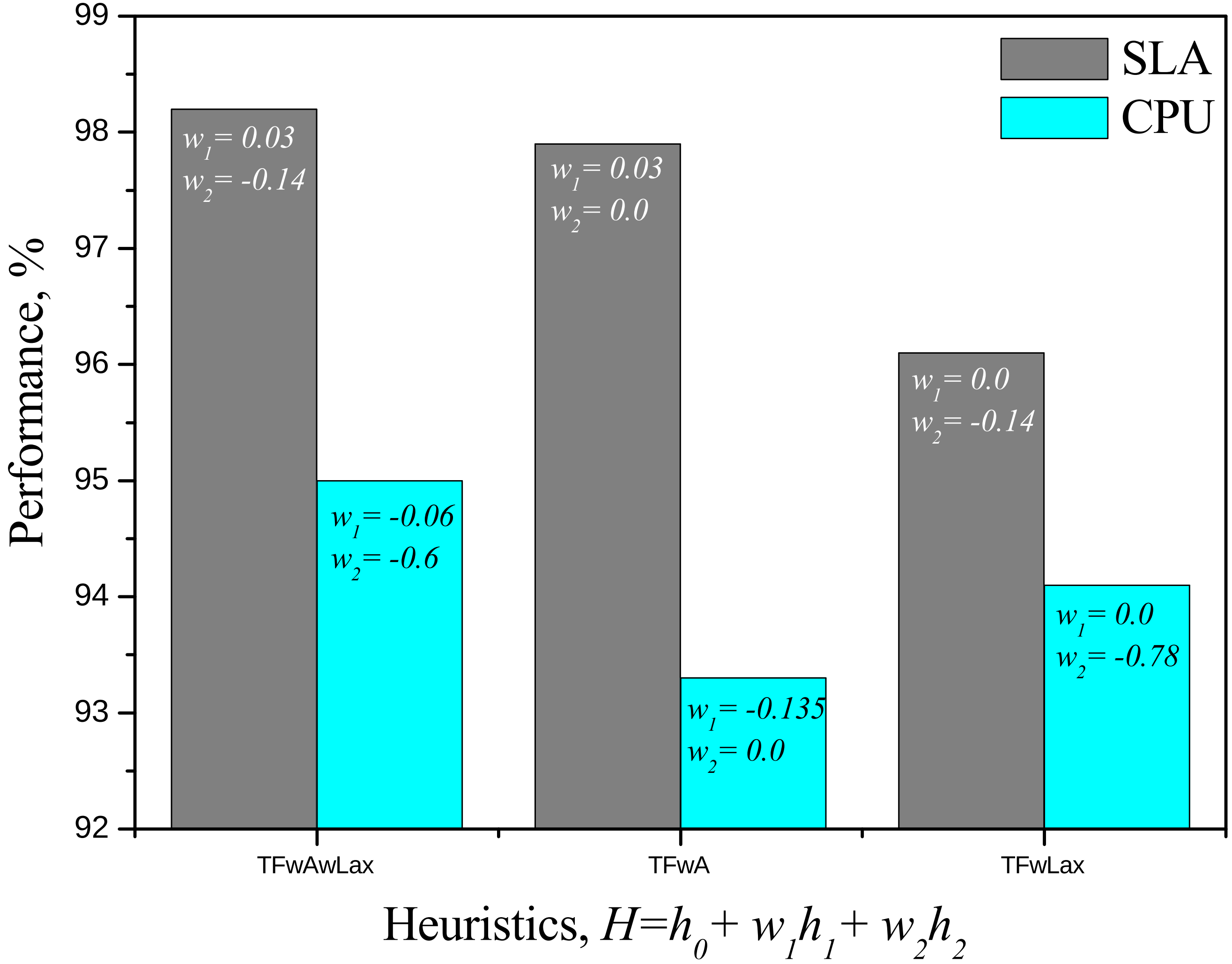


Results: Two Parameter Ordering



SLAs accepted
Using RND ordering

Results: Three Parameter Ordering



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

Scheduling Heuristics

Results: Interesting Observations

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

Higher level overview is here:

Rizos Sakellariou, Viktor Yarmolenko, "Job Scheduling on the Grid: Towards SLA-Based Scheduling", in L. Grandinetti (ed.), *High Performance Computing and Grids in Action"*, IOS Press, 2008

Workload: The Challenges

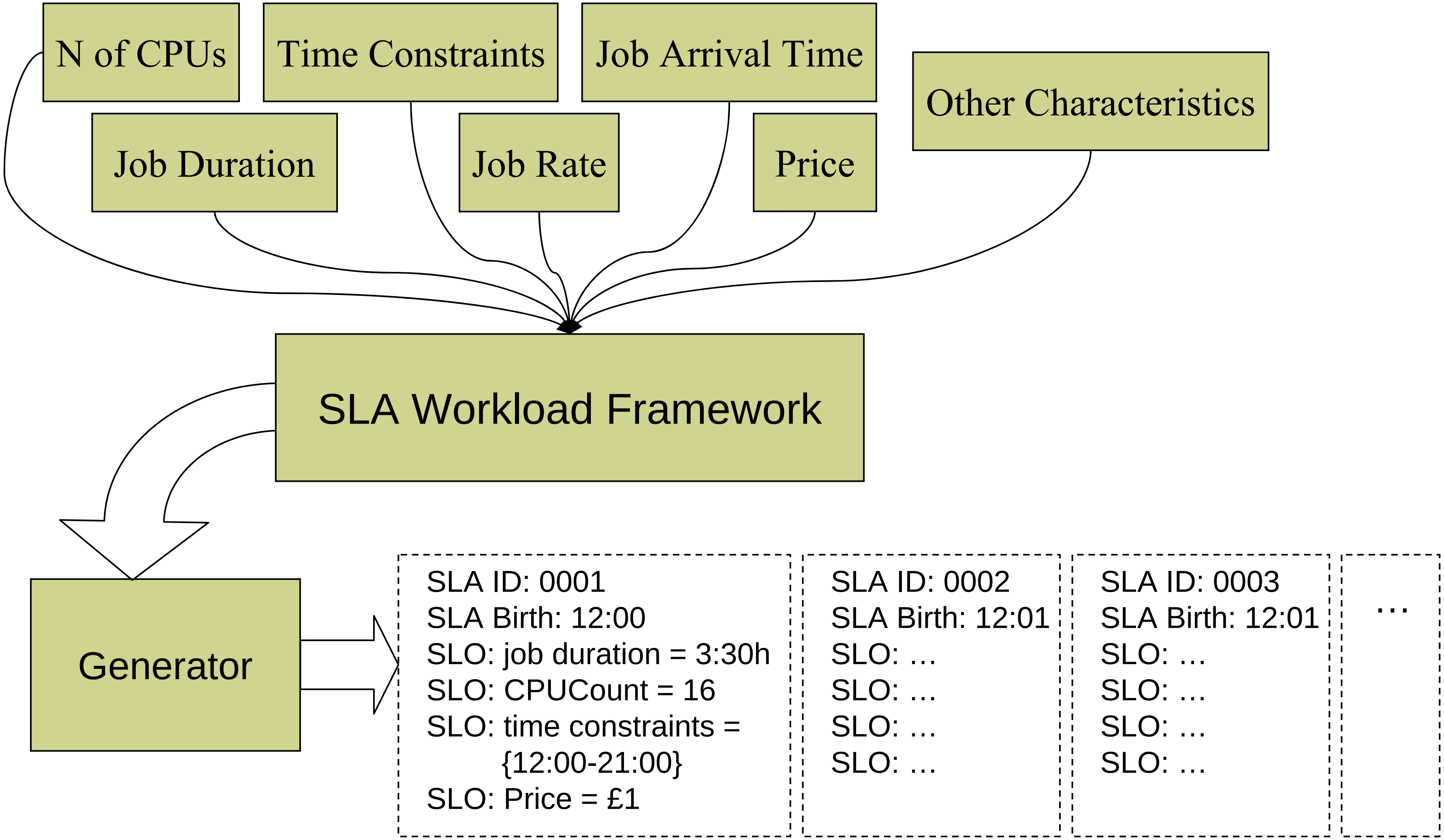
- The performance of the system is heavily dependant on the type of the workload

- There is no or very little data available about SLA workloads

- We need a tool that would allow to assemble a synthetic workload based on real logs, existing models, various other rules, and their combinations, easily integrated in a workload generator tool which would provide a workload stream with desired characteristics and behaviour.

- Novel workloads require new metrics or ways that characterise them (job rate, or throughput may not longer be an adequate description of a workload)

Workload Generator Framework: The Concept (paper in preparation)



N of CPUs

Time Constraints

Job Arrival Time

Other Characteristics

Job Duration

Job Rate

Price

SLA Workload Framework

Generator

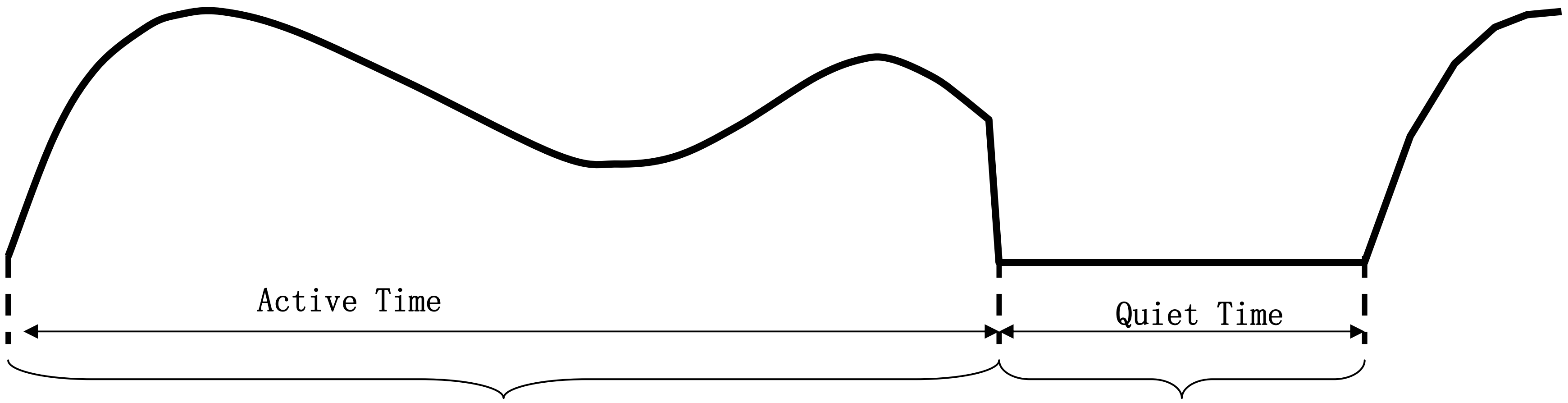
SLA ID: 0001
SLA Birth: 12:00
SLO: job duration = 3:30h
SLO: CPUCount = 16
SLO: time constraints =
{12:00-21:00}
SLO: Price = £1

SLA ID: 0002
SLA Birth: 12:01
SLO: …
SLO: …
SLO: …
SLO: …

SLA ID: 0003
SLA Birth: 12:01
SLO: …
SLO: …
SLO: …
SLO: …

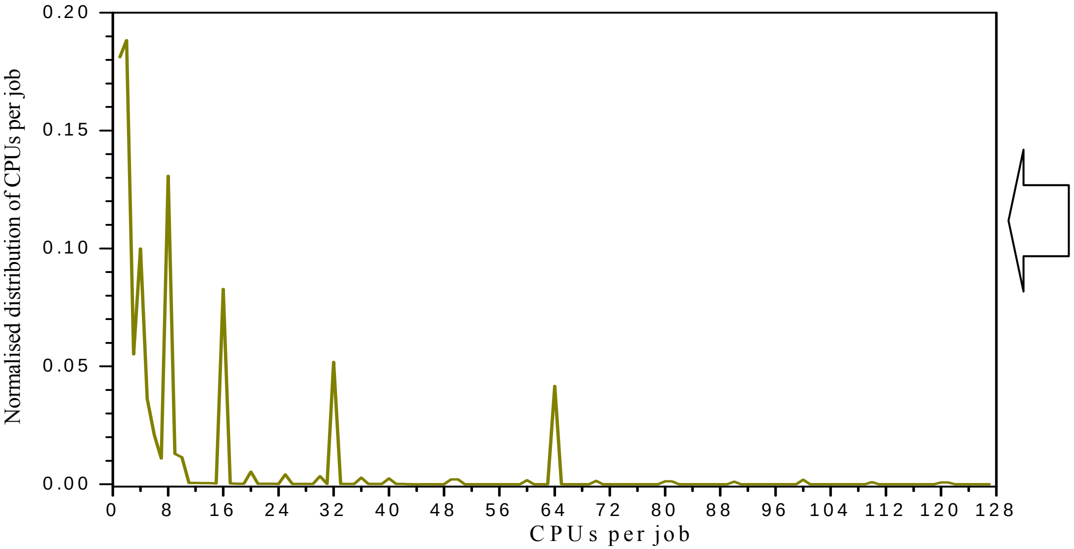
…

Modelling Variation in User Demand: Related Work



Calzarossa & Serrazi Model (1985) models the daily cycle of job rate, 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



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:

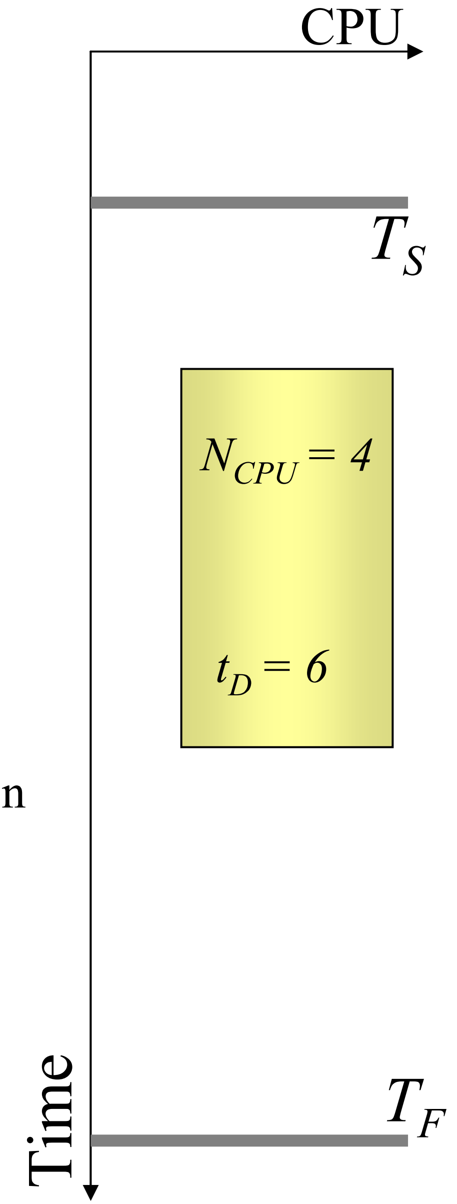
$$t_T = \frac{t_D}{T_F - T_S}$$

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

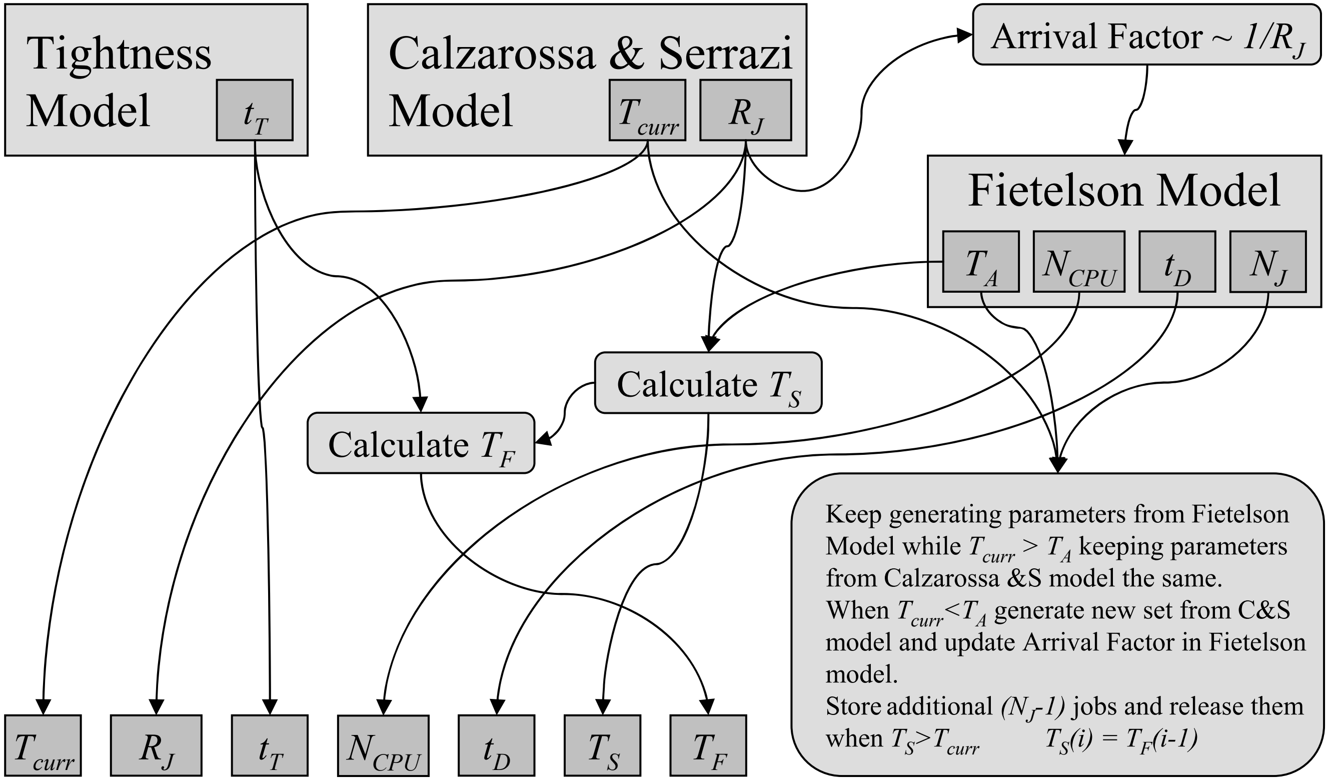
Current state of grid is such that this function is a discrete distribution with non-zero values for:

$t_T = 0$ – no constraints, deadlines

$t_T = 1$ – Advance Reservation



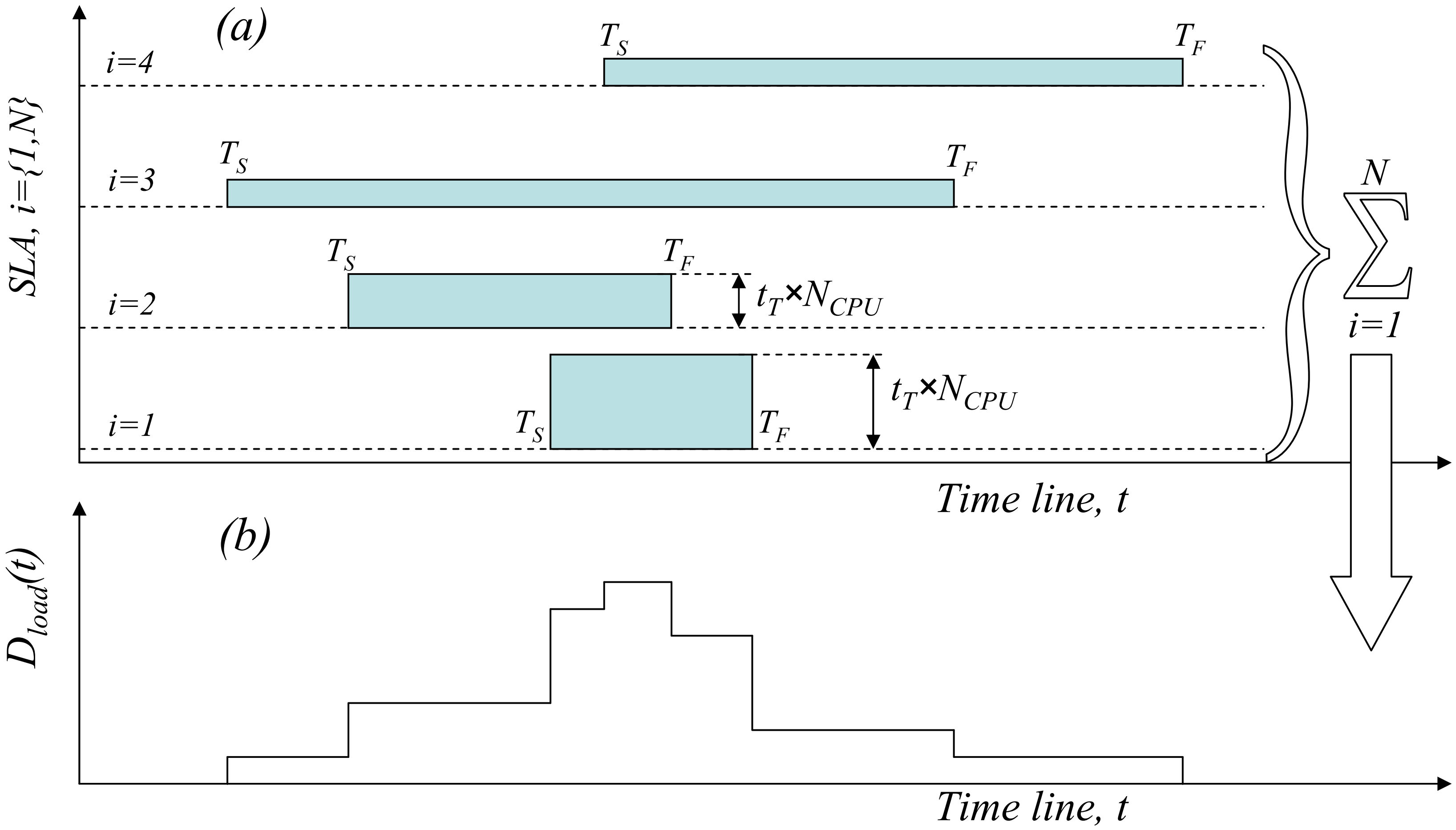
Workload Generator Framework: Example (paper in preparation)



Keep generating parameters from Fietelson Model while $T_{curr} > T_A$ keeping parameters from Calzarossa &S model the same.
When $T_{curr}<T_A$ generate new set from C&S model and update Arrival Factor in Fietelson model.
Store additional (N_J-1) jobs and release them when $T_S>T_{curr}$ $T_S(i) = T_F(i-1)$

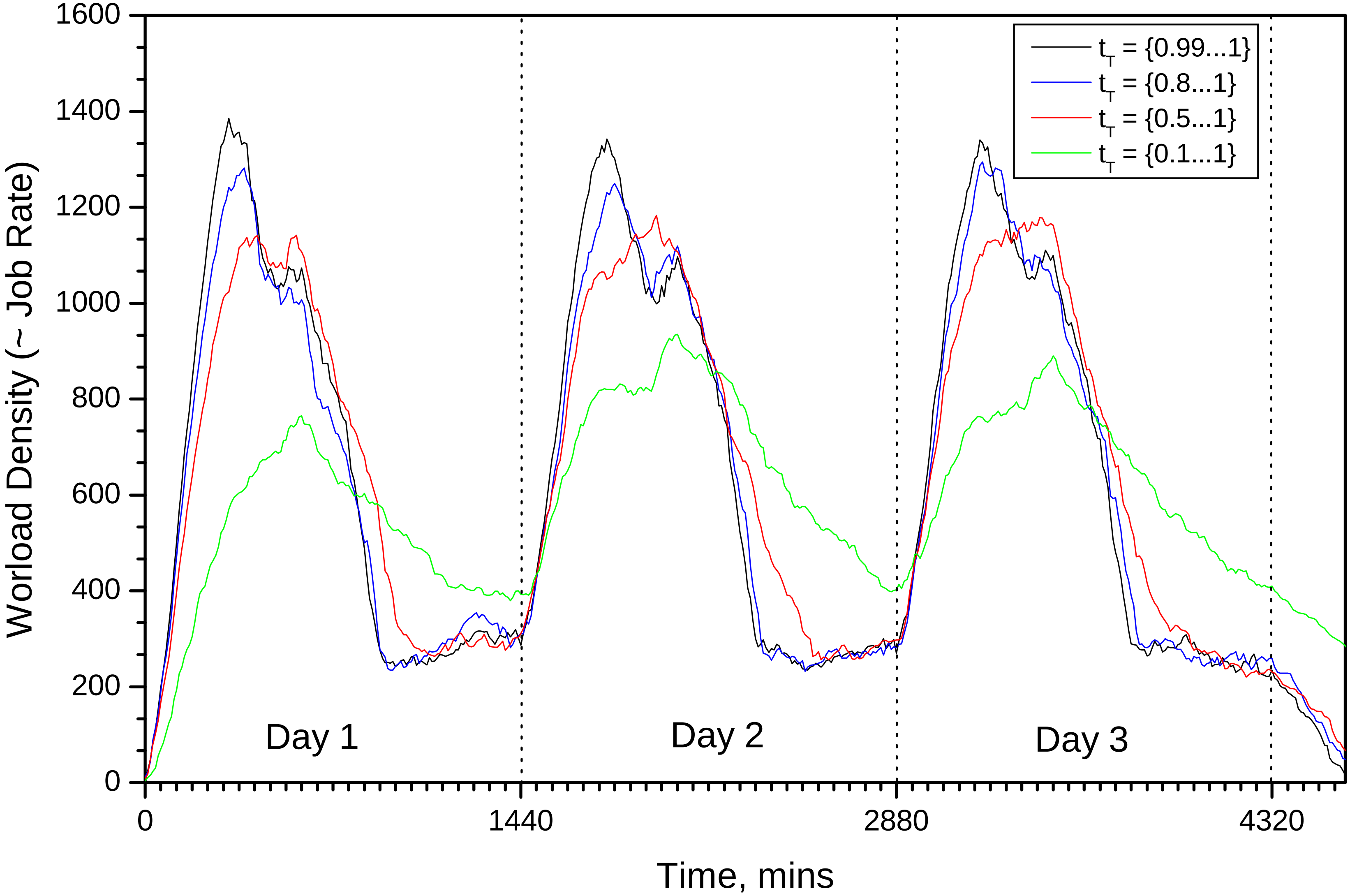
WebApp Demo available from http://www.gridscheduling.org

Workload Generator Framework: What Metrics (paper in preparation)



Example of a new metric: Load Density of the SLA workload

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
- SLAs with notion of trust

Publications

Thanks!

In preparation: Viktor Yarmolenko, Rizos Sakellariou, “A Framework for SLA Workload Generation”

In preparation: Viktor Yarmolenko, Rizos Sakellariou, “An SLA Workload Model for Time Constraints”

Rizos Sakellariou, Viktor Yarmolenko, “Job Scheduling 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 Resources”, *Proceedings of the All Hands Meeting AHM'05*, Nottingham, UK (September 2005)

Keep checking *http://www.gridscheduling.org*