Fault tolerance techniques for high-performance computing Part 2

Anne Benoit

ENS Lyon

Anne.Benoit@ens-lyon.fr

http://graal.ens-lyon.fr/~abenoit

CR02 - 2016/2017

Outline

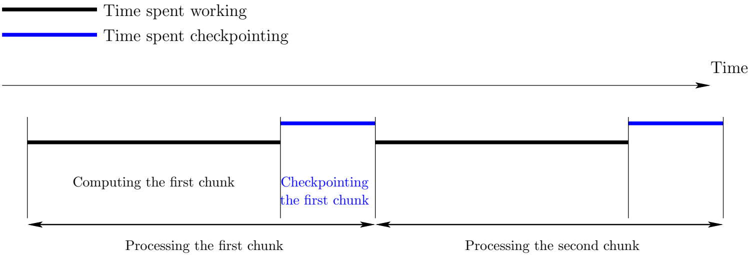
Outline

Outline

Outline

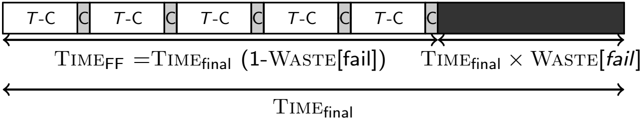
Outline

Checkpointing cost



Blocking model: while a checkpoint is taken, no computation can be performed

Total waste



$$\text{WASTE} = \frac{\text{TIME}_{\text{final}} - \text{TIME}_{\text{base}}}{\text{TIME}_{\text{final}}}$$

$$1 - \text{WASTE} = (1 - \text{WASTE}[FF])(1 - \text{WASTE}[fail])$$

$$\text{WASTE} = \frac{C}{T} + \left(1 - \frac{C}{T}\right)\frac{1}{\mu}\left(D + R + \frac{T}{2}\right)$$

Waste minimization

$$\textsc{Waste} = \frac{C}{T} + \left(1 - \frac{C}{T}\right)\frac{1}{\mu}\left(D + R + \frac{T}{2}\right)$$

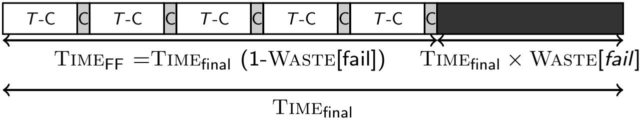
$$\textsc{Waste} = \frac{u}{T} + v + wT$$

$$u = C\left(1 - \frac{D+R}{\mu}\right) \qquad v = \frac{D+R-C/2}{\mu} \qquad w = \frac{1}{2\mu}$$

$\textsc{Waste}$ minimized for $T = \sqrt{\frac{u}{w}}$

$$T = \sqrt{2(\mu - (D+R))C}$$

Comparison with Young/Daly



$$\left(1 - \text{WASTE}[\mathit{fail}]\right)\text{TIME}_{\mathsf{final}} = \text{TIME}_{\mathsf{FF}}$$
$$\Rightarrow T = \sqrt{2(\mu - (D+R))C}$$

Daly: $\text{TIME}_{\mathsf{final}} = \left(1 + \text{WASTE}[\mathit{fail}]\right)\text{TIME}_{\mathsf{FF}}$
$$\Rightarrow T = \sqrt{2(\mu + (D+R))C} + C$$

Young: $\text{TIME}_{\mathsf{final}} = \left(1 + \text{WASTE}[\mathit{fail}]\right)\text{TIME}_{\mathsf{FF}}$ and $D = R = 0$
$$\Rightarrow T = \sqrt{2\mu C} + C$$

Validity of the approach (1/3)

Technicalities

- $\mathbb{E}(N_{faults}) = \frac{\text{TIME}_{\text{final}}}{\mu}$ and $\mathbb{E}(T_{\text{lost}}) = D + R + \frac{T}{2}$
  but expectation of product is not product of expectations (not independent RVs here)

- Enforce $C \leq T$ to get $\text{WASTE}[FF] \leq 1$

- Enforce $D + R \leq \mu$ and bound T to get $\text{WASTE}[fail] \leq 1$
  but $\mu = \frac{\mu_{ind}}{p}$ too small for large p, regardless of μ_{ind}

Validity of the approach (2/3)

Several failures within same period?

- WASTE[fail] accurate only when two or more faults do not take place within same period
- Cap period: $T \leq \gamma\mu$, where γ is some tuning parameter
  - Poisson process of parameter $\theta = \frac{T}{\mu}$
  - Probability of having $k \geq 0$ failures : $P(X = k) = \frac{\theta^k}{k!} e^{-\theta}$
  - Probability of having two or more failures: $\pi = P(X \geq 2) = 1 - (P(X = 0) + P(X = 1)) = 1 - (1 + \theta)e^{-\theta}$
  - $\gamma = 0.27 \Rightarrow \pi \leq 0.03$
    $\Rightarrow$ overlapping faults for only 3% of checkpointing segments

Validity of the approach (3/3)

- Enforce $T \leq \gamma\mu$, $C \leq \gamma\mu$, and $D + R \leq \gamma\mu$
- Optimal period $\sqrt{2(\mu - (D + R))C}$ may not belong to admissible interval $[C, \gamma\mu]$
- Waste is then minimized for one of the bounds of this admissible interval (by convexity)

Wrap up

- Capping periods, and enforcing a lower bound on MTBF
  $\Rightarrow$ mandatory for mathematical rigor ☹

- Not needed for practical purposes ☺
  - actual job execution uses optimal value
  - account for multiple faults by re-executing work until success

- Approach surprisingly robust ☺

Lesson learnt for fail-stop failures

(Not so) Secret data

- Tsubame 2: 962 failures during last 18 months so $\mu = 13$ hrs
- Blue Waters: 2-3 node failures per day
- Titan: a few failures per day
- Tianhe 2: wouldn't say

$$T_{\text{opt}} = \sqrt{2\mu C} \quad \Rightarrow \quad \text{WASTE}[opt] \approx \sqrt{\frac{2C}{\mu}}$$

Petascale:	$C = 20$ min	$\mu = 24$ hrs	$\Rightarrow$ WASTE$[opt] = 17\%$
Scale by 10:	$C = 20$ min	$\mu = 2.4$ hrs	$\Rightarrow$ WASTE$[opt] = 53\%$
Scale by 100:	$C = 20$ min	$\mu = 0.24$ hrs	$\Rightarrow$ WASTE$[opt] = 100\%$

Lesson learnt for fail-stop failures

(... so) Secret data

- Tsuban... 962 failures during last 18 months so ... 13 hrs
- Blue Waters: 2-... de failures per day
- Titan: a few failures pe...
- Tianhe...

Exascale $\neq$ Petascale $\times 1000$
Need more reliable components
Need to checkpoint faster

Petascale:	$C = 20$ min	$\mu = 24$ hrs	$\Rightarrow$ WASTE[*opt*] $= 17\%$
Scale by 10:	$C = 20$ min	$\mu = 2.4$ hrs	$\Rightarrow$ WASTE[*opt*] $= 53\%$
Scale by 100:	$C = 20$ min	$\mu = 0.24$ hrs	$\Rightarrow$ WASTE[*opt*] $= 100\%$

Lesson learnt for fail-stop failures

(Not so) Secret data

- Tsubame 2: 962 failures during last 18 months so $\mu = 13$ hrs
- Blue Waters: 2-3 node failures per day
- Titan: a few failures per day
- Tianhe 2: wouldn't say

Silent errors:
detection latency $\Rightarrow$ additional problems

Petascale:	$C = 20$ min	$\mu = 24$ hrs	$\Rightarrow \textsc{Waste}[opt] = 17\%$
Scale by 10:	$C = 20$ min	$\mu = 2.4$ hrs	$\Rightarrow \textsc{Waste}[opt] = 53\%$
Scale by 100:	$C = 20$ min	$\mu = 0.24$ hrs	$\Rightarrow \textsc{Waste}[opt] = 100\%$

Outline

Exponential failure distribution

1. Expected execution time for a single chunk
2. Expected execution time for a sequential job
3. Expected execution time for a parallel job

Expected execution time for a single chunk

Compute the expected time $\mathbb{E}(T(W, C, D, R, \lambda))$ to execute a work of duration W followed by a checkpoint of duration C.

Recursive Approach

$$\mathbb{E}(T(W)) =$$

Expected execution time for a single chunk

Compute the expected time $\mathbb{E}(T(W, C, D, R, \lambda))$ to execute a work of duration W followed by a checkpoint of duration C.

Recursive Approach

$$\mathbb{E}(T(W)) = \overbrace{\mathcal{P}_{\text{succ}}(W+C)}^{\substack{\text{Probability} \\ \text{of success}}}(W+C)$$

Expected execution time for a single chunk

Compute the expected time $\mathbb{E}(T(W, C, D, R, \lambda))$ to execute a work of duration W followed by a checkpoint of duration C.

Recursive Approach

$$\mathbb{E}(T(W)) = \mathcal{P}_{\text{succ}}(W+C)\overbrace{(W+C)}^{\substack{\text{Time needed}\\ \text{to compute}\\ \text{the work } W \text{ and}\\ \text{checkpoint it}}}$$

Expected execution time for a single chunk

Compute the expected time $\mathbb{E}(T(W, C, D, R, \lambda))$ to execute a work of duration W followed by a checkpoint of duration C.

Recursive Approach

$$\mathbb{E}(T(W)) = \quad \mathcal{P}_{\text{succ}}(W + C)\,(W + C)$$

Expected execution time for a single chunk

Compute the expected time $\mathbb{E}(T(W, C, D, R, \lambda))$ to execute a work of duration W followed by a checkpoint of duration C.

Recursive Approach

$$\mathbb{E}(T(W)) = \begin{array}{l} \mathcal{P}_{\text{succ}}(W+C)\,(W+C) \\ + \\ (1-\mathcal{P}_{\text{succ}}(W+C))\,(\mathbb{E}(T_{lost}(W+C)) + \mathbb{E}(T_{rec}) + \mathbb{E}(T(W))) \end{array}$$

Expected execution time for a single chunk

Compute the expected time $\mathbb{E}(T(W, C, D, R, \lambda))$ to execute a work of duration W followed by a checkpoint of duration C.

Recursive Approach

$$\mathbb{E}(T(W)) = \begin{array}{l} \mathcal{P}_{\text{succ}}(W+C)\,(W+C) \\ + \\ \underbrace{(1-\mathcal{P}_{\text{succ}}(W+C))}_{\text{Probability of failure}}\,(\mathbb{E}(T_{lost}(W+C)) + \mathbb{E}(T_{rec}) + \mathbb{E}(T(W))) \end{array}$$

Expected execution time for a single chunk

Compute the expected time $\mathbb{E}(T(W, C, D, R, \lambda))$ to execute a work of duration W followed by a checkpoint of duration C.

Recursive Approach

$$\mathbb{E}(T(W)) = \begin{array}{l} \mathcal{P}_{\text{succ}}(W+C)\,(W+C) \\ + \\ (1-\mathcal{P}_{\text{succ}}(W+C))\,(\underbrace{\mathbb{E}(T_{lost}(W+C))}_{\substack{\text{Time elapsed} \\ \text{before failure} \\ \text{stroke}}} + \mathbb{E}(T_{rec}) + \mathbb{E}(T(W))) \end{array}$$

Expected execution time for a single chunk

Compute the expected time $\mathbb{E}(T(W, C, D, R, \lambda))$ to execute a work of duration W followed by a checkpoint of duration C.

Recursive Approach

$$\mathbb{E}(T(W)) = \begin{array}{l} \mathcal{P}_{\text{succ}}(W+C)\,(W+C) \\ + \\ (1-\mathcal{P}_{\text{succ}}(W+C))\,(\mathbb{E}(T_{lost}(W+C)) + \underbrace{\mathbb{E}(T_{rec})}_{\substack{\text{Time needed} \\ \text{to perform} \\ \text{downtime} \\ \text{and recovery}}} + \mathbb{E}(T(W))) \end{array}$$

Expected execution time for a single chunk

Compute the expected time $\mathbb{E}(T(W, C, D, R, \lambda))$ to execute a work of duration W followed by a checkpoint of duration C.

Recursive Approach

$$\mathbb{E}(T(W)) = \begin{array}{l} \mathcal{P}_{\text{succ}}(W+C)\,(W+C) \\ + \\ (1-\mathcal{P}_{\text{succ}}(W+C))\,(\mathbb{E}(T_{lost}(W+C)) + \mathbb{E}(T_{rec}) + \underbrace{\mathbb{E}(T(W))}_{\substack{\text{Time needed}\\ \text{to compute } W\\ \text{anew}}}) \end{array}$$

Computation of $\mathbb{E}(T(W, C, D, R, \lambda))$

$$\mathbb{E}(T(W)) = \begin{array}{l} \mathcal{P}_{\text{succ}}(W+C)\,(W+C) \\ + \\ (1-\mathcal{P}_{\text{succ}}(W+C))\,(\mathbb{E}(T_{lost}(W+C)) + \mathbb{E}(T_{rec}) + \mathbb{E}(T(W))) \end{array}$$

- $\mathbb{P}_{suc}(W+C) = e^{-\lambda(W+C)}$
- $\mathbb{E}(T_{lost}(W+C)) = \int_0^\infty x\mathbb{P}(X = x | X < W+C)dx = \frac{1}{\lambda} - \frac{W+C}{e^{\lambda(W+C)}-1}$
- $\mathbb{E}(T_{rec}) = e^{-\lambda R}(D+R) + (1-e^{-\lambda R})(D + \mathbb{E}(T_{lost}(R)) + \mathbb{E}(T_{rec}))$

$$\mathbb{E}(T(W, C, D, R, \lambda)) = e^{\lambda R}\left(\frac{1}{\lambda} + D\right)\left(e^{\lambda(W+C)} - 1\right)$$

Checkpointing a sequential job

- $\mathbb{E}(T(W)) = e^{\lambda R}\left(\frac{1}{\lambda} + D\right)\left(\sum_{i=1}^{K} e^{\lambda(W_i+C)} - 1\right)$
- Optimal strategy uses same-size chunks (convexity)
- $K_0 = \frac{\lambda W}{1+\mathbb{L}(-e^{-\lambda C-1})}$ where $\mathbb{L}(z)e^{\mathbb{L}(z)} = z$ (Lambert function)
- Optimal number of chunks K^* is $\max(1, \lfloor K_0 \rfloor)$ or $\lceil K_0 \rceil$

$$\mathbb{E}_{opt}(T(W)) = K^*\left(e^{\lambda R}\left(\frac{1}{\lambda} + D\right)\right)\left(e^{\lambda(\frac{W}{K^*}+C)} - 1\right)$$

- Can also use Daly's second-order approximation

Checkpointing a parallel job

- p processors $\Rightarrow$ distribution $Exp(\lambda_p)$, where $\lambda_p = p\lambda$
- Use $W(p)$, $C(p)$, $R(p)$ in $\mathbb{E}_{opt}(T(W))$ for a distribution $Exp(\lambda_p = p\lambda)$
- Job types
  - Perfectly parallel jobs: $W(p) = W/p$.
  - Generic parallel jobs: $W(p) = W/p + \delta W$
  - Numerical kernels: $W(p) = W/p + \delta W^{2/3}/\sqrt{p}$
- Checkpoint overhead
  - Proportional overhead: $C(p) = R(p) = \delta V/p = C/p$ (bandwidth of processor network card/link is I/O bottleneck)
  - Constant overhead: $C(p) = R(p) = \delta V = C$ (bandwidth to/from resilient storage system is I/O bottleneck)

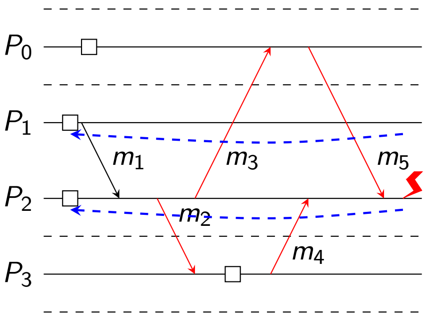
Weibull failure distribution

- No optimality result known
- Heuristic: maximize expected work before next failure
- Dynamic programming algorithms
  - Use a time quantum
  - Trim history of previous failures

Outline

Hierarchical checkpointing

- Clusters of processes
- Coordinated checkpointing protocol within clusters
- Message logging protocols between clusters
- Only processors from failed group need to roll back



☹ Need to log inter-group messages
- Slowdowns failure-free execution
- Increases checkpoint size/time

☺ Faster re-execution with logged messages

Which checkpointing protocol to use?

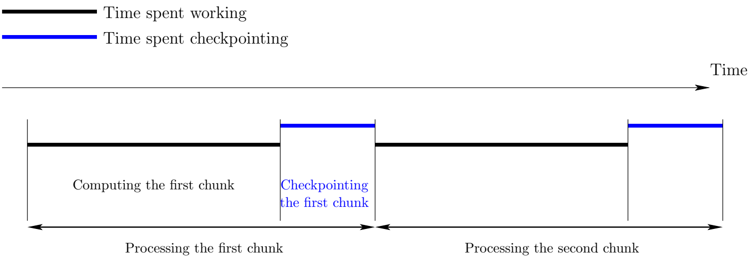
Coordinated checkpointing

- ☺ No risk of cascading rollbacks
- ☺ No need to log messages
- ☹ All processors need to roll back
- ☹ Rumor: May not scale to very large platforms

Hierarchical checkpointing

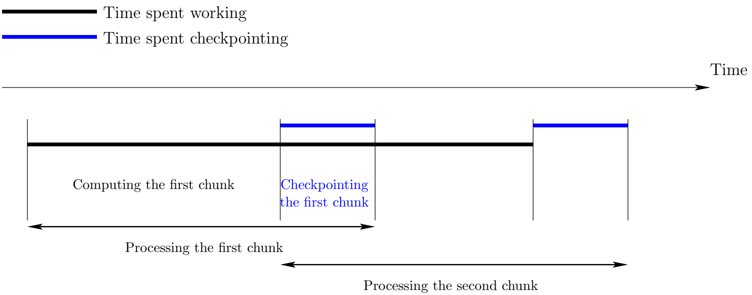
- ☹ Need to log inter-group messages
  - Slowdowns failure-free execution
  - Increases checkpoint size/time
- ☺ Only processors from failed group need to roll back
- ☺ Faster re-execution with logged messages
- ☺ Rumor: Should scale to very large platforms

Blocking vs. non-blocking



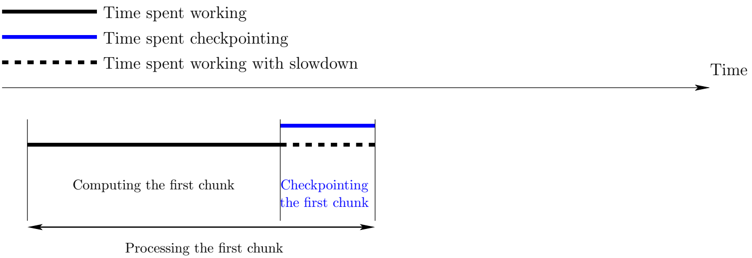
Blocking model: checkpointing blocks all computations

Blocking vs. non-blocking



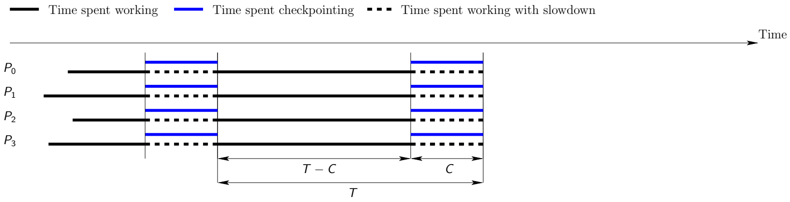
Non-blocking model: checkpointing has no impact on computations (e.g., first copy state to RAM, then copy RAM to disk)

Blocking vs. non-blocking



General model: checkpointing slows computations down: during a checkpoint of duration C, the same amount of computation is done as during a time αC without checkpointing ($0 \leq \alpha \leq 1$)

Waste in fault-free execution

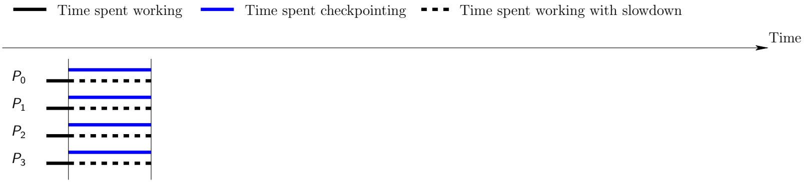


Time elapsed since last checkpoint: T

Amount of computations executed: $\textsc{Work} = (T - C) + \alpha C$

$\textsc{Waste}[FF] = \frac{T - \textsc{Work}}{T}$

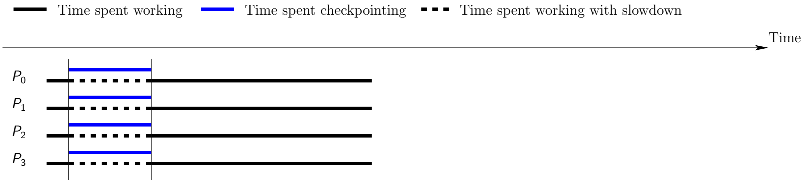
Waste due to failures



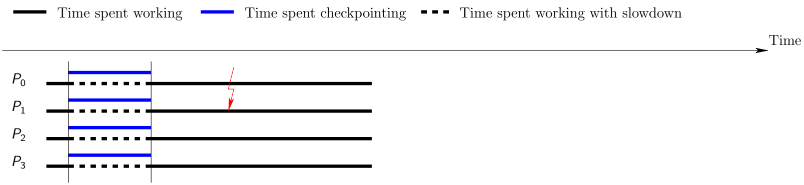
Failure can happen

1. During computation phase
2. During checkpointing phase

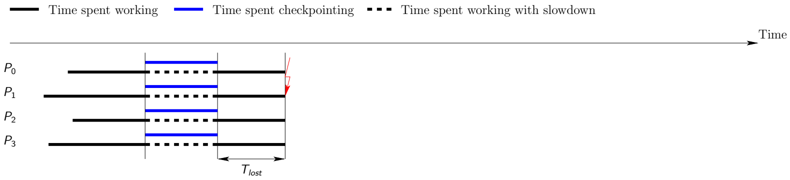
Waste due to failures



Waste due to failures

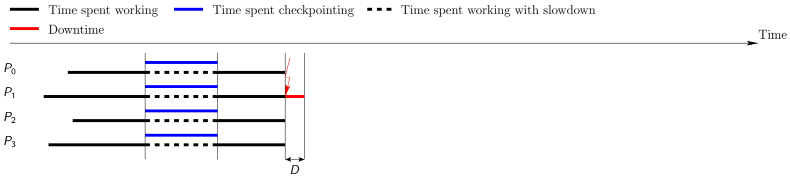


Waste due to failures

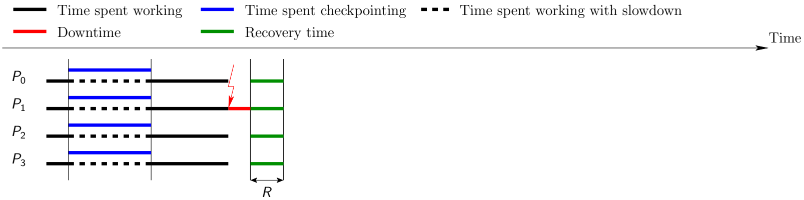


Coordinated checkpointing protocol: when one processor is victim of a failure, all processors lose their work and must roll back to last checkpoint

Waste due to failures in computation phase

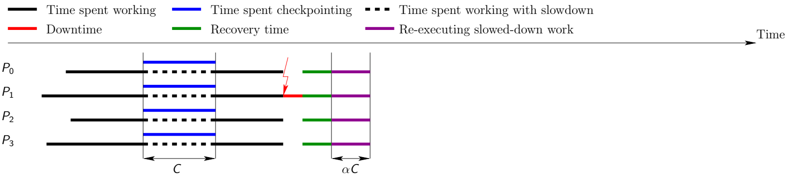


Waste due to failures in computation phase



Coordinated checkpointing protocol: all processors must recover from last checkpoint

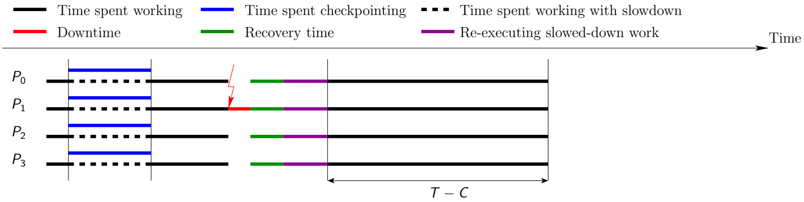
Waste due to failures in computation phase



Redo the work destroyed by the failure, that was done in the checkpointing phase before the computation phase

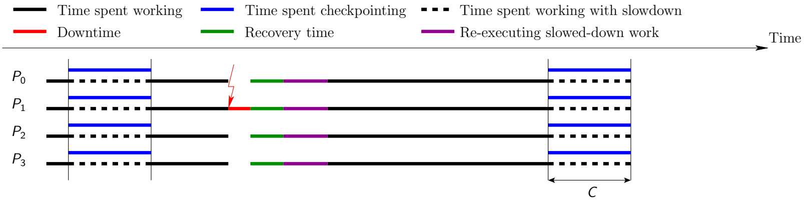
But no checkpoint is taken in parallel, hence this re-execution is faster than the original computation

Waste due to failures in computation phase



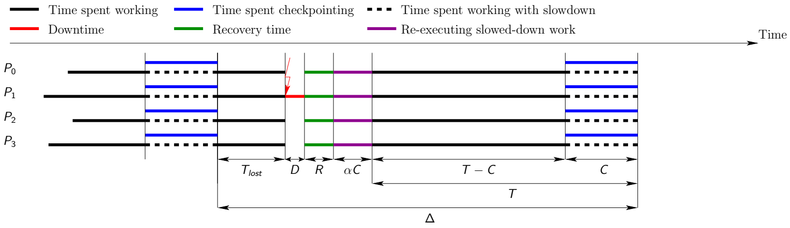
Re-execute the computation phase

Waste due to failures in computation phase



Finally, the checkpointing phase is executed

Total waste

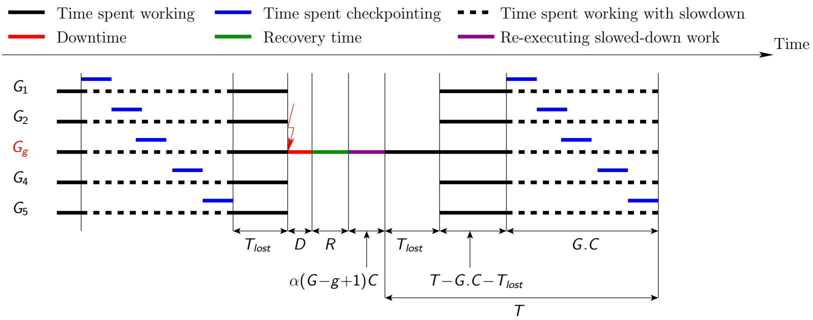


$$\textsc{Waste}[fail] = \frac{1}{\mu}\left(D + R + \alpha C + \frac{T}{2}\right)$$

Optimal period $T_{\text{opt}} = \sqrt{2(1-\alpha)(\mu - (D + R + \alpha C))C}$

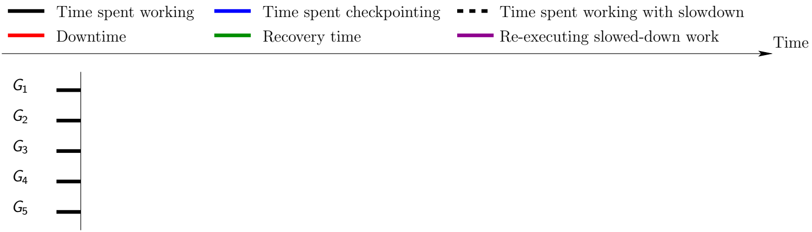
Outline

Hierarchical checkpointing

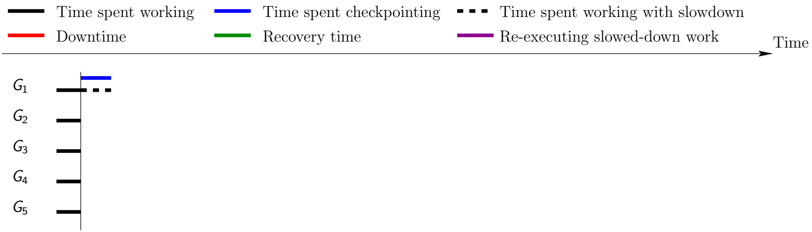


- Processors partitioned into G groups
- Each group includes q processors
- Inside each group: coordinated checkpointing in time $C(q)$
- Inter-group messages are logged

Impact of checkpointing

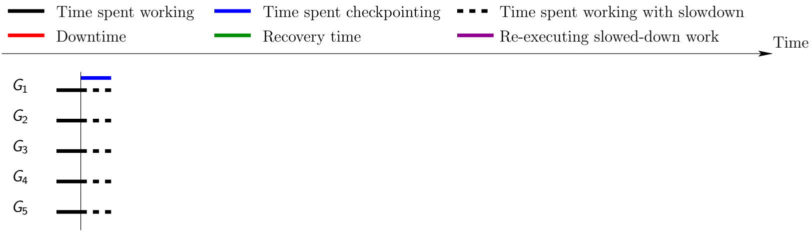


Impact of checkpointing



When a group checkpoints, its own computation speed is slowed-down

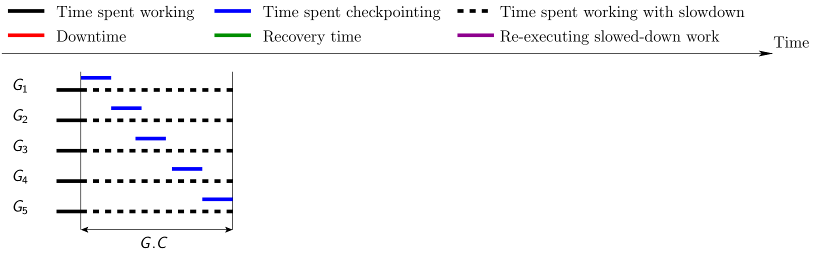
Impact of checkpointing



When a group checkpoints, its own computation speed is slowed-down

This holds for all groups because of the tightly-coupled assumption

Impact of checkpointing

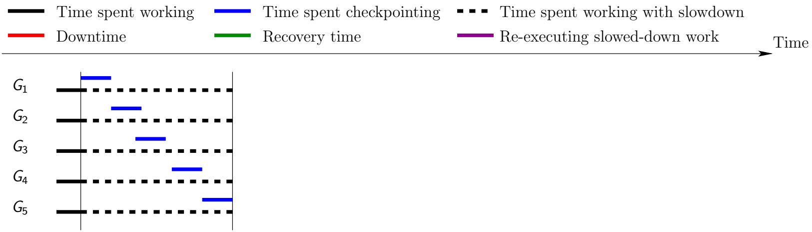


When a group checkpoints, its own computation speed is slowed-down

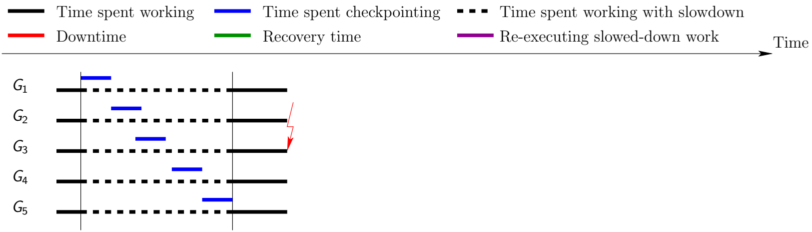
This holds for all groups because of the tightly-coupled assumption

$$\textsc{Waste}[FF] = \frac{T - \textsc{Work}}{T} \text{ where } \textsc{Work} = T - (1-\alpha)GC(q)$$

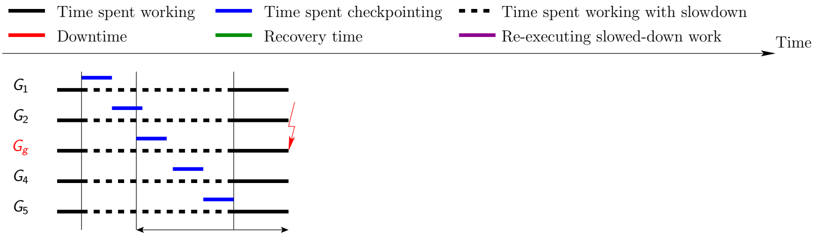
Failure during computation phase



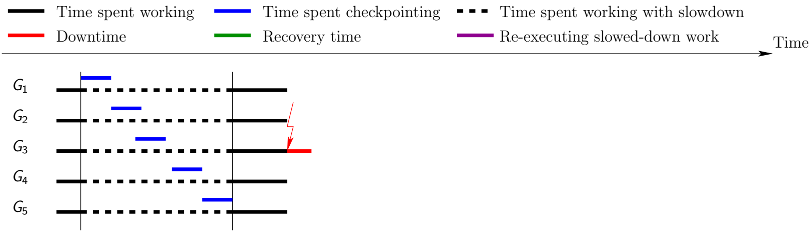
Failure during computation phase



Failure during computation phase

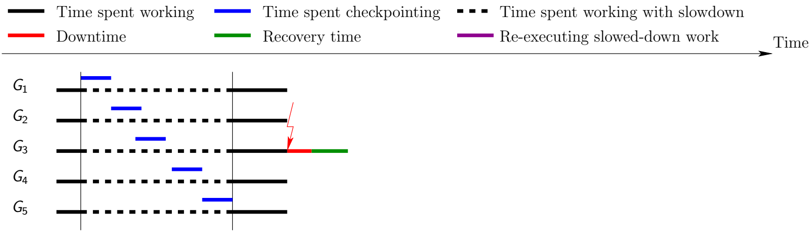


Failure during computation phase



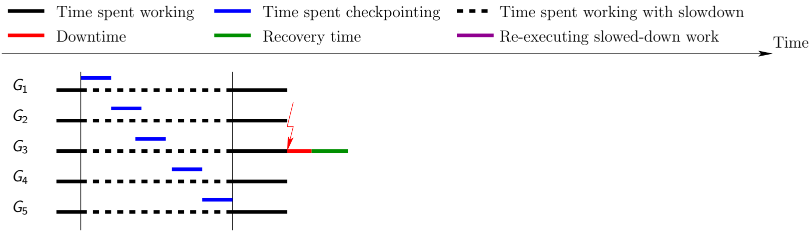
Tightly-coupled model: while one group is in downtime, none can work

Failure during computation phase



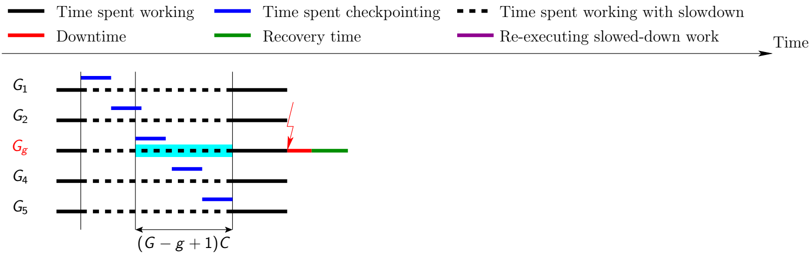
Tightly-coupled model: while one group is in recovery, none can work

Failure during computation phase



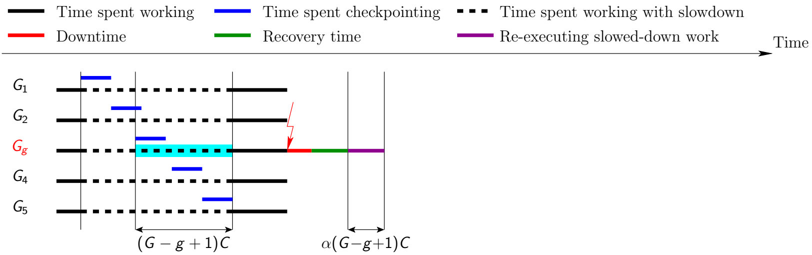
Groups must have completed the same amount of work in between two consecutive checkpoints, independently of the fact that a failure may have happened on the platform in between these checkpoints. Hence, no checkpointing is possible during the rollback.

Failure during computation phase



Redo work done during previous checkpointing phase and that was destroyed by the failure

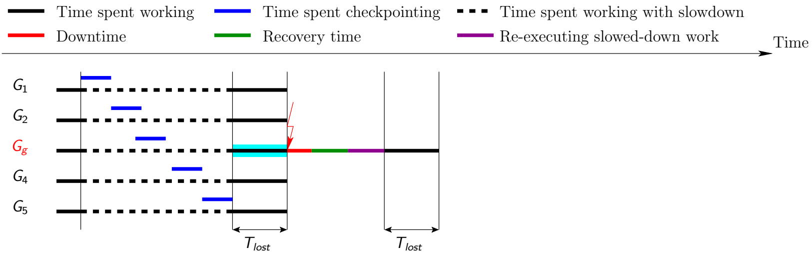
Failure during computation phase



Redo work done during previous checkpointing phase and that was destroyed by the failure

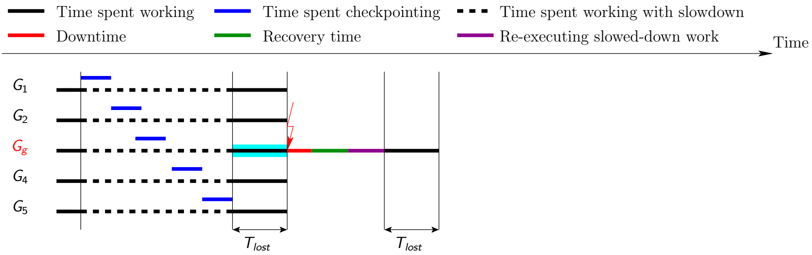
But no checkpoint is taken in parallel, hence this re-computation is faster than the original computation

Failure during computation phase



Redo work done in computation phase and that was destroyed by the failure

Failure during computation phase

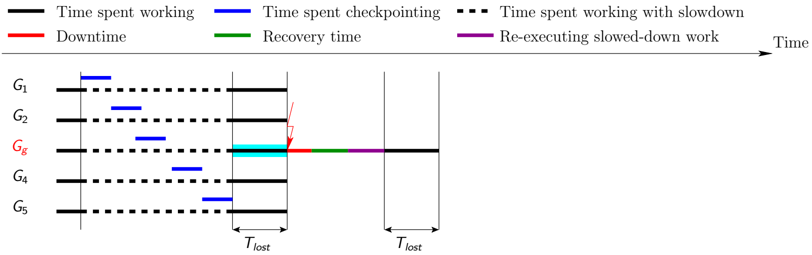


RE-EXEC: $T_{lost} + \alpha(G - g + 1)C$

Expectation: $T_{lost} = \frac{1}{2}(T - G.C)$

Approximated RE-EXEC: $\frac{T - G.C}{2} + \alpha(G - g + 1)C$

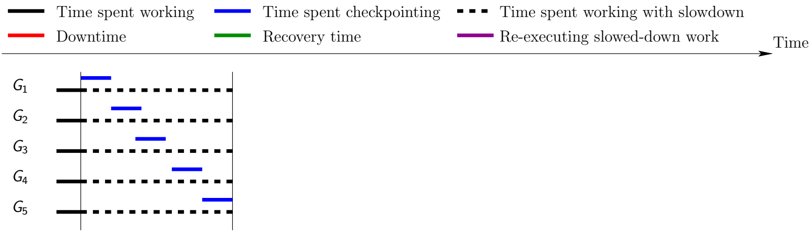
Failure during computation phase



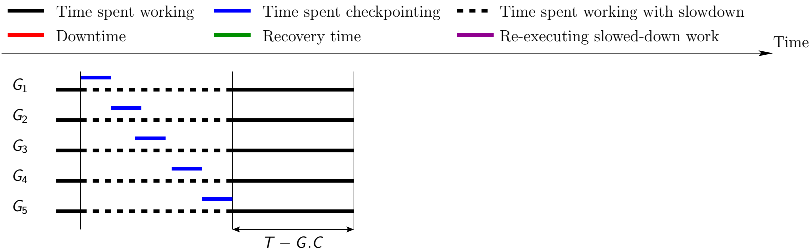
Average approximated RE-EXEC:

$$\frac{1}{G}\sum_{g=1}^{G}\left[\frac{T - G.C(q)}{2} + \alpha(G - g + 1)C(q)\right]$$
$$= \frac{T - G.C(q)}{2} + \alpha\frac{G+1}{2}C(q)$$

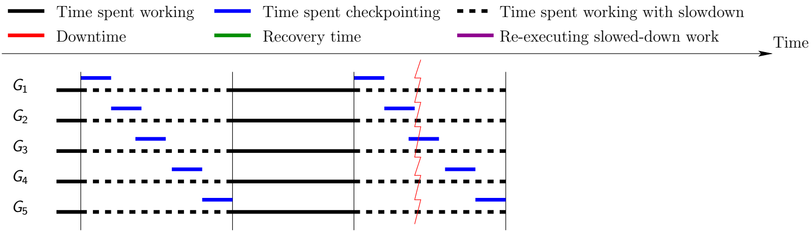
Failure during checkpointing phase



Failure during checkpointing phase



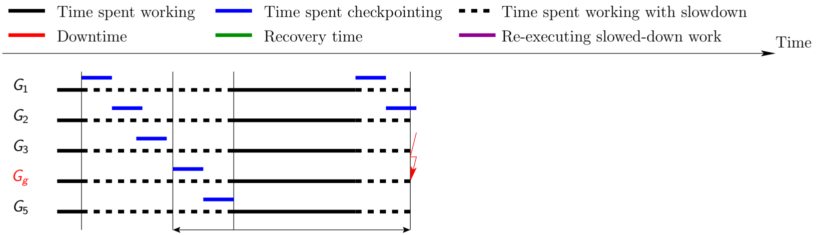
Failure during checkpointing phase



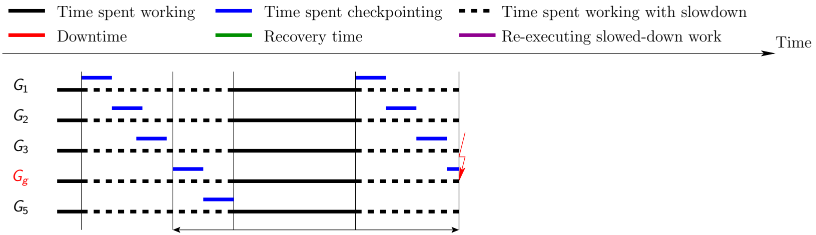
When does the failing group fail?

1. Before starting its own checkpoint
2. While taking its own checkpoint
3. After completing its own checkpoint

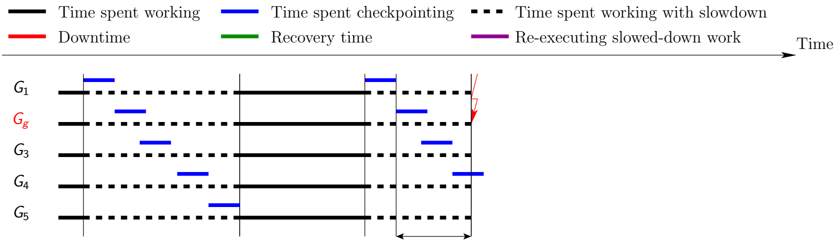
Failure during checkpointing phase: before checkpoint



Failure during checkpointing phase: during checkpoint



Failure during checkpointing phase: after checkpoint



Average waste for failures during checkpointing phase

Average $\textsc{Re-Exec}$ when the failing-group g fails
Overall average $\textsc{Re-Exec}$: $\textsc{Re-Exec}_{ckpt} =$

$$\frac{1}{G}((g-1).\textsc{Re-Exec}_{before\_ckpt} + 1.\textsc{Re-Exec}_{during\_ckpt} + (G-g).\textsc{Re-Exec}_{after\_ckpt})$$

Total waste

$$\textsc{Waste}[FF] = \frac{T - \textsc{Work}}{T} \text{ with } \textsc{Work} = T - (1-\alpha)GC(q)$$

$$\textsc{Waste}[fail] = \frac{1}{\mu}\left(D(q) + R(q) + \textsc{Re-Exec}\right) \text{ with}$$

$$\textsc{Re-Exec} = \frac{T - GC(q)}{T}\textsc{Re-Exec}_{comp} + \frac{GC(q)}{T}\textsc{Re-Exec}_{ckpt}$$

$$\textsc{Waste} = \textsc{Waste}[FF] + \textsc{Waste}[fail] - \textsc{Waste}[FF]\textsc{Waste}[fail]$$

Minimize WASTE subject to:

- $GC(q) \leq T$ (by construction)
- Gets complicated! Use computer algebra software ☹

Outline

Accounting for message logging: Impact on work

- ☹ Logging messages slows down execution:
  $\Rightarrow$ $\textsc{Work}$ becomes $\lambda\textsc{Work}$, where $0 < \lambda < 1$
  Typical value: $\lambda \approx 0.98$
- ☺ Re-execution after a failure is faster:
  $\Rightarrow$ $\textsc{Re-Exec}$ becomes $\frac{\textsc{Re-Exec}}{\rho}$, where $\rho \in [1..2]$
  Typical value: $\rho \approx 1.5$

$$\textsc{Waste}[FF] = \frac{T - \lambda\textsc{Work}}{T}$$

$$\textsc{Waste}[fail] = \frac{1}{\mu}\left(D(q) + R(q) + \frac{\textsc{Re-Exec}}{\rho}\right)$$

Accounting for message logging: Impact on checkpoint size

- Inter-group messages logged continuously
- Checkpoint size increases with amount of work executed before a checkpoint ☹
- $C_0(q)$: Checkpoint size of a group without message logging

$$C(q) = C_0(q)(1 + \beta \textsc{Work}) \Leftrightarrow \beta = \frac{C(q) - C_0(q)}{C_0(q)\textsc{Work}}$$

$$\textsc{Work} = \lambda(T - (1-\alpha)GC(q))$$

$$C(q) = \frac{C_0(q)(1 + \beta\lambda T)}{1 + GC_0(q)\beta\lambda(1-\alpha)}$$

Outline

Three case studies

Coord-IO

Coordinated approach: $C = C_{\text{Mem}} = \frac{\text{Mem}}{\text{b}_{io}}$
where Mem is the memory footprint of the application

Hierarch-IO

Several (large) groups, *I/O-saturated*
$\Rightarrow$ groups checkpoint sequentially

$$C_0(q) = \frac{C_{\text{Mem}}}{G} = \frac{\text{Mem}}{G\text{b}_{io}}$$

Hierarch-Port

Very large number of smaller groups, *port-saturated*
$\Rightarrow$ some groups checkpoint in parallel
Groups of $\text{q}_{\min}$ processors, where $\text{q}_{\min}\text{b}_{port} \geq \text{b}_{io}$

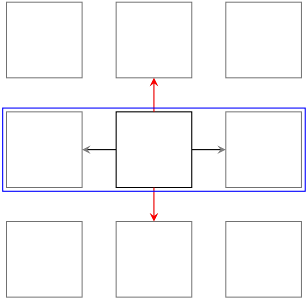
Three applications

1. 2D-stencil
2. Matrix product
3. 3D-Stencil
    - Plane
    - Line

Computing β for 2D-Stencil

$$C(q) = C_0(q) + Logged\_Msg = C_0(q)(1 + \beta \textsc{Work})$$

Real $n \times n$ matrix and $p \times p$ grid
$Work = \frac{9b^2}{s_p}$, $b = n/p$
Each process sends a block to its 4 neighbors



HIERARCH-IO:

- 1 group = 1 grid row
- 2 out of the 4 messages are logged
- $\beta = \frac{Logged\_Msg}{C_0(q)\textsc{Work}} = \frac{2pb}{pb^2(9b^2/s_p)} = \frac{2s_p}{9b^3}$

HIERARCH-PORT:

- β doubles

Three applications: Matrix product

- Three matrices involved: $Mem = 3n^2$, $C_0(q) = 3pb^2$
- Cannon's algorithm: p steps to compute a product
- $Work = \frac{2b^3}{s_p}$, $b = n/p$

HIERARCH-IO:

- 1 group = 1 grid row
- only vertical messages are logged: pb^2
- $\beta = \frac{pb^2}{3pb^2(2b^3/s_p)} = \frac{s_p}{6b^3}$

HIERARCH-PORT:

- β doubles

Three applications: 3D-stencil

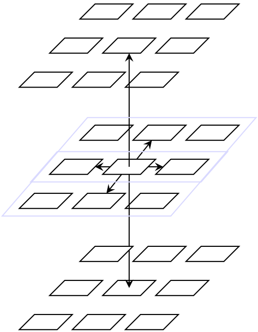
- Real matrix of size $n \times n \times n$ partitioned across a $p \times p \times p$ processor grid
- Each processor holds a cube of size $b = n/p$
- At each iteration:
  - average each matrix element with its 27 closest neighbors
  - exchange the six faces of its cube
- (Parallel) work for one iteration is $\textsc{Work} = \frac{27b^3}{s_p}$

Three hierarchical variants

1. Hierarch-IO-Plane: group = horizontal plane of size p^2: $\beta = \frac{2s_p}{27b^3}$
2. Hierarch-IO-Line: group = horizontal line of size p: $\beta = \frac{4s_p}{27b^3}$
3. Hierarch-Port: groups of size q_{min}: $\beta = \frac{6s_p}{27b^3}$

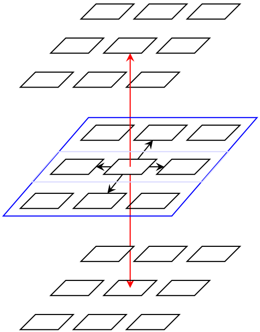
3D-stencil illustration

- 3D-Plane: Vertical messages are logged
- 3D-Line: Twice as many messages are logged



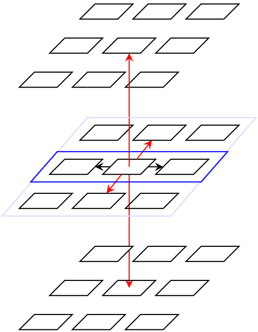
3D-stencil illustration

- 3D-Plane: Vertical messages are logged
- 3D-Line: Twice as many messages are logged



3D-stencil illustration

- 3D-Plane: Vertical messages are logged
- 3D-Line: Twice as many messages are logged



Four platforms: basic characteristics

Name	Number of cores	Number of processors p_{total}	Number of cores per processor	Memory per processor	I/O Network Bandwidth (b_{io}) Read	Write	I/O Bandwidth (b_{port}) Read/Write per processor
Titan	299,008	16,688	16	32GB	300GB/s	300GB/s	20GB/s
K-Computer	705,024	88,128	8	16GB	150GB/s	96GB/s	20GB/s
Exascale-Slim	1,000,000,000	1,000,000	1,000	64GB	1TB/s	1TB/s	200GB/s
Exascale-Fat	1,000,000,000	100,000	10,000	640GB	1TB/s	1TB/s	400GB/s

Name	Scenario	G ($C(q)$)	β for 2D-Stencil	β for Matrix-Product
Titan	Coord-IO	1 (2,048s)	/	/
	Hierarch-IO	136 (15s)	0.0001098	0.0004280
	Hierarch-Port	1,246 (1.6s)	0.0002196	0.0008561
K-Computer	Coord-IO	1 (14,688s)	/	/
	Hierarch-IO	296 (50s)	0.0002858	0.001113
	Hierarch-Port	17,626 (0.83s)	0.0005716	0.002227
Exascale-Slim	Coord-IO	1 (64,000s)	/	/
	Hierarch-IO	1,000 (64s)	0.0002599	0.001013
	Hierarch-Port	200,0000 (0.32s)	0.0005199	0.002026
Exascale-Fat	Coord-IO	1 (64,000s)	/	/
	Hierarch-IO	316 (217s)	0.00008220	0.0003203
	Hierarch-Port	33,3333 (1.92s)	0.00016440	0.0006407

Four platforms: 3D-Stencil

Name	Scenario	G	β for 3D-Stencil
Titan	Coord-IO	1	/
	Hierarch-IO-Plane	26	0.001476
	Hierarch-IO-Line	675	0.002952
	Hierarch-Port	1,246	0.004428
K-Computer	Coord-IO	1	/
	Hierarch-IO-Plane	44	0.003422
	Hierarch-IO-Line	1,936	0.006844
	Hierarch-Port	17,626	0.010266
Exascale-Slim	Coord-IO	1	/
	Hierarch-IO-Plane	100	0.003952
	Hierarch-IO-Line	10,000	0.007904
	Hierarch-Port	200,000	0.011856
Exascale-Fat	Coord-IO	1	/
	Hierarch-IO-Plane	46	0.001834
	Hierarch-IO-Line	2,116	0.003668
	Hierarch-Port	33,333	0.005502

Outline

Simulation parameters

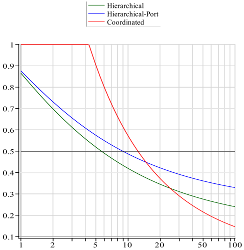
- Failure distribution: Weibull, $k = 0.7$
- Failure free execution on each process: 4 days
- Time-out: 1 year
- No assumption on failures
- $\alpha = 0.3$, $\lambda = 0.98$, $\rho = 1.5$
- Each point: average over 20 randomly generated instances
- Computed period and best period:

$\rightarrow$ Generate 480 periods in the neighborhood of the period from the model

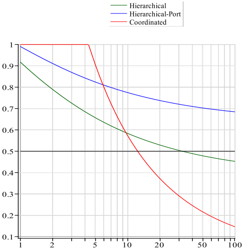
$\rightarrow$ Numerically evaluate the best one through simulations

Plotting formulas – Platform: Titan

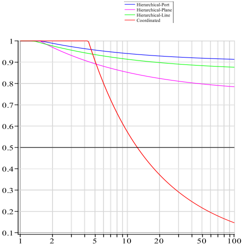
Stencil 2D



Matrix product

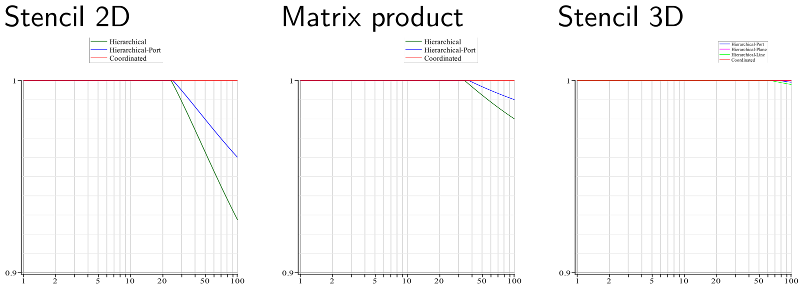


Stencil 3D



Waste as a function of processor MTBF μ_{ind}

Platform: K-Computer



Waste as a function of processor MTBF μ_{ind}

Plotting formulas – Platform: Exascale

$\textsc{Waste} = 1$ for all scenarios!!!

Plotting formulas – Platform: Exascale

WASTE = 1 for all scenarios!!!

Goodbye Exascale?!

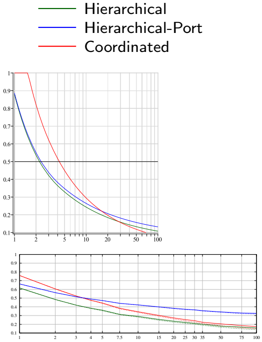
Checkpoint time

Name	C
K-Computer	14,688s
Exascale-Slim	64,000s
Exascale-Fat	64,000s

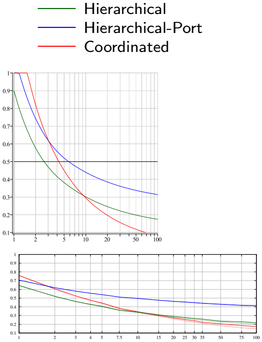
- Large time to dump the memory
- Using $1\%C$
  - faster I/O and storage (two-level checkpoint, SSD, ...)
  - smaller amount of memory written
- Comparing with $0.1\%C$ for exascale platforms

Platform: KComputer with $C = C/100$

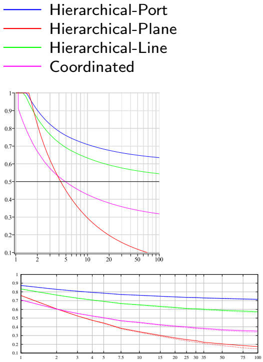
- Solid line: Computed period
- Dotted line: Best period



2D-Stencil



Matrix Product

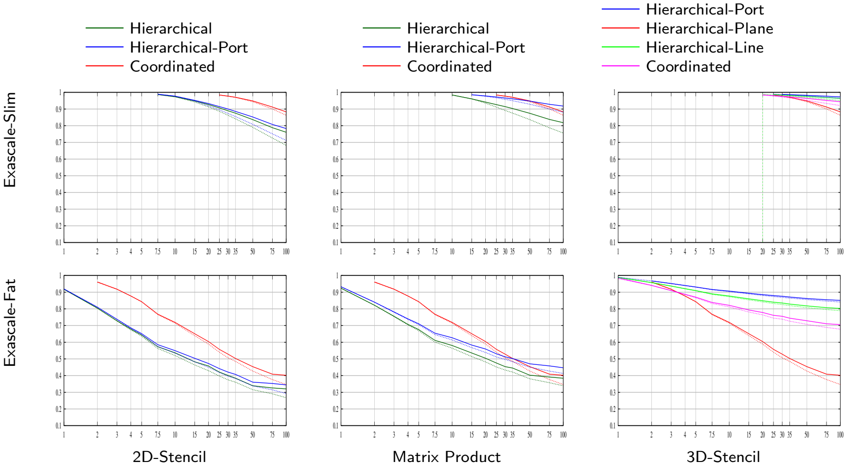


3D-Stencil

Waste as a function of processor MTBF μ_{ind}, $C = C/100$

Platform: Exascale with $C = C/100$

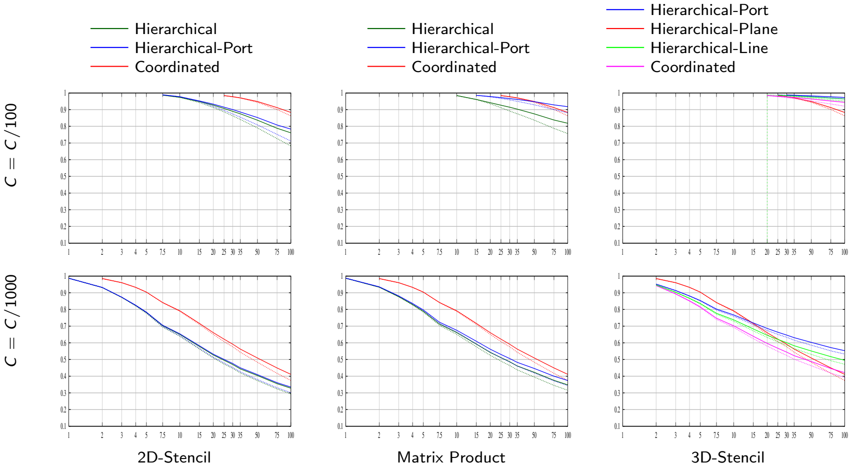
- Solid line: Computed period
- Dotted line: Best period



Waste as a function of processor MTBF μ_{ind}, $C = C/100$

Checkpoint impact: Exascale Slim

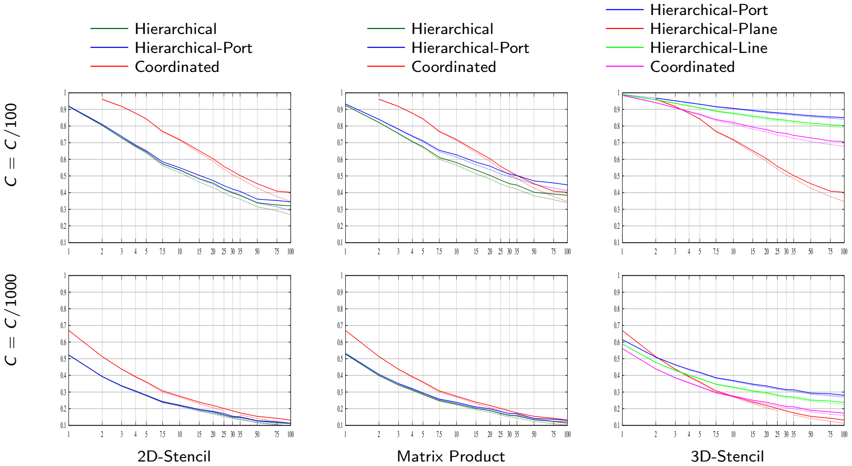
- Solid line: Computed period
- Dotted line: Best period



Waste as a function of processor MTBF μ with checkpoint variation

Checkpoint impact: Exascale Fat

- Solid line: Computed period
- Dotted line: Best period



Waste as a function of processor MTBF μ with checkpoint variation

Conclusion

- Hierarchical protocols very sensitive to message logging: direct relationship between β and the observed waste
- Hierarchical protocols better for small MTBFs: more suitable for failure-prone platforms
- Struggle when communication intensity increases (3D-stencil), but limited waste in all other cases
- The faster the checkpointing time, the smaller the waste
- Exascale-Fat better than Exascale-Slim: fewer processors, hence larger MTBF!
- Simulations with random trace of errors: the model computes near-optimal checkpointing periods

- What could we further add: (partial) replication, prediction, energy

Outline

Motivation

- Checkpoint transfer and storage
  $\Rightarrow$ critical issues of rollback/recovery protocols

- Stable storage: high cost

- Distributed in-memory storage:
  - Store checkpoints in local memory $\Rightarrow$ no centralized storage
    ☺ Much better scalability
  - Replicate checkpoints $\Rightarrow$ application survives single failure
    ☹ Still, risk of fatal failure in some (unlikely) scenarios

Outline

Double checkpoint algorithm

- Platform nodes partitioned into pairs
- Each node in a pair exchanges its checkpoint with its *buddy*
- Each node saves two checkpoints:
  - one locally: storing its own data
  - one remotely: receiving and storing its buddy's data

Two algorithms

- blocking version by Zheng, Shi and Kalé
- non-blocking version by Ni, Meneses and Kalé