Optimal checkpointing periods with fail-stop and silent errors

Anne Benoit

ENS Lyon

Anne.Benoit@ens-lyon.fr

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

3rd JLESC Summer School
June 30, 2016

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 ⇒ Shorter MTBF (Mean Time Between Failures)

Exascale platforms

- Hierarchica
  - 10^5 or 10^6 node
  - Each node equipped wi 10^4 10^3 cores

- Failure-prone

MTBF – o node	1 year	10 ars	120 years
MTBF – latform of 10^6 nodes	30sec	5m	1h

Mor odes = een ilures)

Exascale

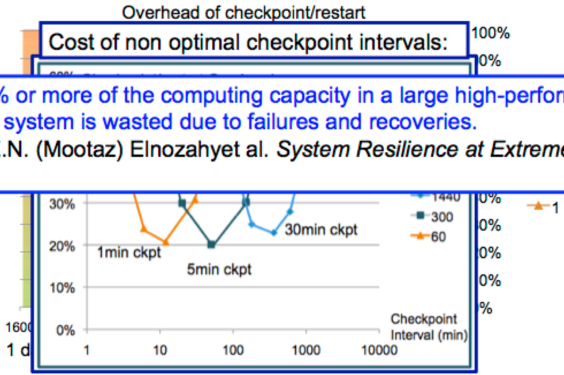
$\neq$ Petascale $\times 1000$

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

Also an issue at Petascale

Joint Laboratory for Petascale Computation — INRIA — NCSA

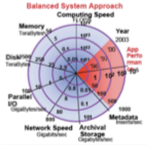
Fault tolerance becomes critical at Petascale (MTTI <= 1day)
Poor fault tolerance design may lead to huge overhead



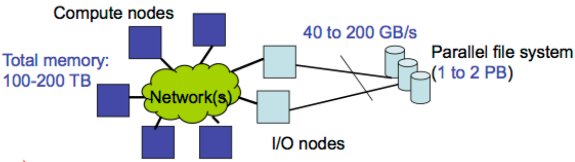
Today, 20% or more of the computing capacity in a large high-performance computing system is wasted due to failures and recoveries.
Dr. E.N. (Mootaz) Elnozahyet al. *System Resilience at Extreme Scale, DARPA*

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

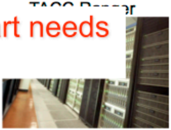
Classic approach for FT: Checkpoint-Restart



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

A few definitions

- Many types of faults: software error, hardware malfunction, memory corruption
- Many possible behaviors: silent, transient, unrecoverable
- Restrict to faults that lead to application failures
- This includes all hardware faults, and some software ones
- Will use terms *fault* and *failure* interchangeably
- Silent errors (SDC) will be addressed later in the course

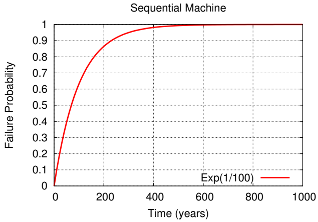
- First question: quantify the rate or frequency at which these faults strike!

A few definitions

- Many types of faults: software error, hardware malfunction, memory corruption
- Many possible behaviors: silent, transient, unrecoverable
- Restrict to faults that lead to application failures
- This includes all hardware faults, and some software ones
- Will use terms *fault* and *failure* interchangeably
- Silent errors (SDC) will be addressed later in the course

- First question: quantify the rate or frequency at which these faults strike!

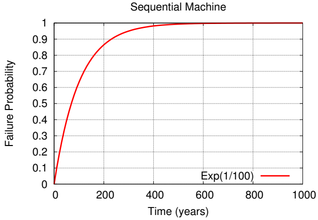
Exponential failure distributions



Exp(λ): Exponential distribution law of parameter λ:

- Probability density function (pdf): $f(t) = \lambda e^{-\lambda t} dt$ for $t \geq 0$
- Cumulative distribution function (cdf): $F(t) = 1 - e^{-\lambda t}$
- Mean: $\mu = \frac{1}{\lambda}$

Exponential failure distributions



X random variable for $Exp(\lambda)$ failure inter-arrival times:

- $\mathbb{P}(X \leq t) = 1 - e^{-\lambda t} dt$ (by definition)
- Memoryless property: $\mathbb{P}(X \geq t + s \,|\, X \geq s) = \mathbb{P}(X \geq t)$ (for all $t, s \geq 0$): at any instant, time to next failure does not depend on time elapsed since last failure
- Mean Time Between Failures (MTBF) $\mu = \mathbb{E}(X) = \frac{1}{\lambda}$

With several processors

- Rebooting only faulty processor
- Platform failure distribution
  $\Rightarrow$ superposition of p IID processor distributions
  $\Rightarrow$ IID only for Exponential
- Define μ_p by

$$\lim_{F \to +\infty} \frac{n(F)}{F} = \frac{1}{\mu_p}$$

$n(F)$ = number of platform failures until time F is exceeded

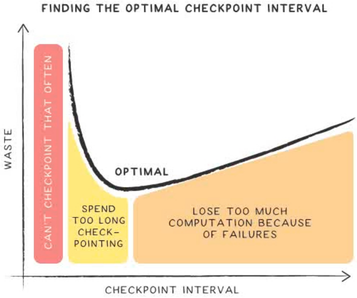
Theorem: $\mu_p = \dfrac{\mu}{p}$ for arbitrary distributions

Summary for the road

- MTBF key parameter and $\mu_p = \frac{\mu}{p}$ ☺
- Exponential distribution OK for most purposes ☺
- Assume failure independence while not (completely) true ☹

General purpose approach

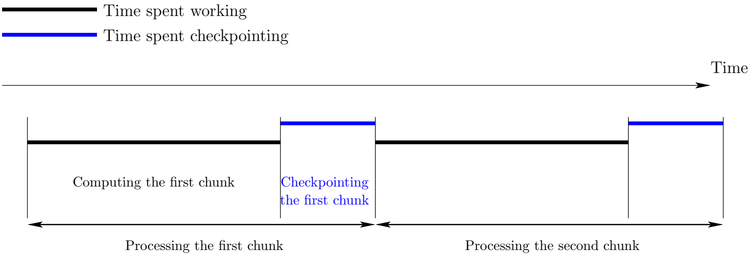
Periodic checkpointing, rollback and recovery



Outline

Outline

Checkpointing cost



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

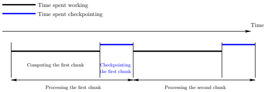
Framework

- Periodic checkpointing policy of period T
- Independent and identically distributed (IID) failures
- Applies to a single processor with MTBF $\mu = \mu_{ind}$
- Applies to a platform with p processors with MTBF $\mu = \frac{\mu_{ind}}{p}$
  - coordinated checkpointing
  - tightly-coupled application
  - progress $\Leftrightarrow$ all processors available

$\Rightarrow$ platform = single (powerful, unreliable) processor ☺

Waste: fraction of time not spent for useful computations

Waste in fault-free execution



- $\text{TIME}_{\text{base}}$: application base time
- TIME_{FF}: with periodic checkpoints but failure-free

$$\text{TIME}_{\text{FF}} = \text{TIME}_{\text{base}} + \#checkpoints \times C$$

$$\#checkpoints = \left\lceil \frac{\text{TIME}_{\text{base}}}{T - C} \right\rceil \approx \frac{\text{TIME}_{\text{base}}}{T - C} \text{ (valid for large jobs)}$$

$$\text{WASTE}[FF] = \frac{\text{TIME}_{\text{FF}} - \text{TIME}_{\text{base}}}{\text{TIME}_{\text{FF}}} = \frac{C}{T}$$

Waste due to failures

- $\textsc{Time}_{\text{base}}$: application base time
- $\textsc{Time}_{\text{FF}}$: with periodic checkpoints but failure-free
- $\textsc{Time}_{\text{final}}$: expectation of time with failures

$$\textsc{Time}_{\text{final}} = \textsc{Time}_{\text{FF}} + N_{faults} \times T_{\text{lost}}$$

N_{faults}: number of failures during execution
T_{lost}: average time lost per failure

$$N_{faults} = \frac{\textsc{Time}_{\text{final}}}{\mu}$$

T_{lost}?

Waste due to failures

- $\text{TIME}_{\text{base}}$: application base time
- TIME_{FF}: with periodic checkpoints but failure-free
- $\text{TIME}_{\text{final}}$: expectation of time with failures

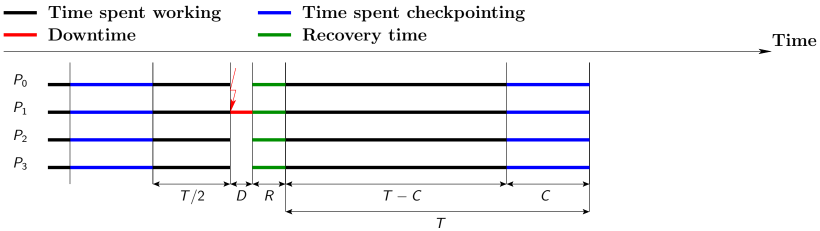
$$\text{TIME}_{\text{final}} = \text{TIME}_{\text{FF}} + N_{faults} \times T_{\text{lost}}$$

N_{faults}: number of failures during execution
T_{lost}: average time lost per failure

$$N_{faults} = \frac{\text{TIME}_{\text{final}}}{\mu}$$

T_{lost}?

Computing T_{lost}



$$T_{\text{lost}} = D + R + \frac{T}{2}$$

Rationale

⇒ Instants when periods begin and failures strike are independent

⇒ Approximation used for all distribution laws

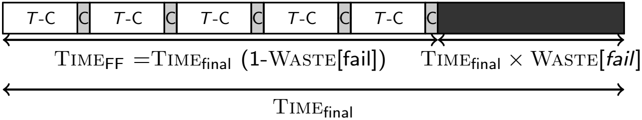
⇒ Exact for Exponential and uniform distributions

Waste due to failures

$$\textsc{Time}_{\textsf{final}} = \textsc{Time}_{\textsf{FF}} + N_{faults} \times T_{\textsf{lost}}$$

$$\textsc{Waste}[fail] = \frac{\textsc{Time}_{\textsf{final}} - \textsc{Time}_{\textsf{FF}}}{\textsc{Time}_{\textsf{final}}} = \frac{1}{\mu}\left(D + R + \frac{T}{2}\right)$$

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 $T = \sqrt{\frac{u}{w}}$

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

Validity of the approach (1/3)

Technicalities

- $\mathbb{E}\left(N_{faults}\right) = \frac{\text{TIME}_{\text{final}}}{\mu}$ and $\mathbb{E}\left(T_{\text{lost}}\right) = D + R + \frac{T}{2}$
  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

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

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	$\Rightarrow \textsc{Waste}[opt] = 100\%$

Lesson learnt for fail-stop failures

(… so) Secret data

- Tsuba… 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	$\Rightarrow$ W…TE[opt] $= 17\%$
Scale … 10:	$C = 20$ min	$\mu = 2.4$ hrs	$\Rightarrow$ WAS…[opt] $= 53\%$
Sca… by 100:	$C = 20$ min	$\mu = 0.24$ hrs	$\Rightarrow$ WASTE[…pt] $= 100\%$

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	$\Rightarrow$ 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$[opt] = 100\%$

Exponential failure distribution

How to 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? How to extend this result for sequential and parallel jobs?

Attend the hands-on session at 14.45: "Mathematical exercises on Daly and extensions"!

Outline

Which checkpointing protocol to use?

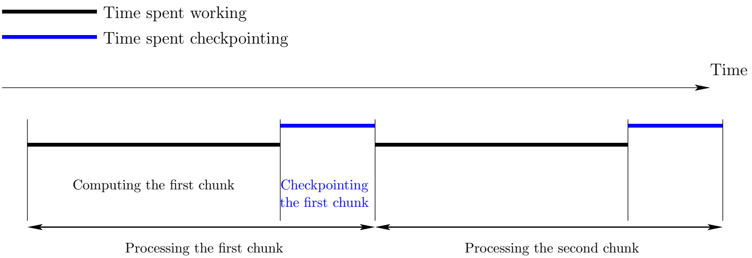
Coordinated checkpointing

- ☺ 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

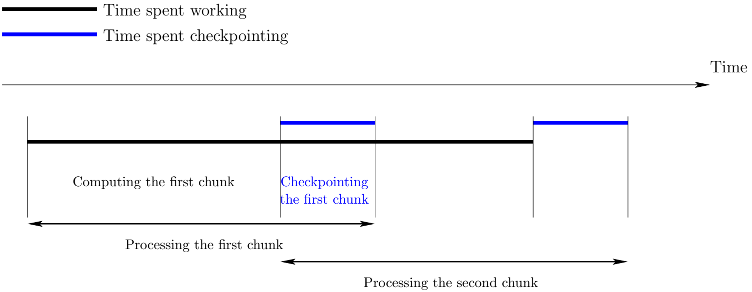
- ☹ Need to log inter-group messages
  - Slowdowns failure-free execution
  - Increases checkpoint size/time
- ☺ Only processors from failed group need to roll back
- ☺ Faster re-execution with logged messages
- ☺ Rumor: Should scale to very large platforms

Blocking vs. non-blocking



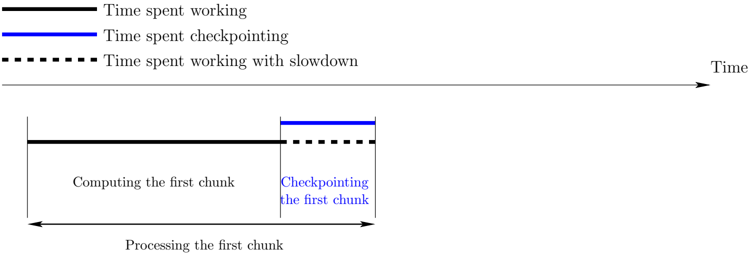
Blocking model: checkpointing blocks all computations

Blocking vs. non-blocking



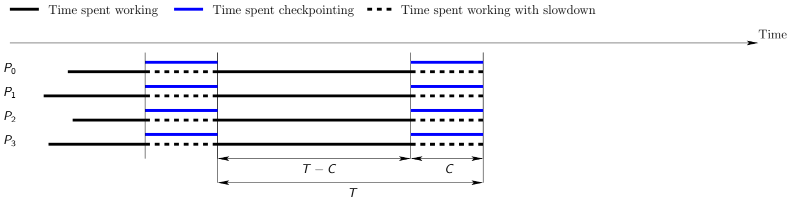
Non-blocking model: checkpointing has no impact on computations (e.g., first copy state to RAM, then copy RAM to disk)

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

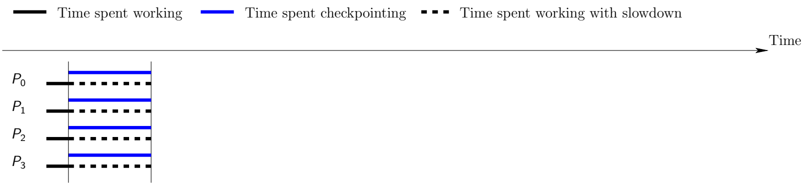


Time elapsed since last checkpoint: T

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

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

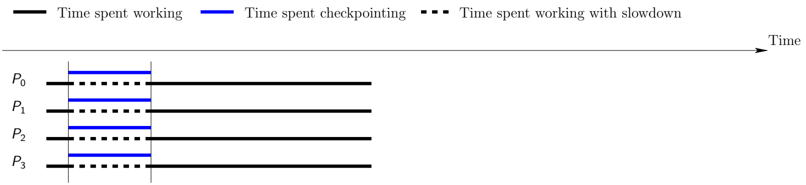
Waste due to failures



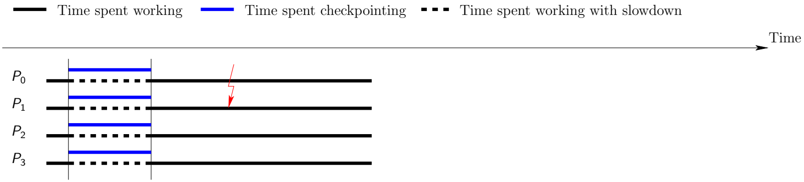
Failure can happen

1. During computation phase
2. During checkpointing phase

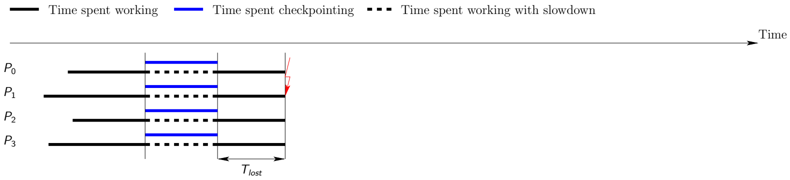
Waste due to failures



Waste due to failures

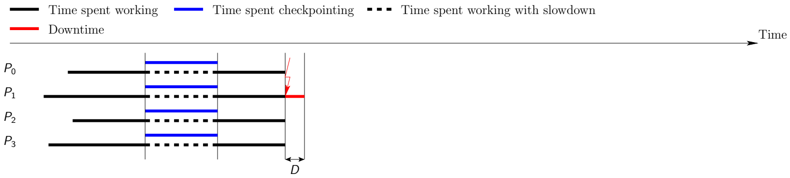


Waste due to failures

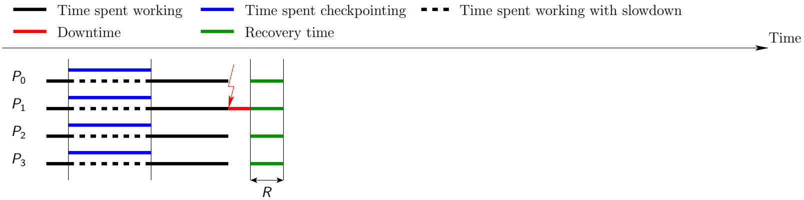


Coordinated checkpointing protocol: when one processor is victim of a failure, all processors lose their work and must roll back to last checkpoint

Waste due to failures in computation phase

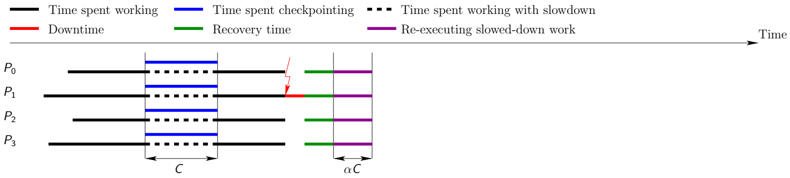


Waste due to failures in computation phase



Coordinated checkpointing protocol: all processors must recover from last checkpoint

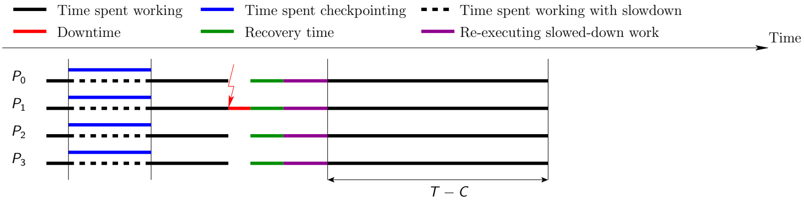
Waste due to failures in computation phase



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

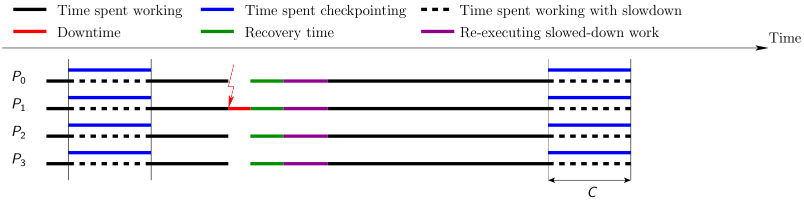
But no checkpoint is taken in parallel, hence this re-execution is faster than the original computation

Waste due to failures in computation phase



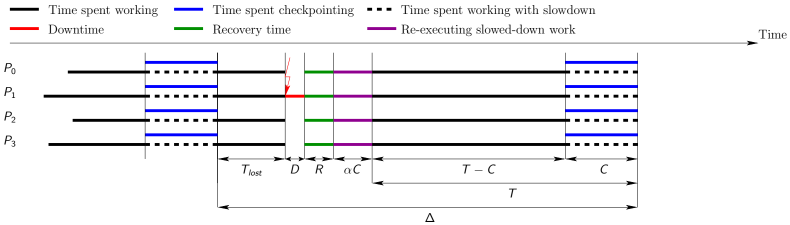
Re-execute the computation phase

Waste due to failures in computation phase



Finally, the checkpointing phase is executed

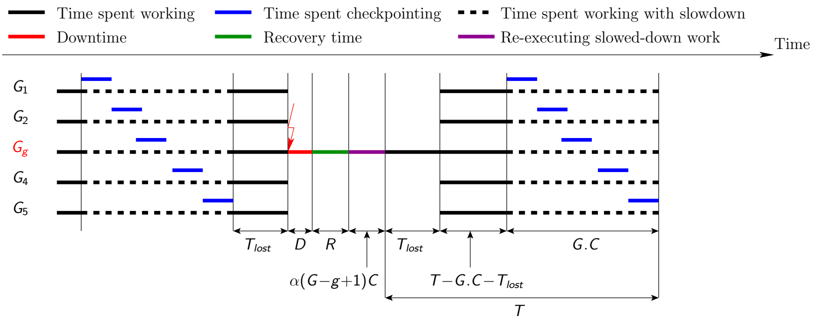
Total waste



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

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

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

Total waste

$$\text{WASTE}[FF] = \frac{T - \text{WORK}}{T} \text{ with } \text{WORK} = T - (1-\alpha)GC(q)$$

$$\text{WASTE}[fail] = \frac{1}{\mu}\left(D(q) + R(q) + \text{RE-EXEC}\right) \text{ with}$$

$$\text{RE-EXEC} = \frac{T - GC(q)}{T}\text{RE-EXEC}_{comp} + \frac{GC(q)}{T}\text{RE-EXEC}_{ckpt}$$

$$\text{WASTE} = \text{WASTE}[FF] + \text{WASTE}[fail] - \text{WASTE}[FF]\text{WASTE}[fail]$$

Minimize WASTE subject to:

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

Conclusion

- Hierarchical protocols better for small MTBFs: more suitable for failure-prone platforms
- Struggle when communication intensity increases, but limited waste in all other cases
- The faster the checkpointing time, the smaller the waste

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

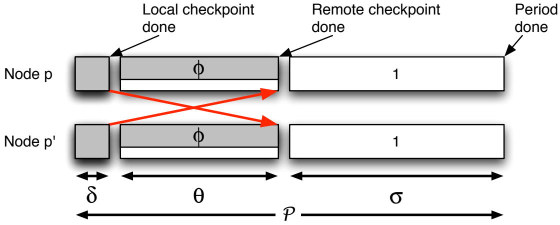
Double checkpointing algorithm

- 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

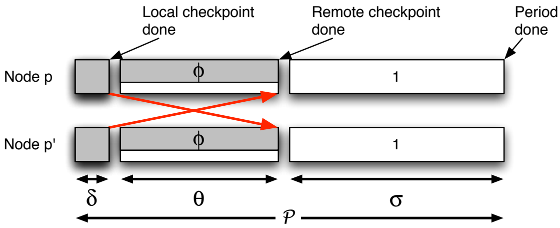
- blocking version by Zheng, Shi and Kalé
- non-blocking version by Ni, Meneses and Kalé

Non-blocking checkpoint algorithm



- Checkpoints taken periodically, with period $P = \delta + \theta + \sigma$
- Phase 1, length δ: local checkpoint, blocking mode. No work
- Phase 2, length θ: remote checkpoint. Overhead ϕ
- Phase 3, length σ: application at full speed 1

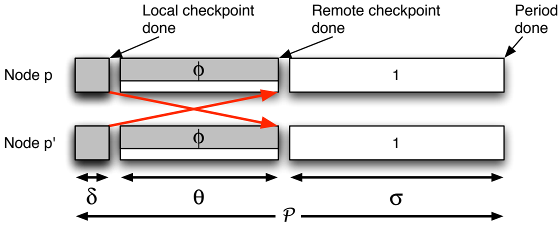
Non-blocking checkpoint algorithm



Work in failure-free period:

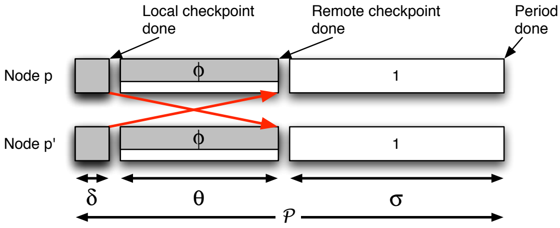
$$W = (\theta - \phi) + \sigma = P - \delta - \phi$$

Cost of overlap



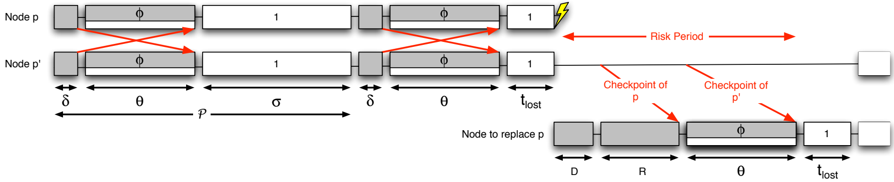
- Overlap computations and checkpoint file exchanges
- Large θ
  $\Rightarrow$ more flexibility to hide cost of file exchange
  $\Rightarrow$ smaller overhead ϕ

Cost of overlap



- $\theta = \theta_{\text{min}}$: fastest communication, fully blocking $\Rightarrow \phi = \theta_{\text{min}}$
- $\theta = \theta_{\text{max}}$: full overlap with computation $\Rightarrow \phi = 0$
- Linear interpolation $\theta(\phi) = \theta_{\text{min}} + \alpha(\theta_{\text{min}} - \phi)$
  - $\phi = 0$ for $\theta = \theta_{\text{max}} = (1 + \alpha)\theta_{\text{min}}$
  - α: rate of overhead decrease w.r.t. communication length

Assessing the risk



- After failure: downtime D and recovery from buddy node
- Two checkpoint files lost, must be re-sent to faulty processor
  1. Checkpoint of faulty node, needed for recovery
     $\Rightarrow$ sent as fast as possible, in time $R = \theta_{\min}$
  2. Checkpoint of buddy node, needed in case buddy fails later on
     $\Rightarrow$ **??**
- Application at risk until complete reception of both messages

Checkpoint of buddy node

Scenario DOUBLENBL

- File sent at same speed as in regular mode, in time $\theta(\phi)$
- Overhead ϕ
- Favors performance, at the price of higher risk

Scenario DOUBLEBOF

- File sent as fast as possible, in time $\theta_{\text{min}} = R$
- Overhead R
- Favors risk reduction, at the price of higher overhead

Computing the waste? Hands-on session at 14:45!

Conclusion

Double checkpointing

- DoubleBoF reduces risk duration, at the cost of increasing failure overhead
- Parameter α for transfer cost overlap
- Unified model for performance/risk bi-criteria assessment

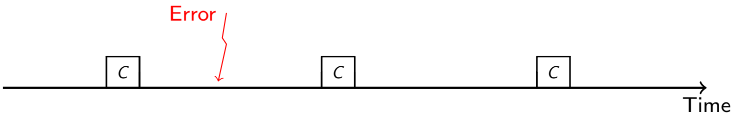
Triple checkpointing

- Save checkpoint on two remote processes instead of one, without much more memory or storage requirements
- Excellent success probability, almost no failure-free overhead
- Assessment of performance and risk factors using unified mode
- Realistic scenarios conclude to superiority of Triple

Outline

General-purpose approach

Periodic checkpointing, rollback and recovery:



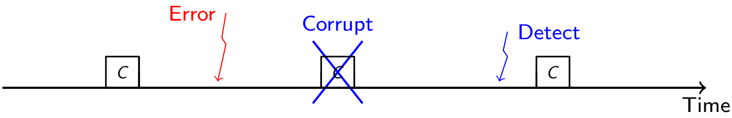
- Works fine for fail-stop errors
- Detection latency in silent errors $\Rightarrow$ risk of saving corrupted checkpoint(s)

Maintaining multiple checkpoints (*Lu, Zheng and Chien, 2013*)

- Requires more stable storage
- Which checkpoint to roll back to?
- Critical failure when all live checkpoints are invalid

General-purpose approach

Periodic checkpointing, rollback and recovery:



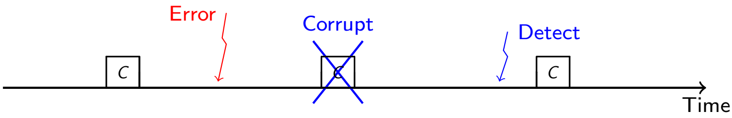
- Works fine for fail-stop errors
- Detection latency in silent errors ⇒ risk of saving corrupted checkpoint(s)

Maintaining multiple checkpoints (*Lu, Zheng and Chien, 2013*)

- Requires more stable storage
- Which checkpoint to roll back to?
- Critical failure when all live checkpoints are invalid

General-purpose approach

Periodic checkpointing, rollback and recovery:



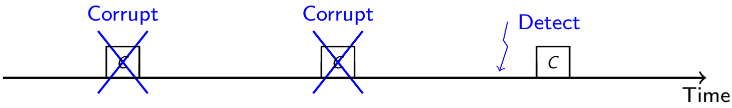
- Works fine for fail-stop errors
- Detection latency in silent errors ⇒ risk of saving corrupted checkpoint(s)

Maintaining multiple checkpoints (*Lu, Zheng and Chien, 2013*)

- Requires more stable storage
- Which checkpoint to roll back to?
- Critical failure when all live checkpoints are invalid

General-purpose approach

Periodic checkpointing, rollback and recovery:



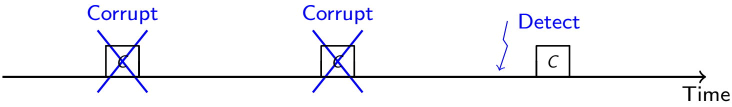
- Works fine for fail-stop errors
- Detection latency in silent errors ⇒ risk of saving corrupted checkpoint(s)

Maintaining multiple checkpoints (*Lu, Zheng and Chien, 2013*)

- Requires more stable storage
- Which checkpoint to roll back to?
- Critical failure when all live checkpoints are invalid

General-purpose approach

Periodic checkpointing, rollback and recovery:



- Works fine for fail-stop errors
- Detection latency in silent errors ⇒ risk of saving corrupted checkpoint(s)

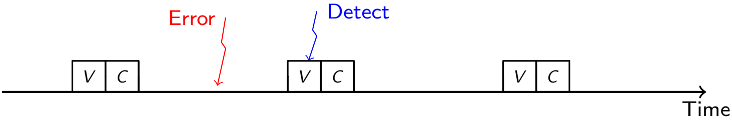
Maintaining multiple checkpoints (*Lu, Zheng and Chien, 2013*)

- Requires more stable storage
- Which checkpoint to roll back to?
- Critical failure when all live checkpoints are invalid

Need to know when silent error occurred

Coping with silent errors

Couple checkpointing with verification:



- Before each checkpoint, run some verification mechanism or error detection test
- Silent error, if any, is detected by verification ⇒ need to maintain only one checkpoint, which is always valid ☺

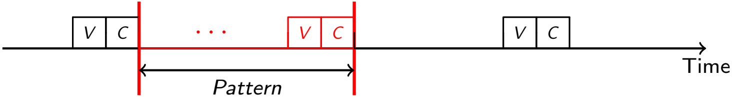
Models and objective

Resilience parameters

- C: Cost of checkpointing
- R: Cost of recovery
- V: Cost of verification

Objective

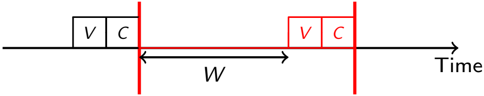
- Design a periodic computing pattern that minimizes the expected execution time (makespan) of the application



Last verification of a pattern is always perfect to avoid saving corrupted checkpoints

Outline

Revisiting Young/Daly (Base Pattern P_c)



Proposition

The expected time to execute a base pattern P_c *of work length* W *is*

$$\mathbb{E}(W) = W + V + C + \lambda W(W + V + R) + O(\lambda^2 W^3)$$

Proof. First, express the expected execution time recursively:

$$\mathbb{E}(W) = W + V + (1 - e^{-\lambda W})(R + \mathbb{E}(W)) + e^{-\lambda W} C$$

Then, solve the recursion and take first-order approximation

Approximation is accurate if platform MTBF is large in front of the resilience parameters

Revisiting Young/Daly (Base Pattern P_c)

Proposition

The optimal work length W^* of the base pattern P_c is

$$W^* = \sqrt{\frac{V+C}{\lambda}}$$

and the optimal expected overhead is

$$\textsc{Overhead}^* = 2\sqrt{\lambda(V+C)} + O(\lambda)$$

Proof. Derive the overhead from the expected execution time:

$$\begin{aligned}\textsc{Overhead} &= \frac{\mathbb{E}(W)}{W} - 1 \\ &= \frac{V+C}{W} + \lambda W + \lambda(V+R) + O(\lambda^2 W^2)\end{aligned}$$

Balance W to minimize $\textsc{Overhead}$

Revisiting Young/Daly (Base Pattern P_c)

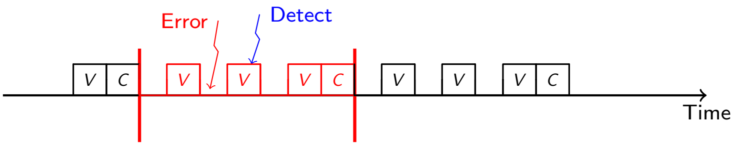
Recall from the waste analysis:

	Fail-stop errors	Silent errors
Pattern	$T = W + C$	$T = W + V + C$
WASTE_{ff}	$\frac{C}{T}$	$\frac{V+C}{T}$
$\text{WASTE}_{\text{fail}}$	$\lambda(D + R + \frac{W}{2})$	$\lambda(R + W + V)$
Optimal period	$\sqrt{\frac{2C}{\lambda}}$	$\sqrt{\frac{V+C}{\lambda}}$
$\text{WASTE}[opt]$	$\sqrt{2\lambda C}$	$2\sqrt{\lambda(V + C)}$

Outline

Pattern with several verifications

Perform several verifications before each checkpoint:



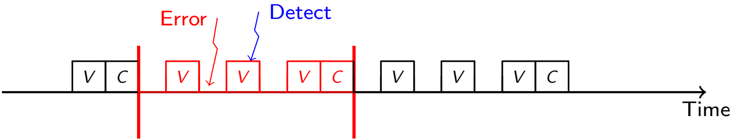
- ☺ silent error is detected earlier in the pattern
- ☹ additional overhead in fault-free executions

What is the optimal checkpointing period?
How many verifications to use?
Where are their positions?

Hands-on session at 14:45!

Pattern with several verifications

Perform several verifications before each checkpoint:



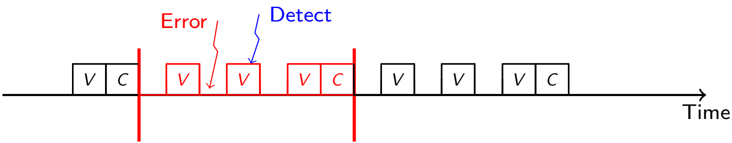
- ☺ silent error is detected earlier in the pattern
- ☹ additional overhead in fault-free executions

What is the optimal checkpointing period?
How many verifications to use?
Where are their positions?

Hands-on session at 14:45!

Pattern with several verifications

Perform several verifications before each checkpoint:



- ☺ silent error is detected earlier in the pattern
- ☹ additional overhead in fault-free executions

What is the optimal checkpointing period?
How many verifications to use?
Where are their positions?

Hands-on session at 14:45!

Outline

Leitmotiv

Resilient research on resilience

Models needed to assess techniques at scale
without bias ☺

Conclusion

- Multiple approaches to Fault Tolerance
- Application-Specific Fault Tolerance will always provide more benefits:
    - Checkpoint size reduction (when needed)
    - Portability (can run on different hardware, different deployment, etc..)
    - Diversity of use (can be used to restart the execution and change parameters in the middle)
    - More about this tomorrow at 10:00: "ABFT techniques" (Frederic Vivien)

Conclusion

- Multiple approaches to Fault Tolerance
- Application-Specific Fault Tolerance will always provide more benefits:
  - Checkpoint size reduction (when needed)
  - Portability (can run on different hardware, different deployment, etc..)
  - Diversity of use (can be used to restart the execution and change parameters in the middle)
  - More about this tomorrow at 10:00: "ABFT techniques" (Frederic Vivien)

Conclusion

- Multiple approaches to Fault Tolerance
- General Purpose Fault Tolerance is a required feature of the platforms
  - Not every computer scientist needs to learn how to write fault-tolerant applications
  - Not all parallel applications can be ported to a fault-tolerant version
- Faults are a feature of the platform. Why should it be the role of the programmers to handle them?

Conclusion

General Purpose Fault Tolerance

- Software/hardware techniques to reduce checkpoint, recovery, migration times and to improve failure prediction
- Need to deal with silent errors and design/use verification mechanisms
- General problem: multi-criteria scheduling problem execution time/energy/reliability
- Add replication
- Consider best resource usage (performance trade-offs)
- Need combine all these approaches and find optimal checkpointing periods!

Several challenging algorithmic/scheduling problems ☺

Bibliography

Exascale

• Toward Exascale Resilience, Cappello F. et al., IJHPCA 23, 4 (2009)

• The International Exascale Software Roadmap, Dongarra, J., Beckman, P. et al., IJHPCA 25, 1 (2011)

Models

• Checkpointing strategies for parallel jobs, Bougeret M. et al., SC'2011

• Unified model for assessing checkpointing protocols at extreme-scale, Bosilca G. et al., INRIA RR-7950, 2012

Buddy

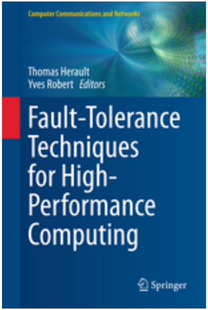
• Revisiting the double checkpointing algorithm, Dongarra J., Hérault T., Robert Y., INRIA RR-8196, 2012

Silent errors

• Assessing general-purpose algorithms to cope with fail-stop and silent errors, Benoit A., Cavelan A., Robert Y., Sun H., INRIA RR-8599, 2014

• Optimal resilience patterns to cope with fail-stop and silent errors, Benoit A., Cavelan A., Robert Y., Sun H., INRIA RR-8786, 2015

Bibliography



New Monograph, Springer Verlag 2015

Thanks to Yves Robert, Thomas Hérault, George Bosilca, Aurélien Bouteiller and Hongyang Sun, from whom I borrowed some slides