Resilient and energy-aware algorithms

Anne Benoit

Ecole Normale Supérieure de Lyon, France
Institut Universitaire de France

Anne.Benoit@ens-lyon.fr
http://graal.ens-lyon.fr/~abenoit/

IPDPS 2014 PhD Forum

Exascale platforms

- Hierarchical
  - 10^5 or 10^6 nodes
  - Each node equipped with 10^4 or 10^3 cores

- Failure-prone

MTBF – one node	1 year	10 years	120 years
MTBF – platform of 10^6 nodes	30sec	5mn	1h

More nodes $\Rightarrow$ Shorter MTBF (Mean Time Between Failures)

- Energy efficiency

  Thermal power close to the one of a nuclear reactor!

  A critical issue to address if we want to achieve Exascale.

Exascale platforms

- Hierarchical
  - 10^5 or 10^6 nodes
  - ch node equipped with 10^4 or 10^3 cores
- Failure-prone

MTBF – one node	ar	10 years	120 years
MTBF – platform of 10^6 n es	30s	5mn	1h

More nodes ⇒ Sh er MTBF (Mean Tim Between Failures)

- Energy eff ency

  Therm po or!

  A c cal is e Ex cale.

Exascale
≠ Petascale ×1000

Outline

Even for today's platforms (courtesy F. Cappello)

Also an issue at Petascale

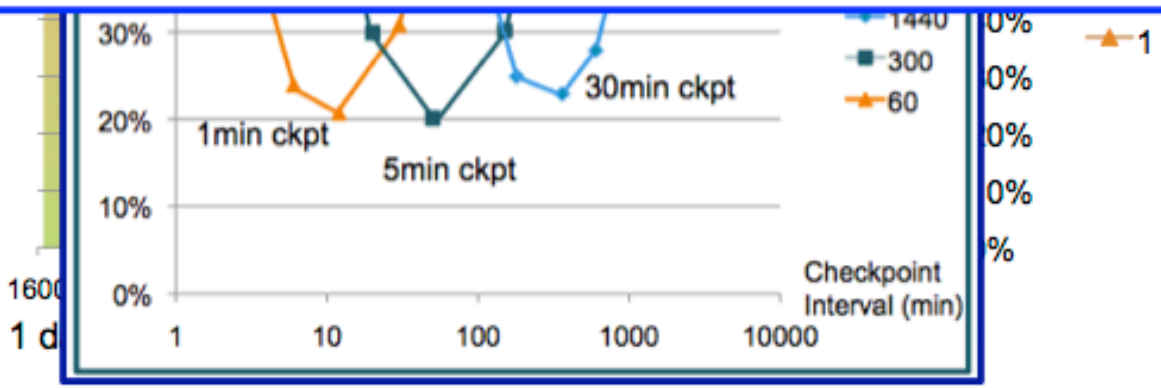
Joint Laboratory for Petascale Computation · INRIA · NCSA

Fault tolerance becomes critical at Petascale (MTTI <= 1day)
Poor fault tolerance design may lead to huge overhead

Overhead of checkpoint/restart

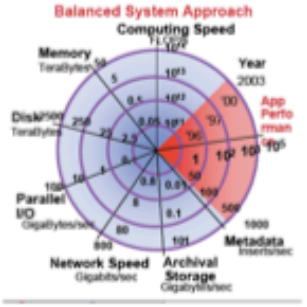
Cost of non optimal checkpoint intervals:

Today, 20% or more of the computing capacity in a large high-performance computing system is wasted due to failures and recoveries.
Dr. E.N. (Mootaz) Elnozahyet al. *System Resilience at Extreme Scale*, DARPA

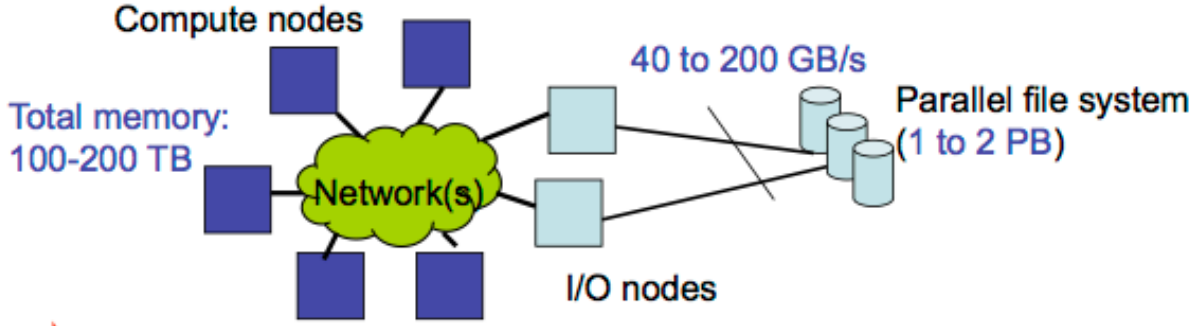


Even for today's platforms (courtesy F. Cappello)

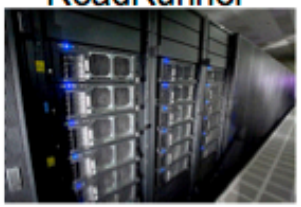
Classic approach for FT: Checkpoint-Restart



Typical "Balanced Architecture" for PetaScale Computers



RoadRunner



⇨ Without optimization, Checkpoint-Restart needs about 1h! (~30 minutes each)

Systems	Perf.	Ckpt time	Source
RoadRunner	1PF	~20 min.	Panasas
LLNL BG/L	500 TF	>20 min.	LLNL
LLNL Zeus	11TF	26 min.	LLNL
YYY BG/P	100 TF	~30 min.	YYY

LLNL BG/L

Scenario for 2015

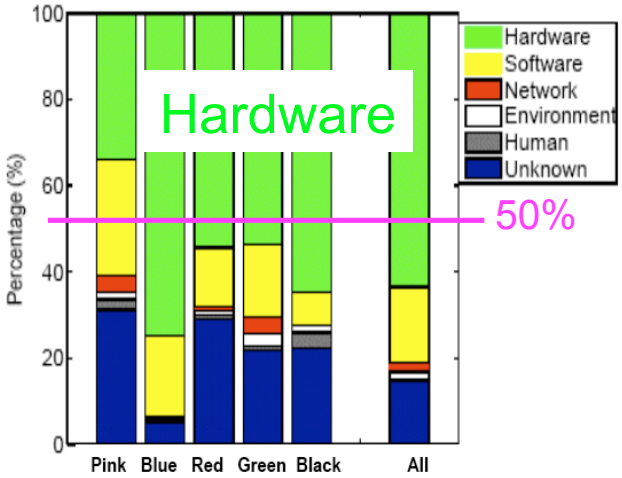
- Phase-change memory
  - read bandwidth 100GB/sec
  - write bandwidth 10GB/sec
- Checkpoint size 128GB

- C: checkpoint save time: $C = 12sec$
- R: checkpoint recovery time: $R = 1.2sec$
- D: down/reboot time: $D = 15sec$

Error sources (courtesy F. Cappello)

Sources of failures

- Analysis of error and failure logs
- In 2005 (Ph. D. of CHARNG-DA LU) : "Software halts account for the most number of outages (59-84 percent), and take the shortest time to repair (0.6-1.5 hours). Hardware problems, albeit rarer, need 6.3-100.7 hours to solve."
- In 2007 (Garth Gibson, ICPP Keynote):



Relative frequency of root cause by system type.

- In 2008 (Oliner and J. Stearley, DSN Conf.):

Type	Raw		Filtered	
	Count	%	Count	%
Hardware	174,586,516	98.04	1,999	18.78
Software	144,899	0.08	6,814	64.01
Indeterminate	3,350,044	1.88	1,832	17.21

Software errors: Applications, OS bug (kernel panic), communication libs, File system error and other.

Hardware errors, Disks, processors, memory, network

Conclusion: Both Hardware and Software failures have to be considered

A few definitions

- Many types of failures: software error, hardware malfunction, memory corruption
- Many possible behaviors: silent, transient, unrecoverable
- Restrict to failures that lead to application failures
- This includes all hardware failures, and some software ones

Outline

Energy: a crucial issue

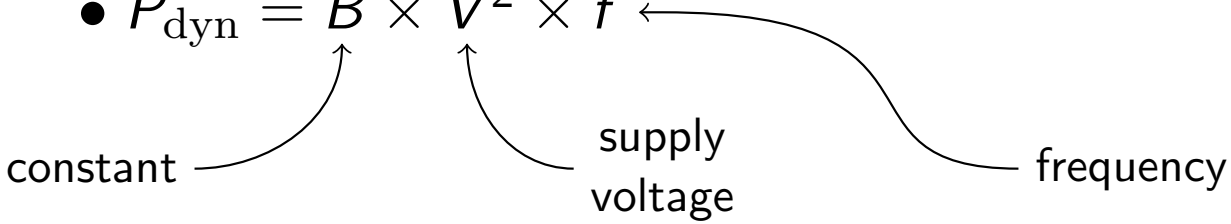
- Data centers
  - $330,000,000,000$ Watts hour in 2007: more than France
  - $533,000,000$ tons of CO_2: in the top ten countries
- Exascale computers (10^{18} floating operations per second)
  - Need effort for feasibility
  - 1% of power saved $\leadsto$ 1 million dollar per year
- Lambda user
  - 1 billion personal computers
  - $500,000,000,000,000$ Watts hour per year
- $\leadsto$ crucial for both environmental and economical reasons

Energy: a crucial issue

- Data centers
  - 330,000,00 ... more than France
  - 533,000,00 ... countries
- Exascale compu ... ons per second)
  - Need effort
  - 1% of powe ... r year
- Lambda user
  - 1 billion per
  - 500,000,00 ... ear
- $\rightsquigarrow$ crucial for both environmental and economical reasons

Power dissipation of a processor

- $P = P_{\text{leak}} + P_{\text{dyn}}$
  - P_{leak}: constant
  - $P_{\text{dyn}} = B \times V^2 \times f$



- Standard approximation: $P = P_{\text{leak}} + f^{\alpha}$ $\quad (2 \leq \alpha \leq 3)$
- Energy $E = P \times \textit{time}$
- Dynamic Voltage and Frequency Scaling (DVFS) to reduce dynamic power
  - Real life: discrete speeds
  - Continuous speeds can be emulated
- Processor shutdown to reduce static power

Speed models for DVFS

		When can we change speed?	
		Anytime	**Beginning of tasks**
Type of speeds	$[s_{min}, s_{max}]$	CONTINUOUS	-
	$\{s_1, ..., s_m\}$	VDD-HOPPING	DISCRETE, INCREMENTAL

- CONTINUOUS: great for theory
- Other "discrete" models more realistic
- VDD-HOPPING simulates CONTINUOUS
- INCREMENTAL is a special case of DISCRETE with equally-spaced speeds: for all $1 \leq q < m$, $s_{q+1} - s_q = \delta$

Outline

Framework

- DAG: $\mathcal{G} = (V, E)$
- $n = |V|$ tasks T_i of weight w_i
- p identical processors fully connected
- DVFS, CONTINUOUS model:
  interval of available continuous speeds $[s_{\min}, s_{\max}]$
- One speed per task

Makespan

Execution time of T_i at speed s_i:

$$d_i = \frac{w_i}{s_i}$$

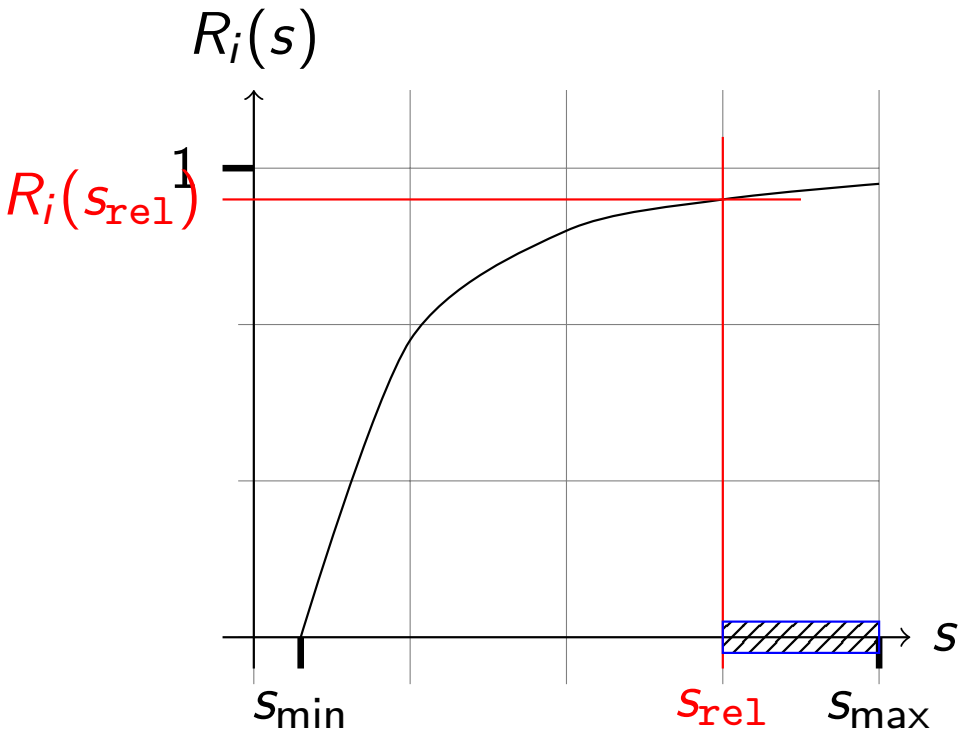
If T_i is executed twice on the same processor at speeds s_i and s_i':

$$d_i = \frac{w_i}{s_i} + \frac{w_i}{s_i'}$$

Constraint on makespan:
end of execution before deadline D

Reliability

- *Transient failure*: local, no impact on the rest of the system
- Reliability R_i of task T_i as a function of speed s
- Threshold reliability (and hence speed $s_{\texttt{rel}}$)



Re-execution: a task is re-executed *on the same processor, just after its first execution*

With two executions, reliability R_i of task T_i is:

$$R_i = 1 - (1 - R_i(s_i))(1 - R_i(s_i'))$$

Constraint on reliability:
RELIABILITY: $R_i \geq R_i(s_{\texttt{rel}})$, and at most one re-execution

Energy

- Energy to execute task T_i once at speed s_i:

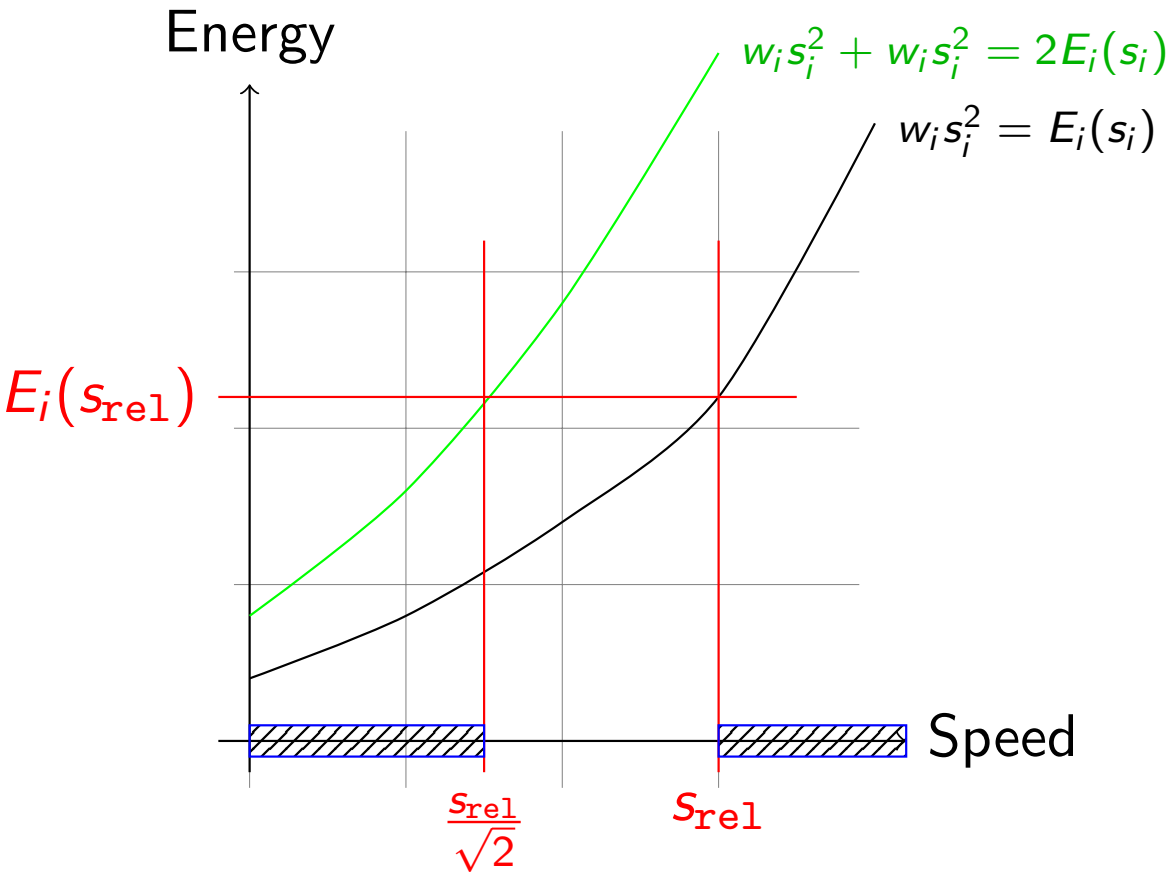
$$E_i(s_i) = w_i s_i^2$$

$\rightarrow$ Dynamic part of classical energy models

- With re-executions, it is natural to take the worst-case scenario:

$$\textsc{Energy} : E_i = w_i \left(s_i^2 + s_i'^2\right)$$

Energy and reliability: set of possible speeds



Tri-Crit-Cont

Given $\mathcal{G} = (V, E)$
Find

- A schedule of the tasks
- A set of tasks $I = \{i \mid T_i \text{ is executed twice}\}$
- Execution speed s_i for each task T_i
- Re-execution speed s_i' for each task in I

such that

$$\sum_{i \in I} w_i (s_i^2 + s_i'^2) + \sum_{i \notin I} w_i s_i^2$$

is minimized, while meeting reliability and deadline constraints

Complexity results

- One speed per task
- Re-execution at same speed as first execution, i.e., $s_i = s'_i$

- TRI-CRIT-CONT is NP-hard even for a linear chain, but not known to be in NP (because of continuous model)
- Polynomial-time solution for a fork

Energy-reducing heuristics

Two steps:

- Mapping (NP-hard) $\rightarrow$ List scheduling
- Speed scaling + re-execution (NP-hard) $\rightarrow$ Energy reducing

- The list scheduling heuristic maps tasks onto processors at speed s_{max}, and we keep this mapping in step two
- Step two = slack reclamation: use of deceleration and re-execution

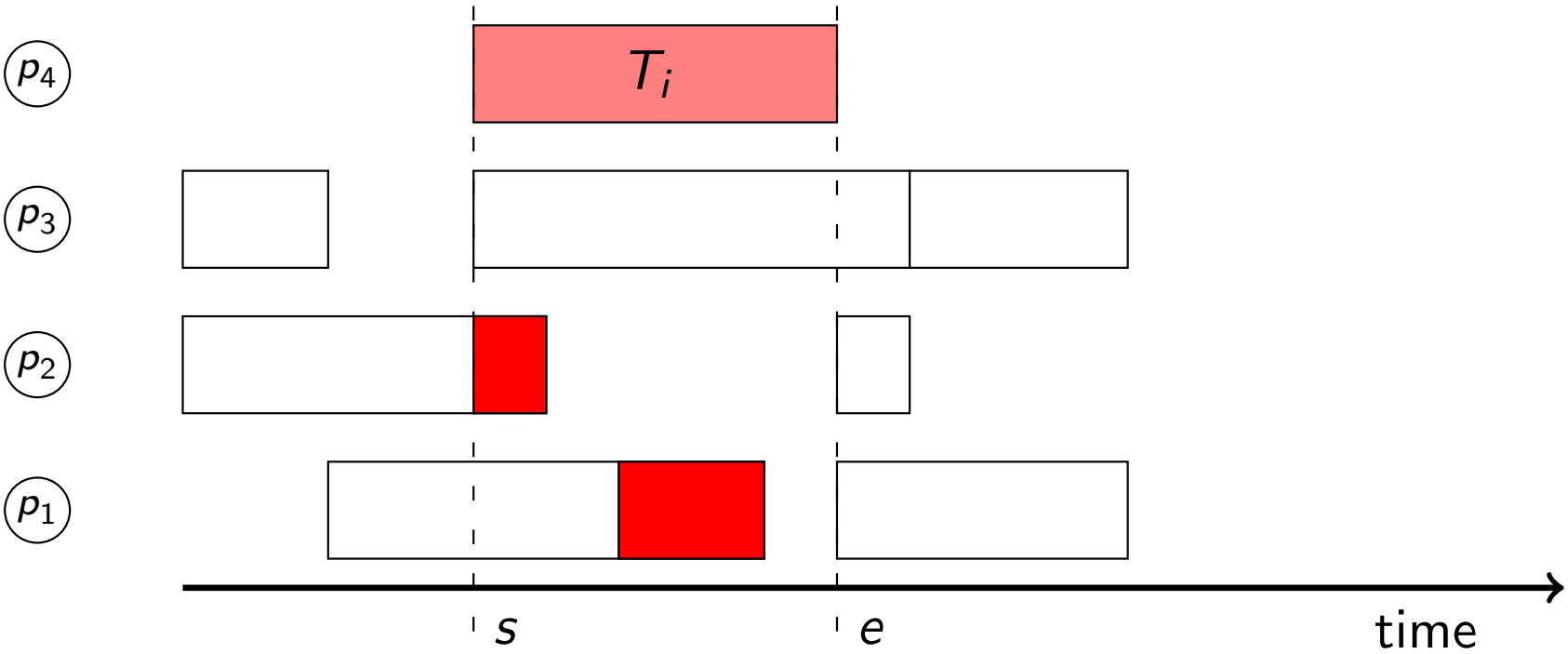
Deceleration and re-execution

- Deceleration: select a set of tasks that we execute at speed $\max(s_{\texttt{rel}}, s_{\max} \frac{\max_{i=1..n} t_i}{D})$: slowest possible speed meeting both reliability and deadline constraints

- Re-execution: greedily select tasks for re-execution

Super-weight (SW) of a task

- SW: sum of the weights of the tasks (including T_i) whose execution interval is included into T_i's execution interval
- SW of task slowed down = estimation of the total amount of work that can be slowed down together with that task



Selected heuristics

- **A.SUS-Crit**: efficient on DAGs with low degree of parallelism
  - Set the speed of every task to $\max(s_{\texttt{rel}}, s_{\max} \frac{\max_{i=1..n} t_i}{D})$
  - Sort the tasks of every *critical path* according to their **SW** and try to re-execute them
  - Sort all the tasks according to their **weight** and try to re-execute them

- **B.SUS-Crit-Slow**: good for highly parallel tasks: re-execute, then decelerate
  - Sort the tasks of every *critical path* according to their **SW** and try to re-execute them. If not possible, then try to slow them down
  - Sort all tasks according to their **weight** and try to re-execute them. If not possible, then try to slow them down

Results

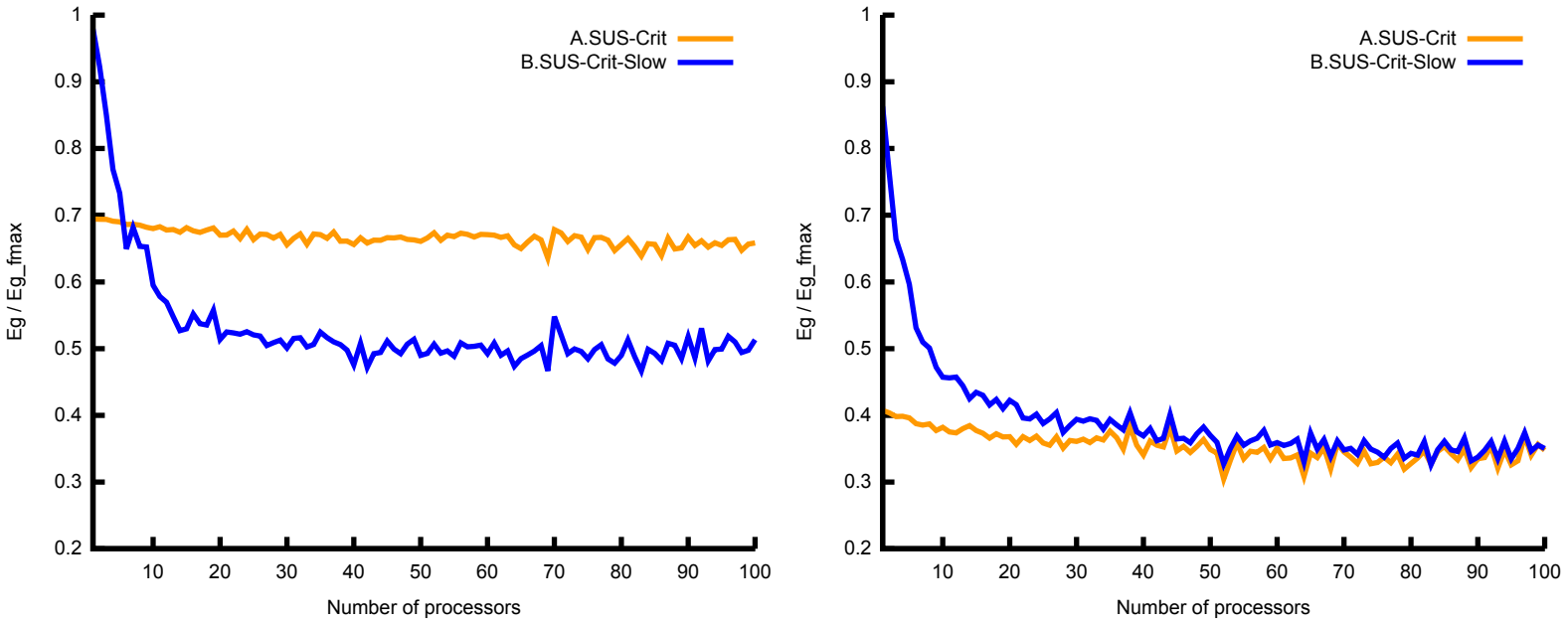
We compare the impact of:

- the number of processors p
- the ratio D of the deadline over the minimum deadline D_{min} (given by the list-scheduling heuristic at speed s_{max})

on the output of each heuristic

Results normalized by heuristic running each task at speed s_{max}; the lower the better

Results



With increasing p, $D = 1.2$ (left), $D = 2.4$ (right)

- A better when number of processors is small
- B better when number of processors is large
- Superiority of B for tight deadlines: decelerates critical tasks that cannot be re-executed

Summary

- Tri-criteria energy/makespan/reliability optimization problem
- Various theoretical results
- Two-step approach for polynomial-time heuristics:
  - List-scheduling heuristic
  - Energy-reducing heuristics
- Two complementary energy-reducing heuristics for TRI-CRIT-CONT

Outline

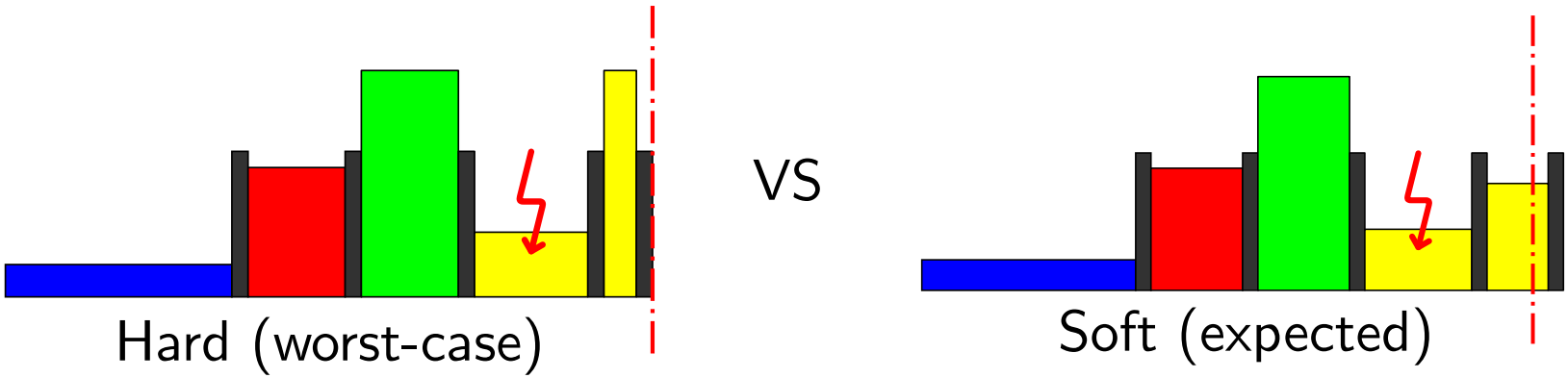
Framework

- Execution of a divisible task (W operations)
- Failures may occur
  - Transient failures
  - Resilience through checkpointing
- Objective: minimize expected energy consumption $\mathbb{E}(E)$, given a deadline bound D

- Probabilistic nature of failure hits: expectation of energy consumption is natural (average cost over many executions)
- Deadline bound: two relevant scenarios (soft or hard deadline)

Soft vs hard deadline

- Soft deadline: met in expectation, i.e., $\mathbb{E}(T) \leq D$ (average response time)
- Hard deadline: met in the worst case, i.e., $T_{wc} \leq D$



Execution time, one single chunk

One single chunk of size W

- Checkpoint overhead: execution time T_C
- Instantaneous failure rate: λ

- First execution at speed s: $T_{\text{exec}} = \frac{W}{s} + T_C$
- Failure probability: $P_{\text{fail}} = \lambda T_{\text{exec}} = \lambda(\frac{W}{s} + T_C)$
- In case of failure: re-execute at speed σ: $T_{\text{reexec}} = \frac{W}{\sigma} + T_C$
- And we assume success after re-execution

- $\mathbb{E}(T) = T_{\text{exec}} + P_{\text{fail}} T_{\text{reexec}} = (\frac{W}{s} + T_C) + \lambda(\frac{W}{s} + T_C)(\frac{W}{\sigma} + T_C)$
- $T_{wc} = T_{\text{exec}} + T_{\text{reexec}} = (\frac{W}{s} + T_C) + (\frac{W}{\sigma} + T_C)$

Energy consumption, one single chunk

One single chunk of size W

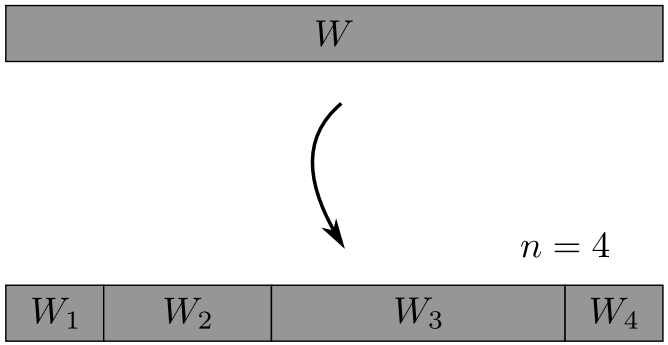
- Checkpoint overhead: energy consumption E_C
- First execution at speed s: $\frac{W}{s} \times s^3 + E_C = Ws^2 + E_C$
- Re-execution at speed σ: $W\sigma^2 + E_C$, with probability P_{fail} $\left(P_{\text{fail}} = \lambda T_{\text{exec}} = \lambda(\frac{W}{s} + T_C)\right)$
- $\mathbb{E}(E) = (Ws^2 + E_C) + \lambda \left(\frac{W}{s} + T_C\right) \left(W\sigma^2 + E_C\right)$

Multiple chunks

- Execution times: sum of execution times for each chunk (worst-case or expected)
- Expected energy consumption: sum of expected energy for each chunk
- Coherent failure model: consider two chunks $W_1 + W_2 = W$
- Probability of failure for first chunk: $P^1_{\text{fail}} = \lambda(\frac{W_1}{s} + T_C)$
- For second chunk: $P^2_{\text{fail}} = \lambda(\frac{W_2}{s} + T_C)$
- With a single chunk of size W: $P_{\text{fail}} = \lambda(\frac{W}{s} + T_C)$, differs from $P^1_{\text{fail}} + P^2_{\text{fail}}$ only because of extra checkpoint
- Trade-off: many small chunks (more T_C to pay, but small re-execution cost) vs few larger chunks (fewer T_C, but increased re-execution cost)

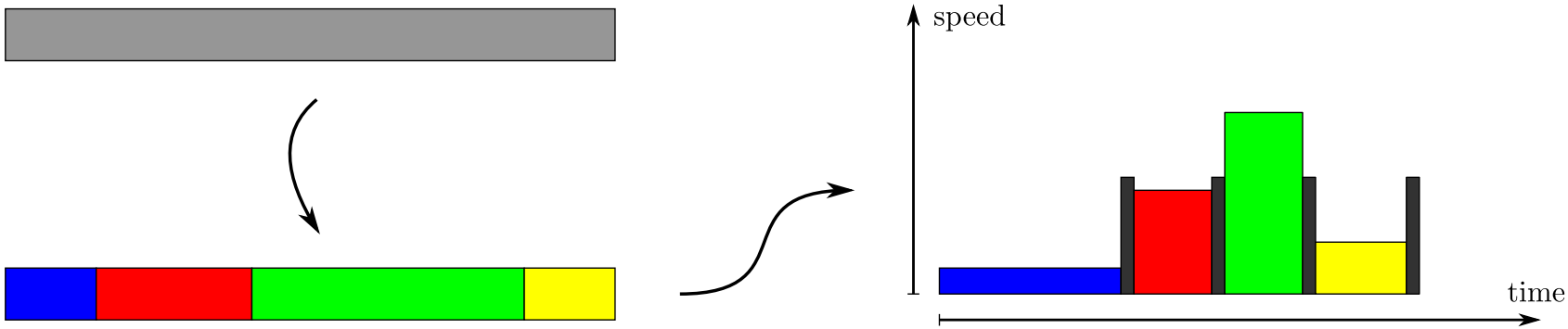
Optimization problem

- Decisions that should be taken before execution:
  - Chunks: how many (n)? which sizes (W_i for chunk i)?
  - Speeds of each chunk: first run (s_i)? re-execution (σ_i)?
- Input: W, T_C (checkpointing time), E_C (energy spent for checkpointing), λ (instantaneous failure rate), D (deadline)



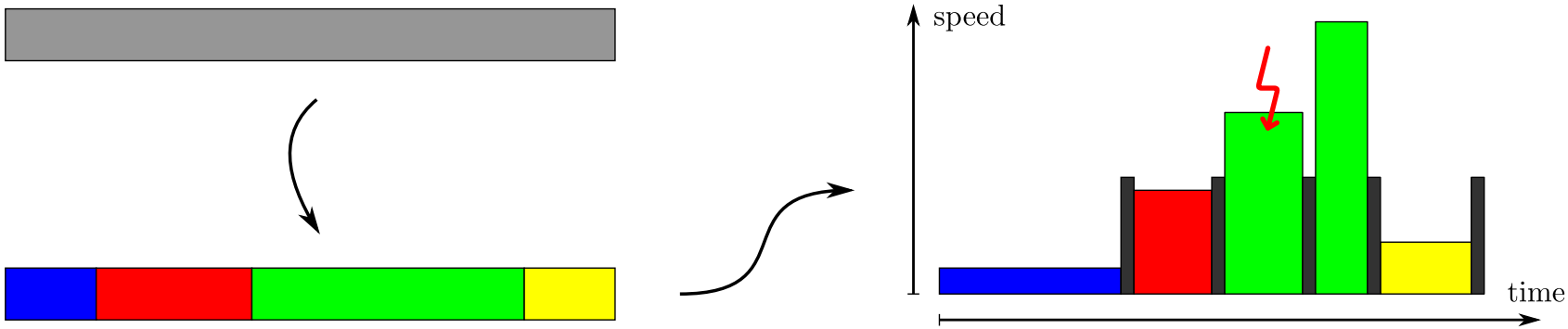
Optimization problem

- Decisions that should be taken before execution:
  - Chunks: how many (n)? which sizes (W_i for chunk i)?
  - Speeds of each chunk: first run (s_i)? re-execution (σ_i)?
- Input: W, T_C (checkpointing time), E_C (energy spent for checkpointing), λ (instantaneous failure rate), D (deadline)



Optimization problem

- Decisions that should be taken before execution:
  - Chunks: how many (n)? which sizes (W_i for chunk i)?
  - Speeds of each chunk: first run (s_i)? re-execution (σ_i)?
- Input: W, T_C (checkpointing time), E_C (energy spent for checkpointing), λ (instantaneous failure rate), D (deadline)



Models

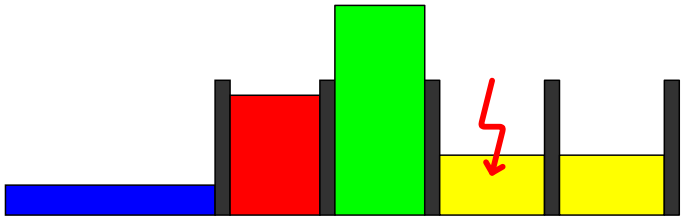
- Chunks

Single chunk of size W

VS

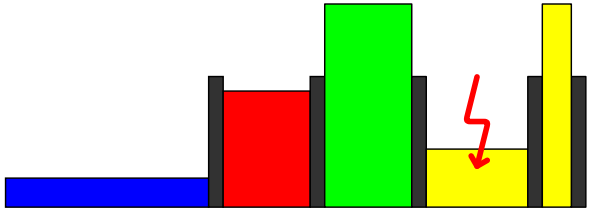
Multiple chunks (n and W_i's)

- Speed per chunk



Single speed (s)

VS

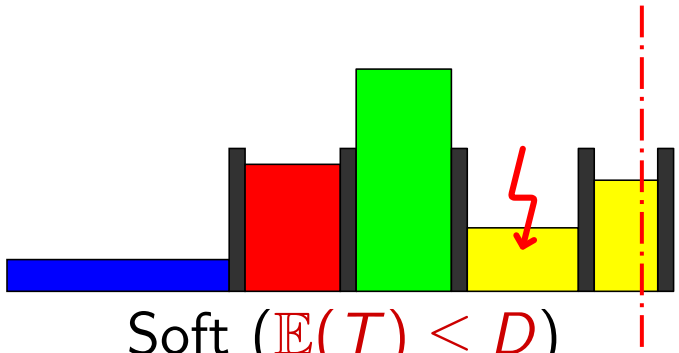


Multiple speeds (s and σ)

- Deadline bound

Hard ($T_{wc} \leq D$)

VS



Soft ($\mathbb{E}(T) \leq D$)

Single chunk and single speed

Consider first that $s = \sigma$ (single speed): need to find optimal speed

- $\mathbb{E}(E)$ is a function of s:
$$\mathbb{E}(E)(s) = (Ws^2 + E_C)(1 + \lambda(\frac{W}{s} + T_C))$$
- Lemma: this function is convex and has a unique minimum $s^\star$ (function of λ, W, E_c, T_c)
$$s^\star = \frac{\lambda W}{6(1+\lambda T_C)} \left(\frac{-(3\sqrt{3}\sqrt{27a^2-4a}-27a+2)^{1/3}}{2^{1/3}} - \frac{2^{1/3}}{(3\sqrt{3}\sqrt{27a^2-4a}-27a+2)^{1/3}} - 1 \right),$$
where $a = \lambda E_C \left(\frac{2(1+\lambda T_C)}{\lambda W}\right)^2$
- $\mathbb{E}(T)$ and T_{wc}: decreasing functions of s
- Minimum speed s_{exp} and s_{wc} required to match deadline D (function of D, W, T_c, and λ for s_{exp})

$\rightarrow$ Optimal speed: maximum between $s^\star$ and s_{exp} or s_{wc}

Single chunk and multiple speeds

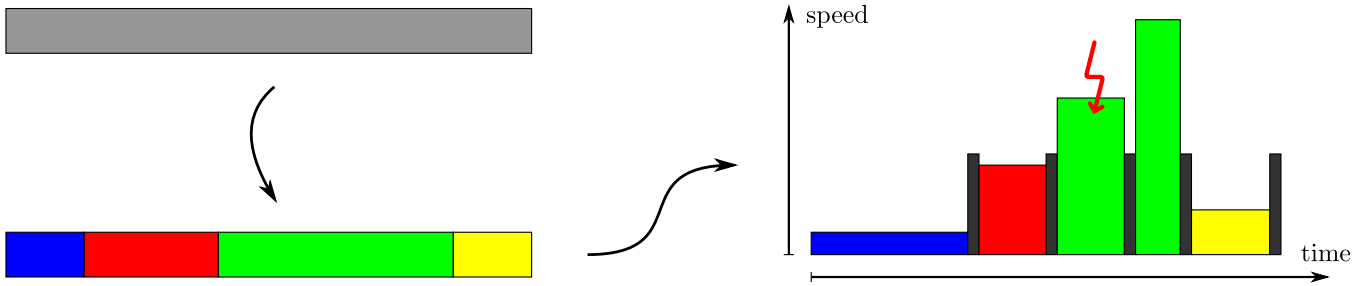
Consider now that $s \neq \sigma$ (multiple speeds): two unknowns

- $\mathbb{E}(E)$ is a function of s and σ:
$$\mathbb{E}(E)(s,\sigma) = (Ws^2 + E_C) + \lambda(\frac{W}{s} + T_C)(W\sigma^2 + E_C)$$
- Lemma: energy minimized when deadline tight (both for wc and exp)
- $\rightsquigarrow$ σ expressed as a function of s:
$$\sigma_{exp} = \frac{\lambda W}{\frac{D}{\frac{W}{s}+T_C} - (1+\lambda T_C)}, \quad \sigma_{wc} = \frac{W}{(D-2T_C)s - W}s$$

$\rightarrow$ Minimization of single-variable function, can be solved numerically (no expression of optimal s)

General problem with multiple chunks

- Divisible task of size W
- Split into n chunks of size W_i: $\sum_{i=1}^{n} W_i = W$
- Chunk i is executed once at speed s_i, and re-executed (if necessary) at speed σ_i
- Unknowns: n, W_i, s_i, σ_i
- $$\mathbb{E}(E) = \sum_{i=1}^{n} \left(W_i s_i^2 + E_C\right) + \lambda \sum_{i=1}^{n} \left(\frac{W_i}{s_i} + T_C\right) \left(W_i \sigma_i^2 + E_C\right)$$



Multiple chunks and single speed

With a single speed, $\sigma_i = s_i$ for each chunk

- Theorem: in optimal solution, n equal-sized chunks ($W_i = \frac{W}{n}$), executed at same speed $s_i = s$
  - Proof by contradiction: consider two chunks W_1 and W_2 executed at speed s_1 and s_2, with either $s_1 \neq s_2$, or $s_1 = s_2$ and $W_1 \neq W_2$
  - $\Rightarrow$ Strictly better solution with two chunks of size $w = (W_1 + W_2)/2$ and same speed s

- Only two unknowns, s and n
- Minimum speed with n chunks: $s^{\star}_{exp}(n) = W \dfrac{1 + 2\lambda T_C + \sqrt{4\frac{\lambda D}{n} + 1}}{2(D - nT_C(1 + \lambda T_C))}$

$\rightarrow$ Minimization of double-variable function, can be solved numerically both for expected and hard deadline

Multiple chunks and multiple speeds

Need to find n, W_i, s_i, σ_i

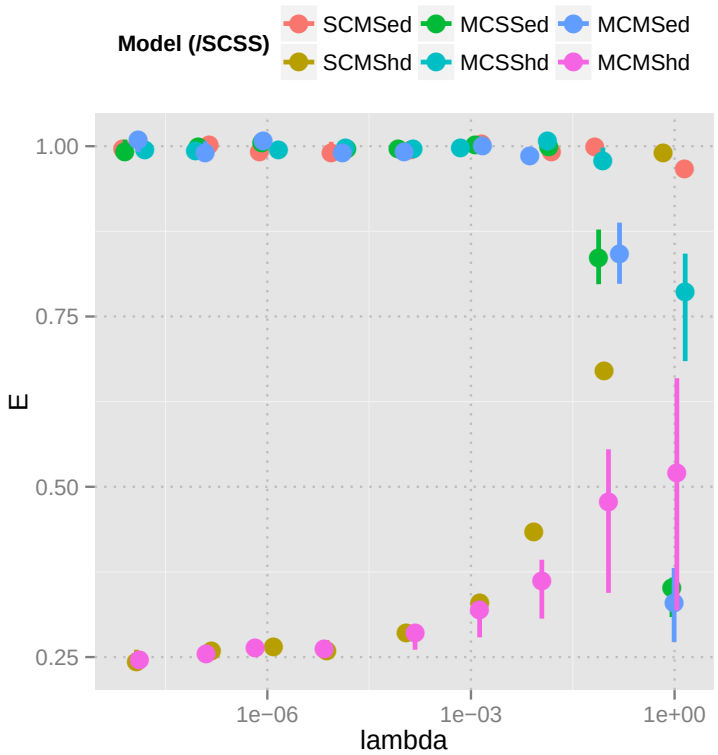
- With expected deadline:
  - All re-execution speeds are equal ($\sigma_i = \sigma$) and tight deadline
  - All chunks have same size and are executed at same speed
- WIth hard deadline:
  - If $s_i = s$ and $\sigma_i = \sigma$, then all W_i's are equal
  - Conjecture: equal-sized chunks, same first-execution / re-execution speeds
- σ as a function of s, bound on s given n

$\rightarrow$ Minimization of double-variable function, can be solved numerically

Simulation settings

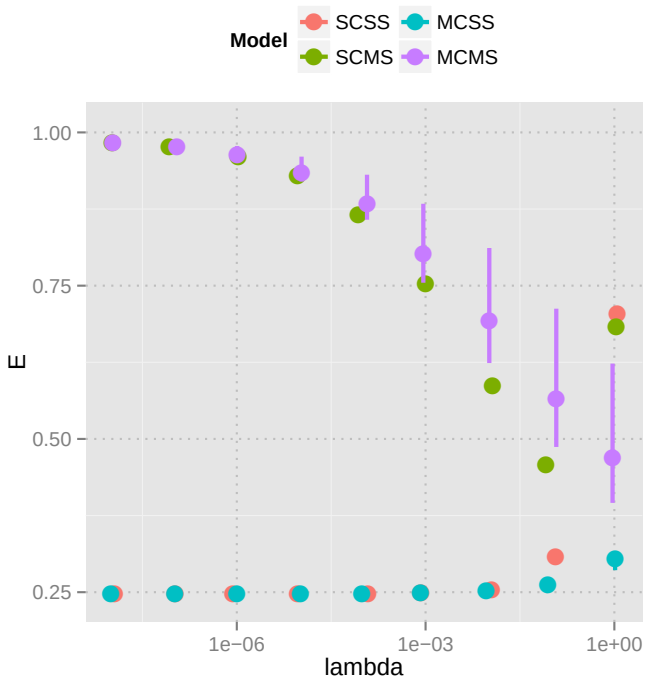
- Large set of simulations: illustrate differences between models
- Maple software to solve problems
- We plot relative energy consumption as a function of λ
  - The lower the better
  - Given a deadline constraint (hard or expected), normalize with the result of single-chunk single-speed
  - Impact of the constraint: normalize expected deadline with hard deadline
- Parameters varying within large ranges

Comparison with single-chunk single-speed



- Results identical for any value of W/D
- For expected deadline, with small λ ($< 10^{-2}$), using multiple chunks or multiple speeds do not improve energy ratio: re-execution term negligible;
  increasing λ: improvement with multiple chunks
- For hard deadline, better to run at high speed during second execution: use multiple speeds; use multiple chunks if frequent failures

Expected vs hard deadline constraint



- Important differences for single speed models, confirming previous conclusions: with hard deadline, use multiple speeds
- Multiple speeds: no difference for small λ: re-execution at maximum speed has little impact on expected energy consumption;
  increasing λ: more impact of re-execution, and expected deadline may use slower re-execution speed, hence reducing energy consumption

Outline

Conclusion

- Resilience and energy consumption are two of the main challenges for Exascale platforms
- Tri-criteria heuristics aiming at minimizing the energy consumption, with re-execution to deal with reliability
- Checkpointing techniques for reliability while minimizing energy consumption

On-going and future research directions

- Investigate other reliability models (for instance, local constraints on reliability of each task, or global reliability of success of the execution of the DAG)
- Consider both re-execution and replication (recent results for linear chains and independent tasks: approximation algorithms)
- Checkpointing at the exascale: find the optimal checkpointing period (with the goal of minimizing the energy consumption)

What we had:

Energy-efficient scheduling + frequency scaling

What we aim at: