

An overview of fault-tolerant techniques for HPC

Yves Robert

ENS Lyon & Institut Universitaire de France
University of Tennessee Knoxville

yves.robert@ens-lyon.fr

<http://graal.ens-lyon.fr/~yrobert/keynote-ic3-delhi2013.pdf>

P2S2 Keynote – October 1, 2013

Outline

- 1 Introduction
 - Large-scale computing platforms
 - Faults and failures
- 2 ABFT for dense linear algebra kernels
- 3 Checkpointing
 - Process checkpointing
 - Coordinated checkpointing
 - Young/Daly's approximation
- 4 Probabilistic models for checkpointing
 - Coordinated checkpointing
 - Hierarchical checkpointing
- 5 Other techniques
 - Replication
 - Failure Prediction
 - Silent errors
 - In-memory checkpointing
- 6 Conclusion

Thanks ...



Outline

- 1 Introduction
 - Large-scale computing platforms
 - Faults and failures
- 2 ABFT for dense linear algebra kernels
- 3 Checkpointing
 - Process checkpointing
 - Coordinated checkpointing
 - Young/Daly's approximation
- 4 Probabilistic models for checkpointing
 - Coordinated checkpointing
 - Hierarchical checkpointing
- 5 Other techniques
 - Replication
 - Failure Prediction
 - Silent errors
 - In-memory checkpointing
- 6 Conclusion

Outline

- 1 Introduction
 - Large-scale computing platforms
 - Faults and failures
- 2 ABFT for dense linear algebra kernels
- 3 Checkpointing
 - Process checkpointing
 - Coordinated checkpointing
 - Young/Daly's approximation
- 4 Probabilistic models for checkpointing
 - Coordinated checkpointing
 - Hierarchical checkpointing
- 5 Other techniques
 - Replication
 - Failure Prediction
 - Silent errors
 - In-memory checkpointing
- 6 Conclusion

Exascale platforms (courtesy J. Dongarra)

Potential System Architecture with a cap of \$200M and 20MW

Systems	2011 K computer	2019	Difference Today & 2019
System peak	10.5 Pflop/s	1 Eflop/s	O(100)
Power	12.7 MW	~20 MW	
System memory	1.6 PB	32 - 64 PB	O(10)
Node performance	128 GF	1,2 or 15TF	O(10) – O(100)
Node memory BW	64 GB/s	2 - 4TB/s	O(100)
Node concurrency	8	O(1k) or 10k	O(100) – O(1000)
Total Node Interconnect BW	20 GB/s	200-400GB/s	O(10)
System size (nodes)	88,124	O(100,000) or O(1M)	O(10) – O(100)
Total concurrency	705,024	O(billion)	O(1,000)
MTTI	days	O(1 day)	- O(10)

Exascale platforms (courtesy C. Engelmann & S. Scott)

Toward Exascale Computing (My Roadmap)

Based on proposed DOE roadmap with MTTI adjusted to scale linearly

Systems	2009	2011	2015	2018
System peak	2 Peta	20 Peta	100-200 Peta	1 Exa
System memory	0.3 PB	1.6 PB	5 PB	10 PB
Node performance	125 GF	200GF	200-400 GF	1-10TF
Node memory BW	25 GB/s	40 GB/s	100 GB/s	200-400 GB/s
Node concurrency	12	32	O(100)	O(1000)
Interconnect BW	1.5 GB/s	22 GB/s	25 GB/s	50 GB/s
System size (nodes)	18,700	100,000	500,000	O(million)
Total concurrency	225,000	3,200,000	O(50,000,000)	O(billion)
Storage	15 PB	30 PB	150 PB	300 PB
IO	0.2 TB/s	2 TB/s	10 TB/s	20 TB/s
MTTI	4 days	19 h 4 min	3 h 52 min	1 h 56 min
Power	6 MW	~10MW	~10 MW	~20 MW

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)

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	5min	1h

Exascale

More nodes = \neq Petascale $\times 1000$ (between failures)

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

Joint Laboratory for Petascale Computing

Also an issue at Petascale

INRIA NCSA

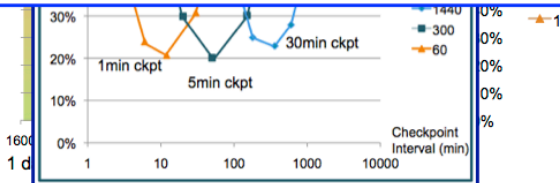
Fault tolerance becomes critical at Petascale (MTTI \leq 1day)
 Poor fault tolerance design may lead to huge overhead

Overhead of checkpoint/restart

Cost of non optimal checkpoint intervals: 100%

Today, 20% or more of the computing capacity in a large high-performance computing system is wasted due to failures and recoveries.

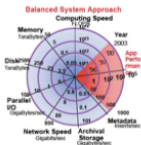
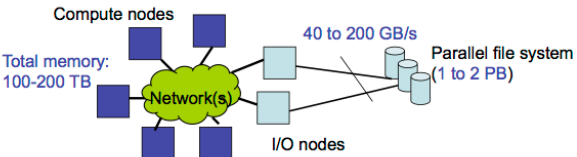
Dr. E.N. (Mootaz) Elnozahy et al. *System Resilience at Extreme Scale, DARPA*



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

Classic approach for FT: Checkpoint-Restart

Typical "Balanced Architecture" for PetaScale Computers



RoadRunner



TACO



LLNL BG/L



➔ 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

Outline

1

Introduction

- Large-scale computing platforms
- **Faults and failures**

2

ABFT for dense linear algebra kernels

3

Checkpointing

- Process checkpointing
- Coordinated checkpointing
- Young/Daly's approximation

4

Probabilistic models for checkpointing

- Coordinated checkpointing
- Hierarchical checkpointing

5

Other techniques


- Replication
- Failure Prediction
- Silent errors
- In-memory checkpointing

6

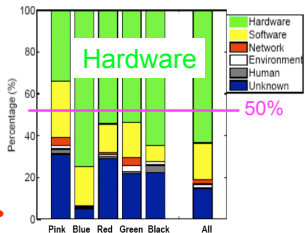
Conclusion

Error sources (courtesy Franck 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): 
- 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.

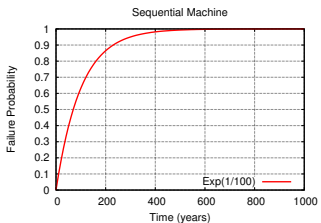
Hardware errors, Disks, processors, memory, network

Conclusion: Both Hardware and Software failures have to be considered

A few definitions

- Many types of 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

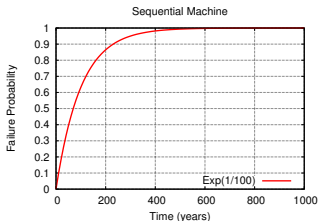
Failure distributions: (1) Exponential



$Exp(\lambda)$: Exponential distribution law of parameter λ :

- Pdf: $f(t) = \lambda e^{-\lambda t} dt$ for $t \geq 0$
- Cdf: $F(t) = 1 - e^{-\lambda t}$
- Mean = $\frac{1}{\lambda}$

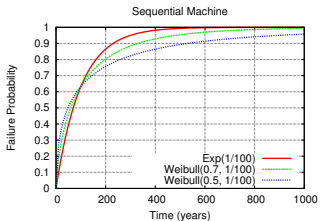
Failure distributions: (1) Exponential



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

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

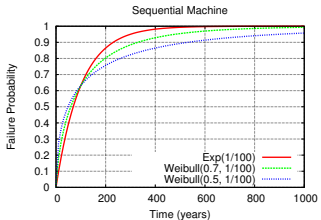
Failure distributions: (2) Weibull



Weibull(k, λ): Weibull distribution law of shape parameter k and scale parameter λ :

- Pdf: $f(t) = k\lambda(t\lambda)^{k-1}e^{-(\lambda t)^k} dt$ for $t \geq 0$
- Cdf: $F(t) = 1 - e^{-(\lambda t)^k}$
- Mean = $\frac{1}{\lambda}\Gamma(1 + \frac{1}{k})$

Failure distributions: (2) Weibull



X random variable for $Weibull(k, \lambda)$ failure inter-arrival times:

- If $k < 1$: failure rate decreases with time
 "infant mortality": defective items fail early
- If $k = 1$: $Weibull(1, \lambda) = Exp(\lambda)$ constant failure time

Failure distributions: with several processors

- Processor (or node): any entity subject to failures
⇒ approach **agnostic to granularity**

- If the MTBF is μ with one processor,
what is its value μ_p with p processors?

- Well, it depends 😞

Failure distributions: with several processors

- Processor (or node): any entity subject to failures
⇒ approach **agnostic to granularity**

- If the MTBF is μ with one processor,
what is its value μ_p with p processors?

- Well, it depends 😞

With rejuvenation

- Rebooting all p processors after a failure
- Platform failure distribution
 \Rightarrow minimum of p IID processor distributions
- With p distributions $Exp(\lambda)$:

$$\min (Exp(\lambda_1), Exp(\lambda_2)) = Exp(\lambda_1 + \lambda_2)$$

$$\mu = \frac{1}{\lambda} \Rightarrow \mu_p = \frac{\mu}{p}$$

- With p distributions $Weibull(k, \lambda)$:

$$\min_{1..p} (Weibull(k, \lambda)) = Weibull(k, p^{1/k} \lambda)$$

$$\mu = \frac{1}{\lambda} \Gamma(1 + \frac{1}{k}) \Rightarrow \mu_p = \frac{\mu}{p^{1/k}}$$

Without rejuvenation (= real life)

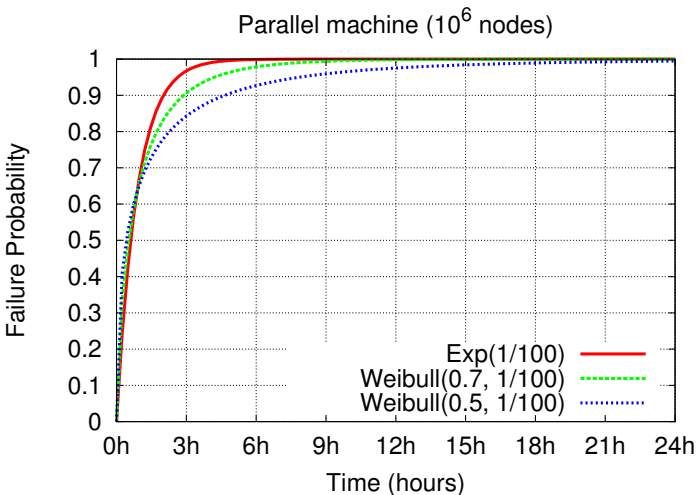
- Rebooting only faulty processor
- Platform failure distribution
⇒ superposition of p IID processor distributions

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

Values from the literature

- MTBF of one processor: between 1 and 125 years
- Shape parameters for Weibull: $k = 0.5$ or $k = 0.7$
- Failure trace archive from INRIA
(<http://fta.inria.fr>)
- Computer Failure Data Repository from LANL
(<http://institutes.lanl.gov/data/fdata>)

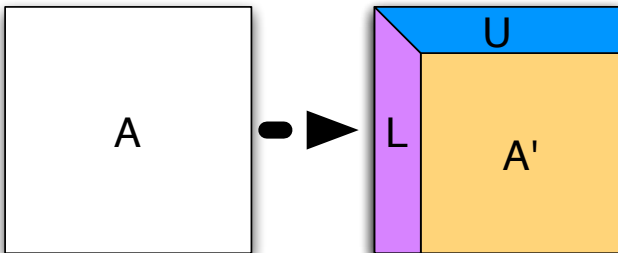
Does it matter?



Outline

- 1 Introduction
 - Large-scale computing platforms
 - Faults and failures
- 2 ABFT for dense linear algebra kernels
- 3 Checkpointing
 - Process checkpointing
 - Coordinated checkpointing
 - Young/Daly's approximation
- 4 Probabilistic models for checkpointing
 - Coordinated checkpointing
 - Hierarchical checkpointing
- 5 Other techniques
 - Replication
 - Failure Prediction
 - Silent errors
 - In-memory checkpointing
- 6 Conclusion

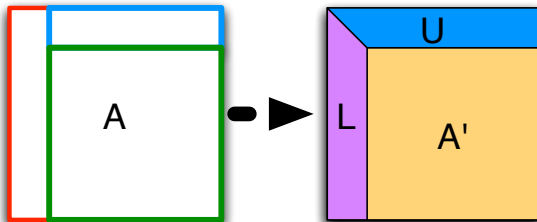
Tiled LU factorization



- Solve $A \cdot x = b$ (hard)
- Transform A into a LU factorization
- Solve $L \cdot y = B \cdot b$, then $U \cdot x = y$

Tiled LU factorization

TRSM - Update row block

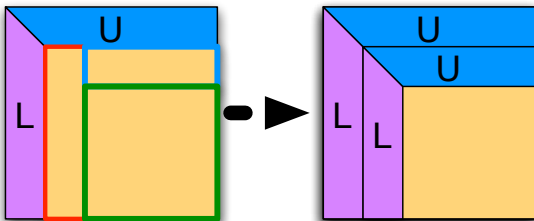


GETF2: factorize a column block
 GEMM: Update the trailing matrix

- Solve $A \cdot x = b$ (hard)
- Transform A into a LU factorization
- Solve $L \cdot y = B \cdot b$, then $U \cdot x = y$

Tiled LU factorization

TRSM - Update row block



GETF2: factorize a column block

GEMM: Update the trailing matrix

- Solve $A \cdot x = b$ (hard)
- Transform A into a LU factorization
- Solve $L \cdot y = B \cdot b$, then $U \cdot x = y$

Tiled LU factorization

0	2	4	0	2	4	0	2
1	3	5	1	3	5	1	3
0	2	4	0	2	4	0	2
1	3	5	1	3	5	1	3
0	2	4	0	2	4	0	2
1	3	5	1	3	5	1	3
0	2	4	0	2	4	0	2
1	3	5	1	3	5	1	3

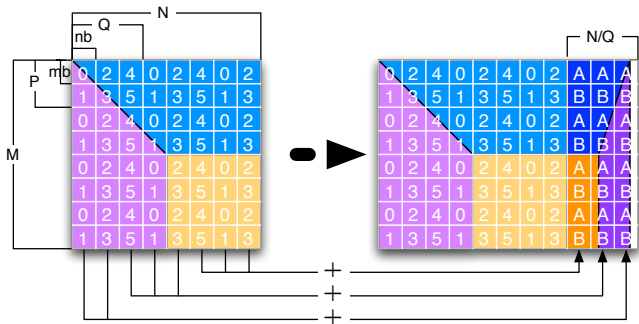


Failure of rank 2

0		4	0		4	0	
1	3	5	1	3	5	1	3
0		4	0		4	0	
1	3	5	1	3	5	1	3
0		4	0		4	0	
1	3	5	1	3	5	1	3
0		4	0		4	0	
1	3	5	1	3	5	1	3

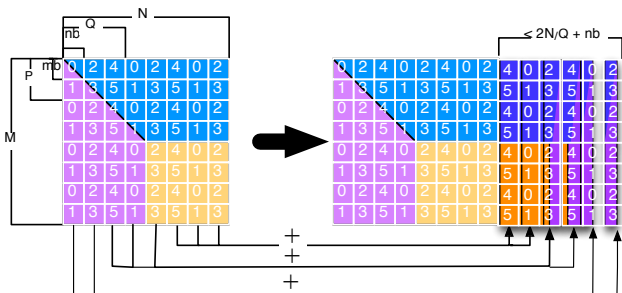
- 2D Block Cyclic Distribution (here 2×3)
- A single failure \Rightarrow many data lost

Algorithm Based Fault Tolerant LU decomposition



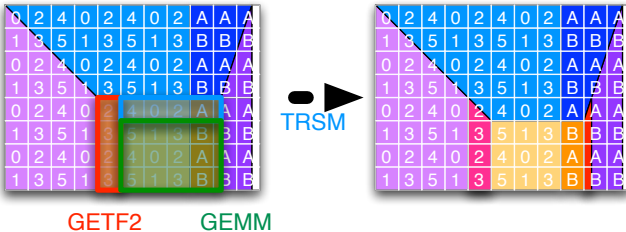
- Checksum: invertible operation on row/column data
 - Checksum replication avoided by **dedicating** additional computing resources to checksum storage

Algorithm Based Fault Tolerant LU decomposition



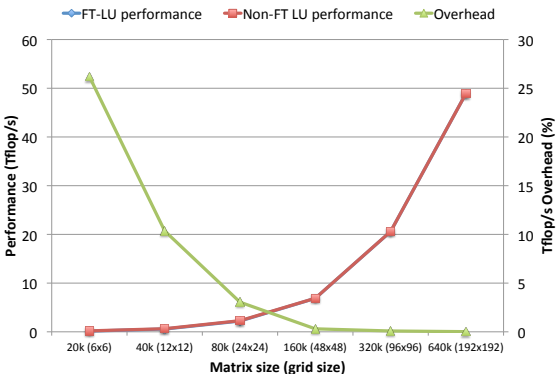
- Checksum: invertible operation on row/column data
 - Checksum blocks are doubled, to allow recovery when data and checksum are lost together (no extra resource needed)

Algorithm Based Fault Tolerant LU decomposition



- Checksum: invertible operation on row/column data
 - Key idea of ABFT: applying the operation on data and checksum preserves the checksum properties

Performance



MPI-Next ULFM Performance

- Open MPI with ULFM; Kraken supercomputer;

Outline

- 1 Introduction
 - Large-scale computing platforms
 - Faults and failures
- 2 ABFT for dense linear algebra kernels
- 3 Checkpointing**
 - Process checkpointing
 - Coordinated checkpointing
 - Young/Daly's approximation
- 4 Probabilistic models for checkpointing
 - Coordinated checkpointing
 - Hierarchical checkpointing
- 5 Other techniques
 - Replication
 - Failure Prediction
 - Silent errors
 - In-memory checkpointing
- 6 Conclusion

Outline

- 1 Introduction
 - Large-scale computing platforms
 - Faults and failures
- 2 ABFT for dense linear algebra kernels
- 3 Checkpointing**
 - **Process checkpointing**
 - Coordinated checkpointing
 - Young/Daly's approximation
- 4 Probabilistic models for checkpointing
 - Coordinated checkpointing
 - Hierarchical checkpointing
- 5 Other techniques
 - Replication
 - Failure Prediction
 - Silent errors
 - In-memory checkpointing
- 6 Conclusion

Maintaining Redundant Information

Goal

- General Purpose Fault Tolerance Techniques: work despite application behavior
- Two adversaries: **Failures** & **Application**
- Use automatically computed redundant information
 - At given instants: checkpoint
 - At any instant: replication
 - Anything in between: checkpoint + message logging

Process Checkpointing

Goal

- Save the current state of the *process*
 - FT Protocols save a *possible* state of the parallel application

Techniques

- User-level checkpointing
- System-level checkpointing
- Blocking call
- Asynchronous call

System-level checkpointing

Blocking Checkpointing

Relatively intuitive: `checkpoint(filename)`

Cost: no process activity during whole checkpoint operation

- Different implementations: OS syscall; dynamic library; compiler assisted
- Create a serial file that can be loaded in a process image. Usually on same architecture / OS / software environment

- Entirely transparent
- Preemptive (often needed for library-level checkpointing)

- Lack of portability
- Large size of checkpoint (\approx memory footprint)

Storage

Remote Reliable Storage

Intuitive. I/O intensive. Disk usage.

Memory Hierarchy

- local memory
- local disk (SSD, HDD)
- remote disk
 - Scalable Checkpoint Restart Library
<http://scalablecr.sourceforge.net>

Checkpoint is valid when finished on reliable storage

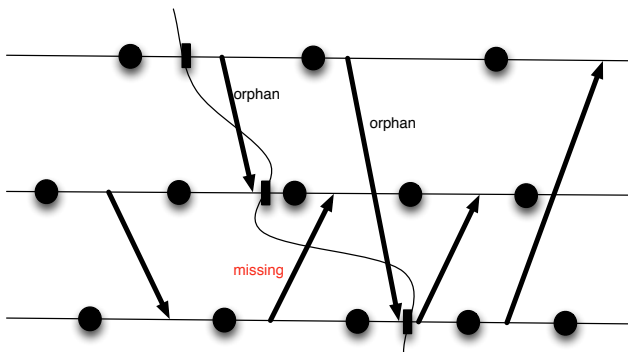
Distributed Memory Storage

- In-memory checkpointing
- Disk-less checkpointing

Outline

- 1 Introduction
 - Large-scale computing platforms
 - Faults and failures
- 2 ABFT for dense linear algebra kernels
- 3 Checkpointing**
 - Process checkpointing
 - Coordinated checkpointing**
 - Young/Daly's approximation
- 4 Probabilistic models for checkpointing
 - Coordinated checkpointing
 - Hierarchical checkpointing
- 5 Other techniques
 - Replication
 - Failure Prediction
 - Silent errors
 - In-memory checkpointing
- 6 Conclusion

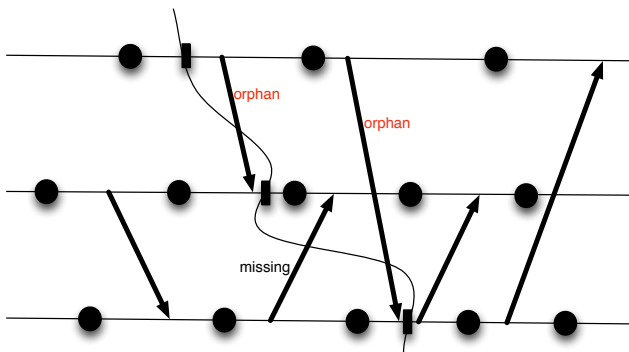
Coordinated checkpointing



Definition (Missing Message)

A message is missing if in the current configuration, the sender sent it, while the receiver did not receive it

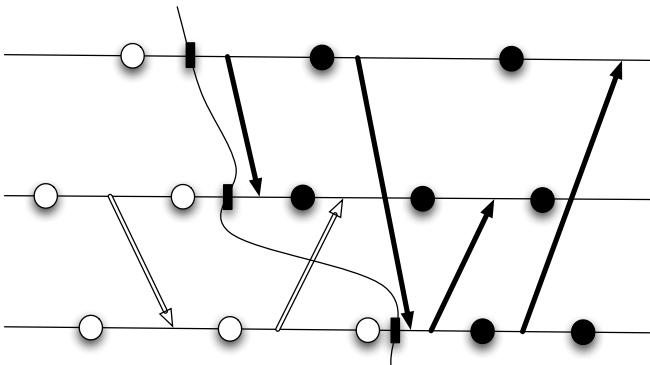
Coordinated checkpointing



Definition (Orphan Message)

A message is orphan if in the current configuration, the receiver received it, while the sender did not send it

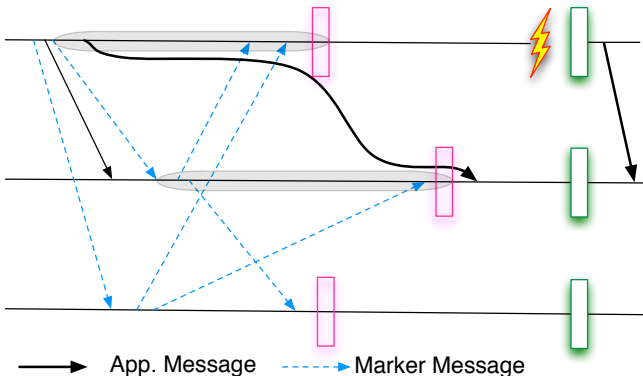
Coordinated checkpointing



Create a consistent view of the application (no orphan messages)

- Messages belong to a checkpoint wave or another
- All communication channels must be flushed (all2all)

Coordinated checkpointing



- Silences the network during checkpoint
- Missing messages recorded

Outline

- 1 Introduction
 - Large-scale computing platforms
 - Faults and failures

- 2 ABFT for dense linear algebra kernels

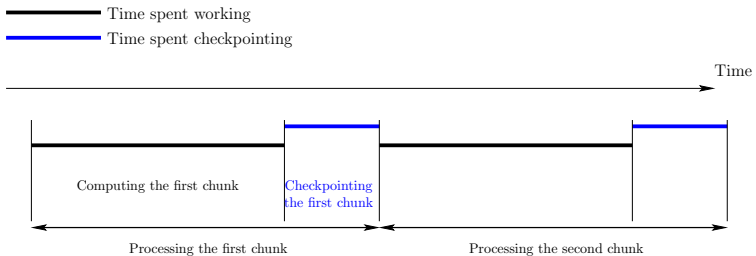
- 3 Checkpointing**
 - Process checkpointing
 - Coordinated checkpointing
 - Young/Daly's approximation**

- 4 Probabilistic models for checkpointing
 - Coordinated checkpointing
 - Hierarchical checkpointing

- 5 Other techniques
 - Replication
 - Failure Prediction
 - Silent errors
 - In-memory checkpointing

- 6 Conclusion

Checkpointing cost



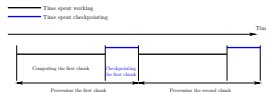
Blocking model: while a checkpoint is taken, no computation can be performed

Framework

- Periodic checkpointing policy of period T
- Independent and identically distributed 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**

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

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

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

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

Waste due to failures

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

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

N_{faults} number of failures during execution

T_{lost} : average time lost par failures

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

$T_{\text{lost}}?$

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

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

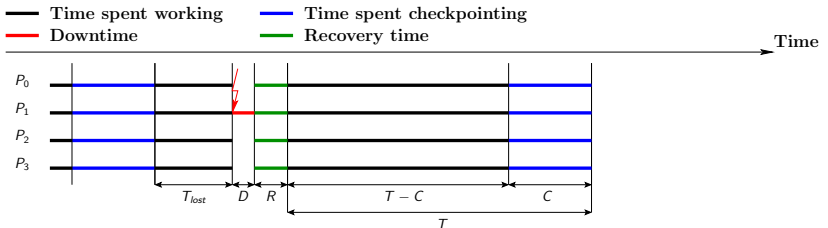
N_{faults} number of failures during execution

T_{lost} : average time lost par failures

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

$T_{\text{lost}}?$

Computing T_{lost}



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

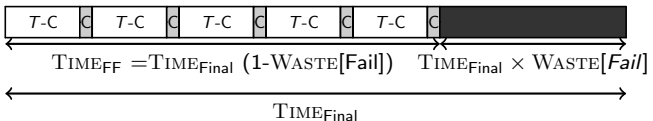
- ⇒ Instants when periods begin and failures strike are independent
- ⇒ Valid for all distribution laws, regardless of their particular shape

Waste due to failures

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

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

Total waste



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

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

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

Waste minimization

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

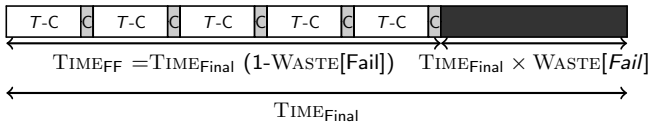
$$\text{WASTE} = \frac{u}{T} + v + wT$$

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

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

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

Comparison with Young/Daly



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

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

Daly: $\text{TIME}_{\text{final}} = (1 + \text{WASTE}[fail]) \text{TIME}_{\text{FF}}$

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

Young: $\text{TIME}_{\text{final}} = (1 + \text{WASTE}[fail]) \text{TIME}_{\text{FF}}$ and $D = R = 0$

$$\Rightarrow T = \sqrt{2\mu C} + C$$

Validity of the approach (1/3)

Technicalities

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

Outline

- 1 Introduction
 - Large-scale computing platforms
 - Faults and failures
- 2 ABFT for dense linear algebra kernels
- 3 Checkpointing
 - Process checkpointing
 - Coordinated checkpointing
 - Young/Daly's approximation
- 4 Probabilistic models for checkpointing**
 - Coordinated checkpointing**
 - Hierarchical checkpointing**
- 5 Other techniques
 - Replication
 - Failure Prediction
 - Silent errors
 - In-memory checkpointing
- 6 Conclusion

Outline

- 1 Introduction
 - Large-scale computing platforms
 - Faults and failures

- 2 ABFT for dense linear algebra kernels

- 3 Checkpointing
 - Process checkpointing
 - Coordinated checkpointing
 - Young/Daly's approximation

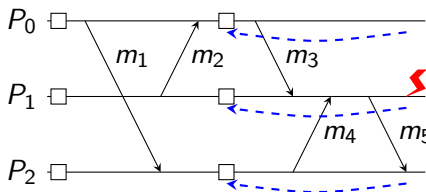
- 4 Probabilistic models for checkpointing
 - **Coordinated checkpointing**
 - Hierarchical checkpointing

- 5 Other techniques
 - Replication
 - Failure Prediction
 - Silent errors
 - In-memory checkpointing

- 6 Conclusion

Background: coordinated checkpointing protocols

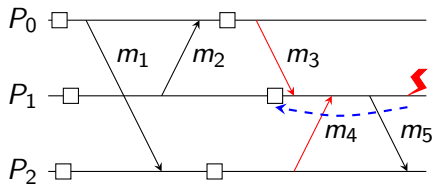
- Coordinated checkpoints over all processes
- Global restart after a failure



- 😊 No risk of cascading rollbacks
- 😊 No need to log messages
- 😞 All processors need to roll back

Background: message logging protocols

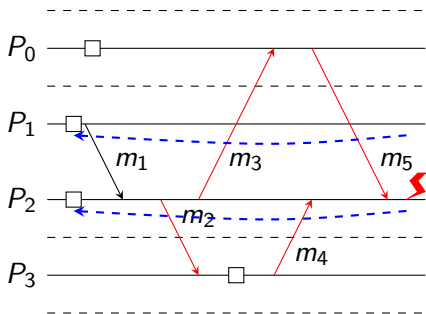
- Message content logging (sender memory)
- Restart of failed process only



- ☺ No cascading rollbacks
- ☺ Number of processes to roll back
- ☹ Memory occupation
- ☹ Overhead

Background: hierarchical protocols

- 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-groups messages
 - Slows down failure-free execution
 - Increases checkpoint size/time
- 😊 Faster re-execution with logged messages

Which checkpointing protocol to use?

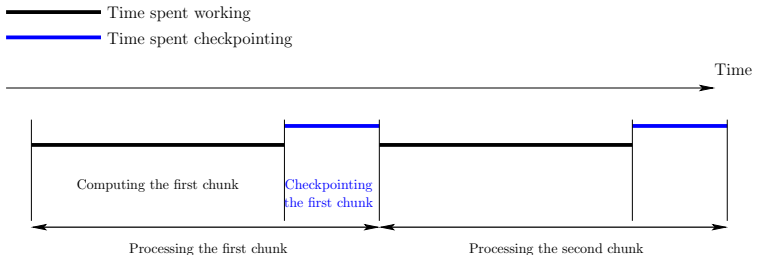
Coordinated checkpointing

- 😊 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

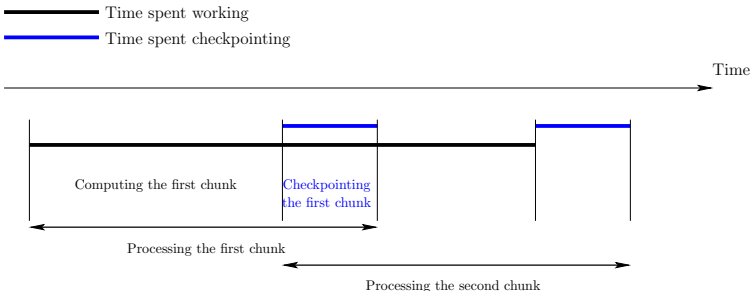
- 😞 Need to log inter-groups messages
 - Slowdowns failure-free execution
 - Increases checkpoint size/time
- 😊 Only processors from failed group need to roll back
- 😊 Faster re-execution with logged messages
- 😊 Rumor: Should scale to very large platforms

Coordinated checkpointing



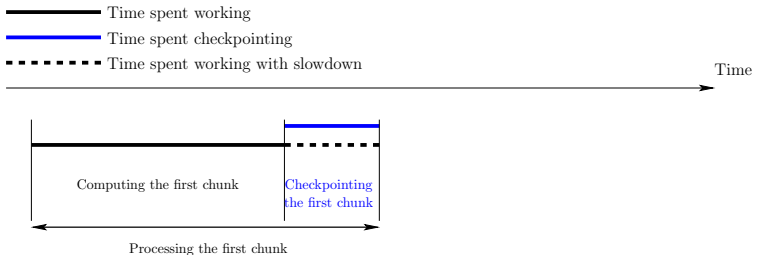
Blocking model: checkpointing blocks all computations

Coordinated checkpointing



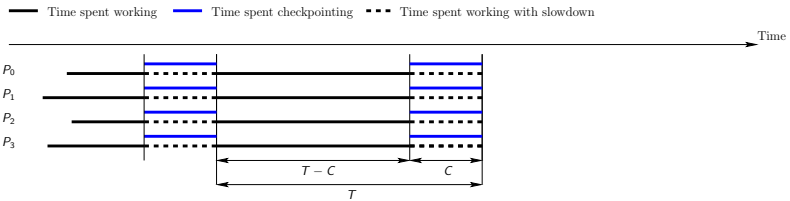
Non-blocking model: checkpointing has no impact on computations (e.g., first copy state to RAM, then copy RAM to disk)

Coordinated checkpointing



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

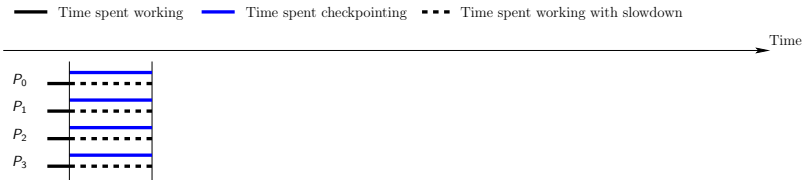


Time elapsed since last checkpoint: T

Amount of computations executed: $WORK = (T - C) + \alpha C$

$$WASTE[FF] = \frac{T - WORK}{T}$$

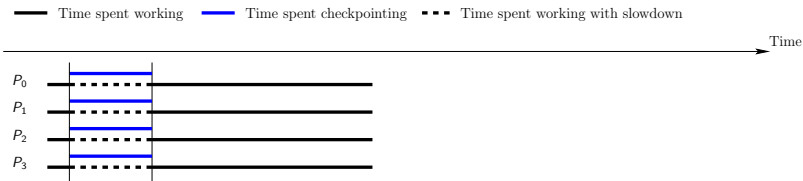
Waste due to failures



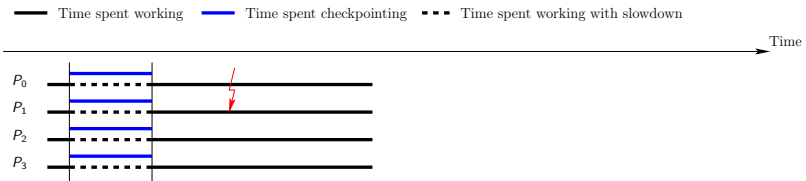
Failure can happen

- ① During computation phase
- ② During checkpointing phase

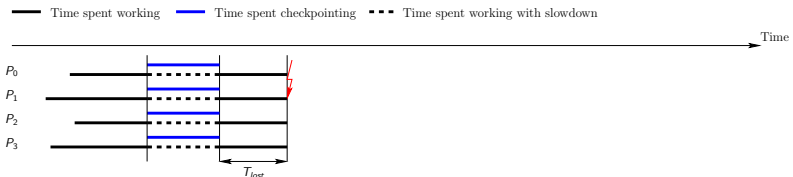
Waste due to failures



Waste due to failures

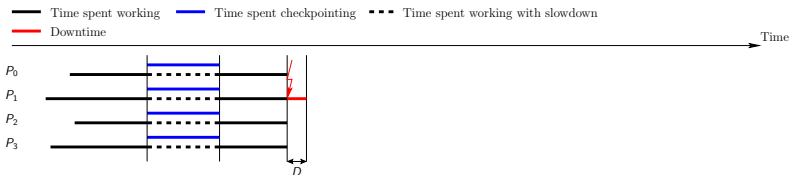


Waste due to failures

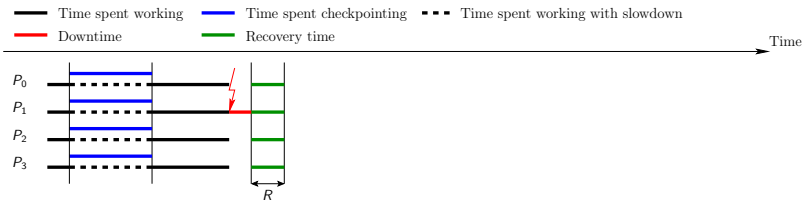


Coordinated checkpointing protocol: when one processor is victim of a failure, all processors lose their work and must roll back to last checkpoint

Waste due to failures in computation phase

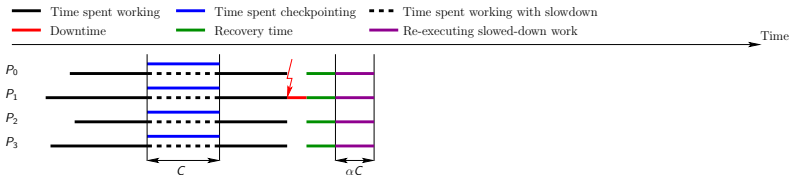


Waste due to failures in computation phase



Coordinated checkpointing protocol: all processors must recover from last checkpoint

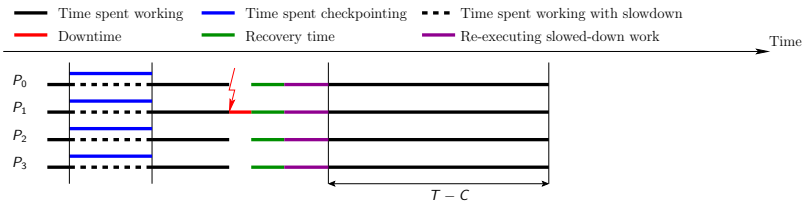
Waste due to failures in computation phase



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

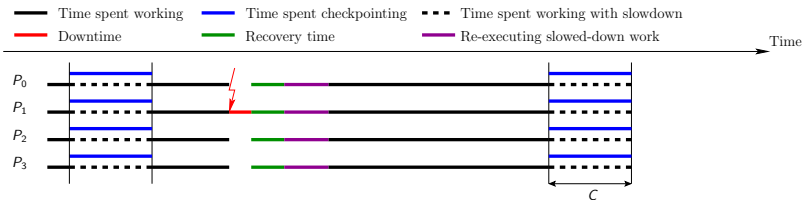
But no checkpoint is taken in parallel, hence this re-execution is faster than the original computation

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

Outline

- 1 Introduction
 - Large-scale computing platforms
 - Faults and failures

- 2 ABFT for dense linear algebra kernels

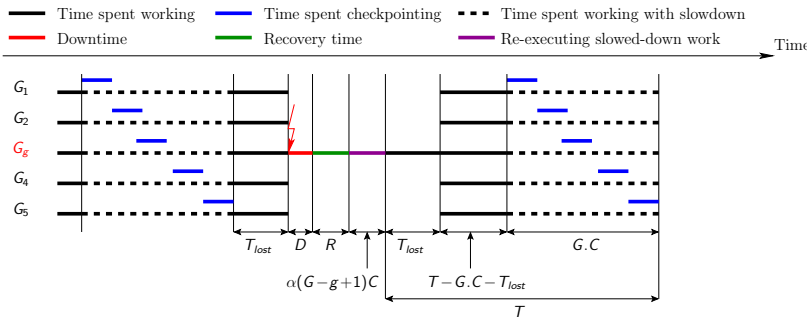
- 3 Checkpointing
 - Process checkpointing
 - Coordinated checkpointing
 - Young/Daly's approximation

- 4 Probabilistic models for checkpointing
 - Coordinated checkpointing
 - **Hierarchical checkpointing**

- 5 Other techniques
 - Replication
 - Failure Prediction
 - Silent errors
 - In-memory checkpointing

- 6 Conclusion

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

Accounting for message logging: Impact on work

- ☹ Logging messages slows down execution:
 \Rightarrow WORK becomes λ WORK, where $0 < \lambda < 1$
 Typical value: $\lambda \approx 0.98$
- 😊 Re-execution after a failure is faster:
 \Rightarrow RE-EXEC becomes $\frac{\text{RE-EXEC}}{\rho}$, where $\rho \in [1..2]$
 Typical value: $\rho \approx 1.5$

$$\text{WASTE}[FF] = \frac{T - \lambda \text{WORK}}{T}$$

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

Accounting for message logging: Impact on checkpoint size

- Inter-groups 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

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

$$\text{WORK} = \lambda(T - (1 - \alpha)GC(q))$$

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

Three case studies

Coord-IO

Coordinated approach: $C = C_{\text{Mem}} = \frac{\text{Mem}}{b_{io}}$

where Mem is the memory footprint of the application

Hierarch-IO

Several (large) groups, *I/O-saturated*

⇒ groups checkpoint sequentially

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

Hierarch-Port

Very large number of smaller groups, *port-saturated*

⇒ some groups checkpoint in parallel

Groups of q_{\min} processors, where $q_{\min} b_{port} \geq b_{io}$

Three applications

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

Computing β for 2D-Stencil

$$C(q) = C_0(q) + \text{Logged_Msg} = C_0(q)(1 + \beta \text{WORK})$$

Real $n \times n$ matrix and $p \times p$ grid

$$\text{Work} = \frac{9b^2}{s_p}, \quad b = n/p$$

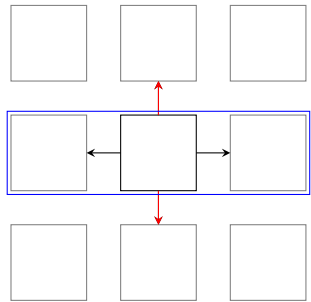
Each process sends a block to its 4 neighbors

HIERARCH-IO:

- 1 group = 1 grid row
- 2 out of the 4 messages are logged
- $\beta = \frac{2s_p}{9b^3}$

HIERARCH-PORT:

- β doubles



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})		I/O Bandwidth (b_{port})
					Read	Write	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	200,000 (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

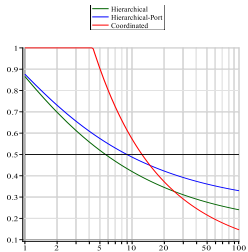
Checkpoint time

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

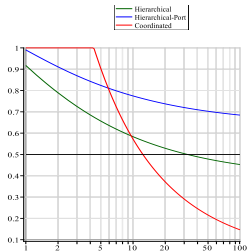
- Large time to dump the memory
- Using $1\%C$
- Comparing with $0.1\%C$ for exascale platforms
- $\alpha = 0.3$, $\lambda = 0.98$ and $\rho = 1.5$

Plotting formulas – Platform: Titan

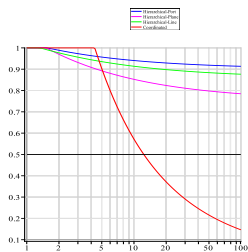
Stencil 2D



Matrix product



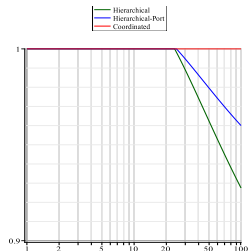
Stencil 3D



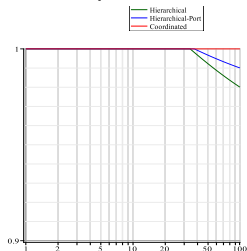
Waste as a function of processor MTBF μ_{ind}

Platform: K-Computer

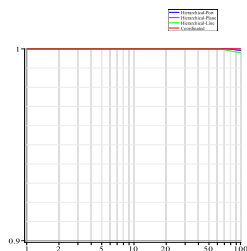
Stencil 2D



Matrix product



Stencil 3D



Waste as a function of processor MTBF μ_{ind}

Plotting formulas – Platform: Exascale

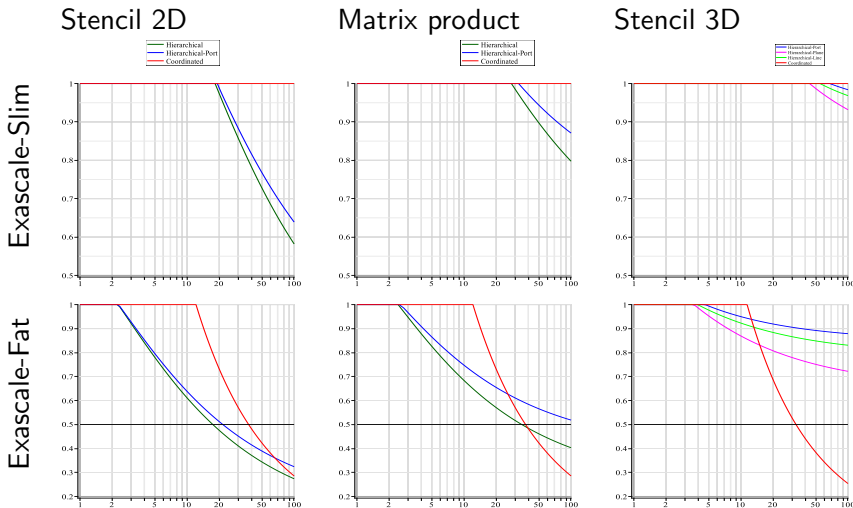
WASTE = 1 for all scenarios!!!

Plotting formulas – Platform: Exascale

WASTE = 1 for all scenarios!!!

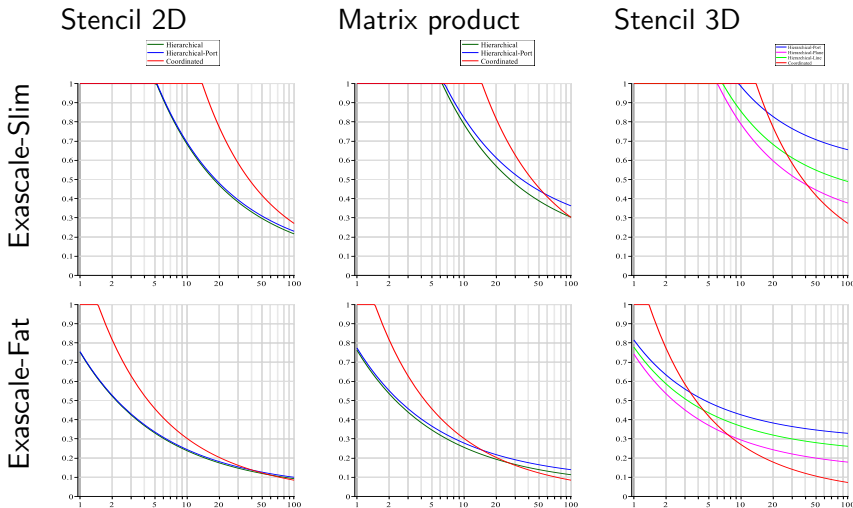
Goodbye Exascale?!

Plotting formulas – Platform: Exascale with $C = 1,000$



Waste as a function of processor MTBF μ_{ind} , $C = 1,000$

Plotting formulas – Platform: Exascale with $C = 100$

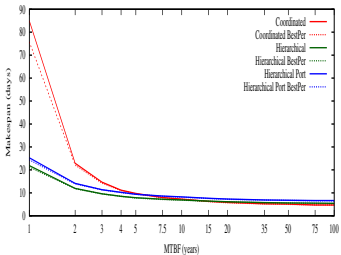


Waste as a function of processor MTBF μ_{ind} , $C = 100$

Simulations – Platform: Titan

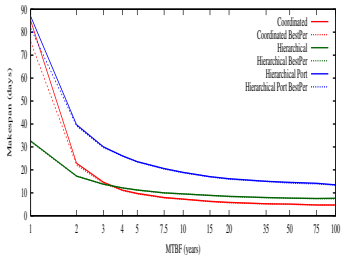
Stencil 2D

Coordinated ———
 Coordinated BestPer - - - - -



Matrix product

Hierarchical ———
 Hierarchical BestPer - - - - -
 Hierarchical Port ———
 Hierarchical Port BestPer - - - - -



Makespan (in days) as a function of processor MTBF μ_{ind}

Simulations – Platform: Exascale with $C = 1,000$

Stencil 2D

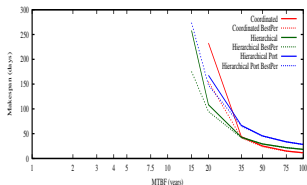
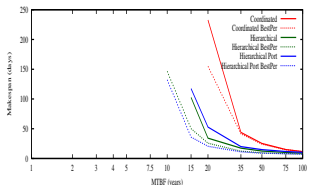
Matrix product

Coordinated ———
Coordinated BestPer - - - - -

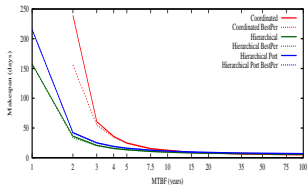
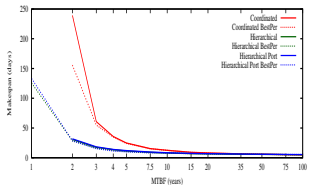
Hierarchical ———
Hierarchical BestPer - - - - -

Hierarchical Port ———
Hierarchical Port BestPer - - - - -

Exascale-Slim



Exascale-Fat



Makespan (in days) as a function of processor MTBF μ_{ind} , $C = 1,000$

Simulations – Platform: Exascale with $C = 100$

Stencil 2D

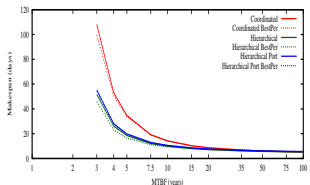
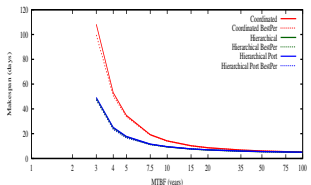
Matrix product

Coordinated ———
Coordinated BestPer - - - - -

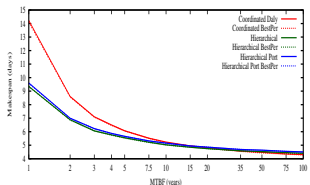
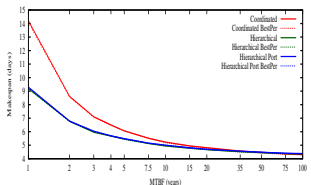
Hierarchical ———
Hierarchical BestPer - - - - -

Hierarchical Port ———
Hierarchical Port BestPer - - - - -

Exascale-Slim



Exascale-Fat



Makespan (in days) as a function of processor MTBF μ_{ind} , $C = 100$

Outline

- 1 Introduction
 - Large-scale computing platforms
 - Faults and failures
- 2 ABFT for dense linear algebra kernels
- 3 Checkpointing
 - Process checkpointing
 - Coordinated checkpointing
 - Young/Daly's approximation
- 4 Probabilistic models for checkpointing
 - Coordinated checkpointing
 - Hierarchical checkpointing
- 5 Other techniques
 - Replication
 - Failure Prediction
 - Silent errors
 - In-memory checkpointing
- 6 Conclusion

Outline

- 1 Introduction
 - Large-scale computing platforms
 - Faults and failures

- 2 ABFT for dense linear algebra kernels

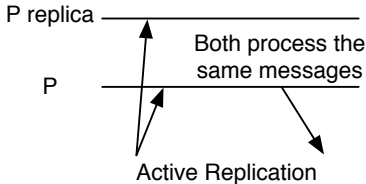
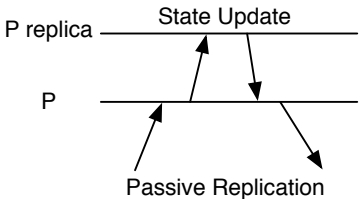
- 3 Checkpointing
 - Process checkpointing
 - Coordinated checkpointing
 - Young/Daly's approximation

- 4 Probabilistic models for checkpointing
 - Coordinated checkpointing
 - Hierarchical checkpointing

- 5 Other techniques
 - **Replication**
 - Failure Prediction
 - Silent errors
 - In-memory checkpointing

- 6 Conclusion

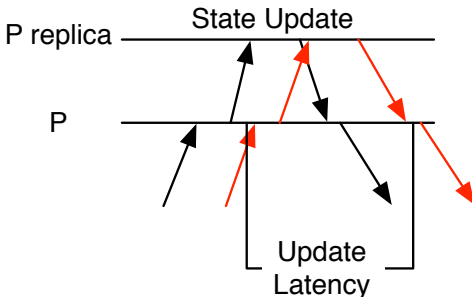
Replication



Idea

- Each process is replicated on a resource that has small chance to be hit by the same failure as its replica
- In case of failure, one of the replicas will continue working, while the other recovers
- Passive Replication / Active Replication

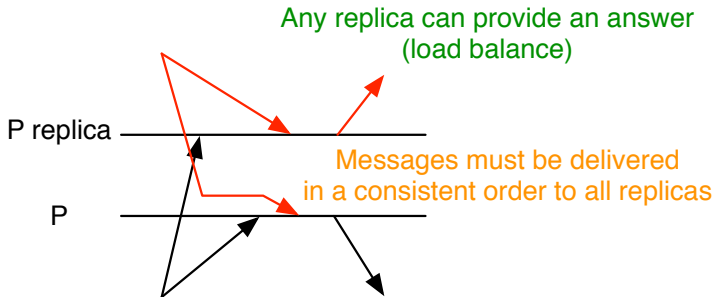
Replication



Challenges

- Passive replication: latency of state update
- Active replication: ordering of decision → internal additional communications
- **By nature: replication → at most 50% machine efficiency**

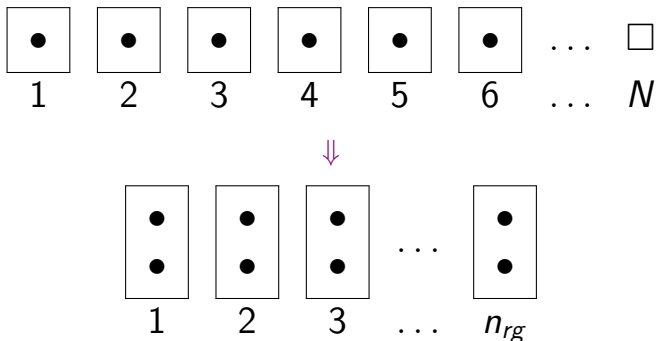
Replication



Challenges

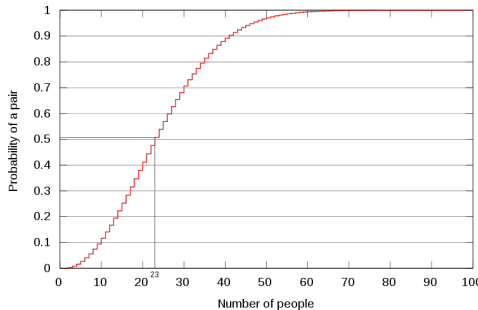
- Passive replication: latency of state update
- Active replication: ordering of decision → internal additional communications
- **By nature: replication → at most 50% machine efficiency**

PROCESS REPLICATION

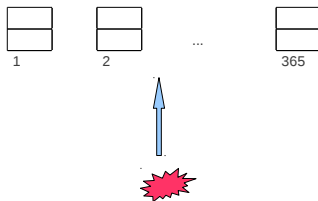


- Each process replicated $g \geq 2$ times \rightarrow *replica-group*
- $n_{rg} =$ number of replica-groups ($g \times n_{rg} = N$)
- Study for $g = 2$ by Ferreira et al., SC'2011

Analogy with birthday problem

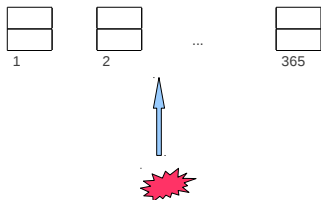


Analogy with birthday problem



$n = n_{rg}$ bins, throw balls until one bin gets two balls

Analogy with birthday problem

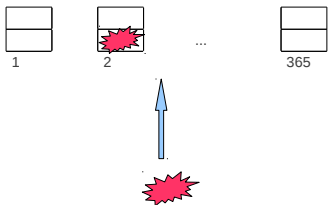


$n = n_{rg}$ bins, throw balls until one bin gets two balls

Expected number of balls to throw:

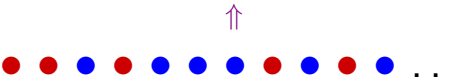
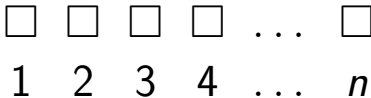
$$\text{Birthday}(n) = 1 + \int_0^{+\infty} e^{-x} (1 + x/n)^{n-1} dx$$

Analogy with birthday problem



But second failure may hit already struck replica ☹️

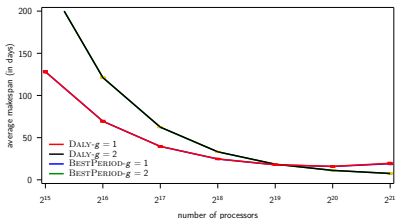
Analogy with birthday problem



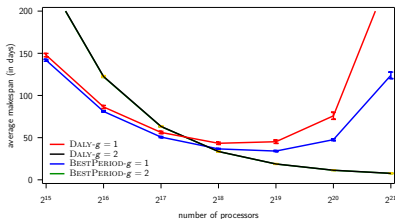
$n = n_{rg}$ bins, red and blue balls

Mean Number of Failures to Interruption (bring down application)
 $MNFTI$ = expected number of balls to throw
until one bin gets one ball of each color

Failure distribution



(a) Exponential



(b) Weibull, $k = 0.7$

Crossover point for replication when $\mu_{ind} = 125$ years

Outline

- 1 Introduction
 - Large-scale computing platforms
 - Faults and failures
- 2 ABFT for dense linear algebra kernels
- 3 Checkpointing
 - Process checkpointing
 - Coordinated checkpointing
 - Young/Daly's approximation
- 4 Probabilistic models for checkpointing
 - Coordinated checkpointing
 - Hierarchical checkpointing
- 5 Other techniques
 - Replication
 - **Failure Prediction**
 - Silent errors
 - In-memory checkpointing
- 6 Conclusion

Framework

Predictor

- Exact prediction dates (at least C seconds in advance)
- Recall r : fraction of faults that are predicted
- Precision p : fraction of fault predictions that are correct

Events

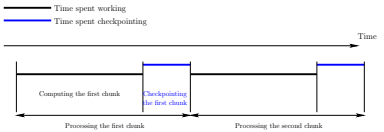
- *true positive*: predicted faults
- *false positive*: fault predictions that did not materialize as actual faults
- *false negative*: unpredicted faults

Algorithm

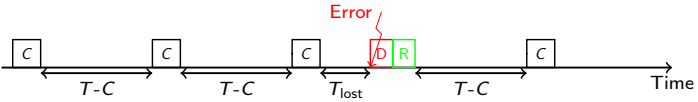
- 1 While no fault prediction is available:
 - checkpoints taken periodically with period T
- 2 When a fault is predicted at time t :
 - take a checkpoint ALAP (completion right at time t)
 - after the checkpoint, complete the execution of the period

Computing the waste

① **Fault-free execution:** $WASTE[FF] = \frac{C}{T}$



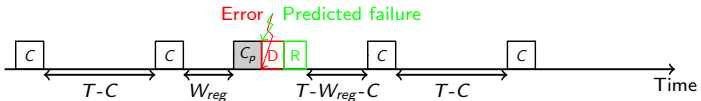
② **Unpredicted faults:** $\frac{1}{\mu_{NP}} \left[D + R + \frac{T}{2} \right]$



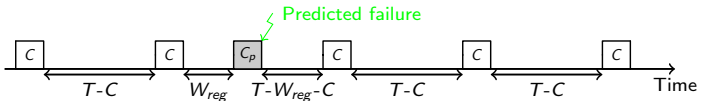
$$WASTE[fail] = \frac{1}{\mu} \left[(1-r) \frac{T}{2} + D + R + \frac{r}{p} C \right] \Rightarrow T_{opt} \approx \sqrt{\frac{2\mu C}{1-r}}$$

Computing the waste

③ Predictions: $\frac{1}{\mu p} [p(C + D + R) + (1 - p)C]$



with actual fault (true positive)

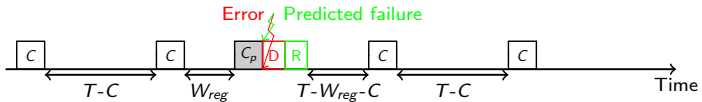


no actual fault (false negative)

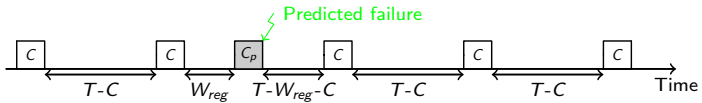
$$\text{WASTE}[fail] = \frac{1}{\mu} \left[(1 - r) \frac{T}{2} + D + R + \frac{r}{p} C \right] \Rightarrow T_{opt} \approx \sqrt{\frac{2\mu C}{1 - r}}$$

Computing the waste

③ Predictions: $\frac{1}{\mu p} [p(C + D + R) + (1 - p)C]$



with actual fault (true positive)



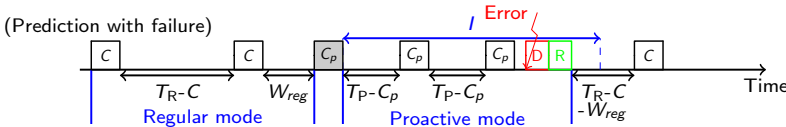
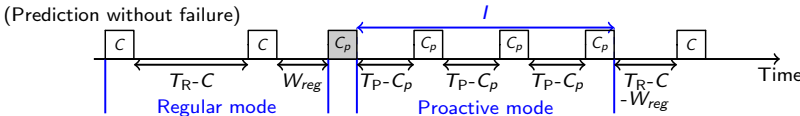
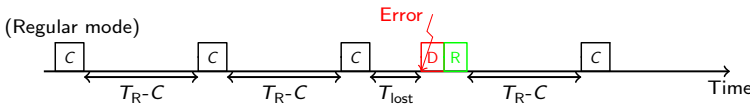
no actual fault (false negative)

$$\text{WASTE}[fail] = \frac{1}{\mu} \left[(1 - r) \frac{T}{2} + D + R + \frac{r}{p} C \right] \Rightarrow T_{opt} \approx \sqrt{\frac{2\mu C}{1 - r}}$$

Refinements

- Use different value C_p for proactive checkpoints
- Avoid checkpointing too frequently for false negatives
 - ⇒ Only trust predictions with some fixed probability q
 - ⇒ Ignore predictions with probability $1 - q$
 - Conclusion: trust predictor always or never ($q = 0$ or $q = 1$)
- Trust prediction depending upon position in current period
 - ⇒ Increase q when progressing
 - ⇒ Break-even point $\frac{C_p}{p}$

With prediction windows



Gets too complicated! 😞

Outline

- 1 Introduction
 - Large-scale computing platforms
 - Faults and failures

- 2 ABFT for dense linear algebra kernels

- 3 Checkpointing
 - Process checkpointing
 - Coordinated checkpointing
 - Young/Daly's approximation

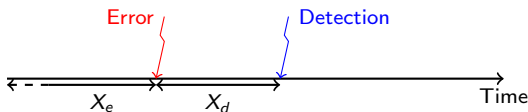
- 4 Probabilistic models for checkpointing
 - Coordinated checkpointing
 - Hierarchical checkpointing

- 5 Other techniques
 - Replication
 - Failure Prediction
 - **Silent errors**
 - In-memory checkpointing

- 6 Conclusion

Silent errors

- Instantaneous error detection \Rightarrow fail-stop failures, e.g. resource crash
- Silent errors (data corruption) \Rightarrow detection latency

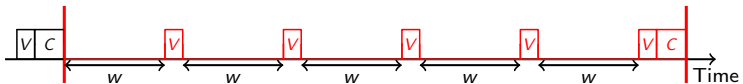


Error and detection latency

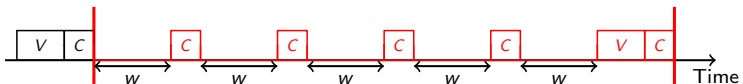
- Last checkpoint may have saved an already corrupted state
- Even when saving k checkpoints: which one to roll back to?

Coupling checkpointing and verification

- Verification mechanism of cost V
- Repeat periodic pattern:



Small cost V : 5 verifications for 1 checkpoint



Large cost V : 5 checkpoints for 1 verification

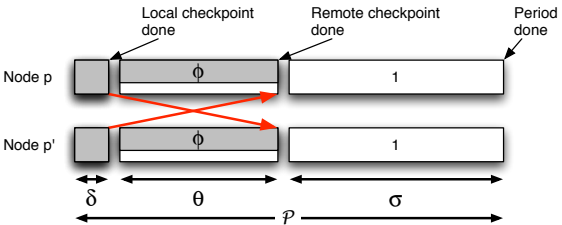
Outline

- 1 Introduction
 - Large-scale computing platforms
 - Faults and failures
- 2 ABFT for dense linear algebra kernels
- 3 Checkpointing
 - Process checkpointing
 - Coordinated checkpointing
 - Young/Daly's approximation
- 4 Probabilistic models for checkpointing
 - Coordinated checkpointing
 - Hierarchical checkpointing
- 5 Other techniques
 - Replication
 - Failure Prediction
 - Silent errors
 - **In-memory checkpointing**
- 6 Conclusion

Motivation

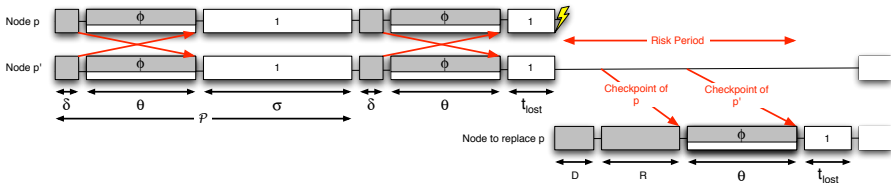
- Checkpoint transfer and storage
⇒ critical issues of rollback/recovery protocols
- Stable storage: high cost
- Distributed in-memory storage:
 - Store checkpoints in local memory ⇒ no centralized storage
😊 Much better scalability
 - Replicate checkpoints ⇒ application survives single failure
😞 Still, risk of fatal failure in some (unlikely) scenarios

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

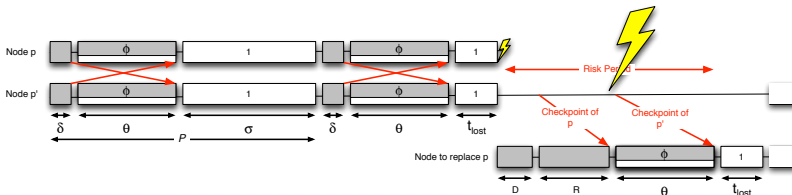
Failures



- After failure: downtime D and recovery from buddy node
- Two checkpoint files lost, must be re-sent to faulty processor

Best trade-off between performance and risk?

Failures



- After failure: downtime D and recovery from buddy node
- Two checkpoint files lost, must be re-sent to faulty processor
- Application **at risk** until complete reception of both messages

Best trade-off between performance and risk?

Outline

- 1 Introduction
 - Large-scale computing platforms
 - Faults and failures

- 2 ABFT for dense linear algebra kernels

- 3 Checkpointing
 - Process checkpointing
 - Coordinated checkpointing
 - Young/Daly's approximation

- 4 Probabilistic models for checkpointing
 - Coordinated checkpointing
 - Hierarchical checkpointing

- 5 Other techniques
 - Replication
 - Failure Prediction
 - Silent errors
 - In-memory checkpointing

- 6 Conclusion

Conclusion

- Multiple approaches to Fault Tolerance
- Application-specific FT will always provide more benefits
- General-purpose FT will always be needed
 - 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

- Software/hardware techniques to reduce checkpoint, recovery, migration times and to improve failure prediction
- Multi-criteria scheduling problem
 - execution time/energy/reliability
 - add replication
 - best resource usage (performance trade-offs)
- Need combine all these approaches!

Several challenging algorithmic/scheduling problems 😊

Extended version of this talk: see SC'12 or ICS'13 tutorial with Thomas Héroult. Available at

<http://graal.ens-lyon.fr/~yrobert/>

Thanks

INRIA & ENS Lyon

- Anne Benoit
- Frédéric Vivien
- PhD students (Guillaume Aupy, Dounia Zaidouni)

UT Knoxville

- George Bosilca
- Aurélien Bouteiller
- Jack Dongarra
- **Thomas Héroult (joint tutorial at SC'12 & ICS'13)**

Others

- Franck Cappello, UIUC-Inria joint lab
- Henri Casanova, Univ. Hawai'i