Fault-Tolerance Techniques for Computing at Scale

Yves Robert

ENS Lyon & Institut Universitaire de France
University of Tennessee Knoxville

yves.robert@ens-lyon.fr

http://graal.ens-lyon.fr/~yrobert/keynote-ccgrid2014.pdf

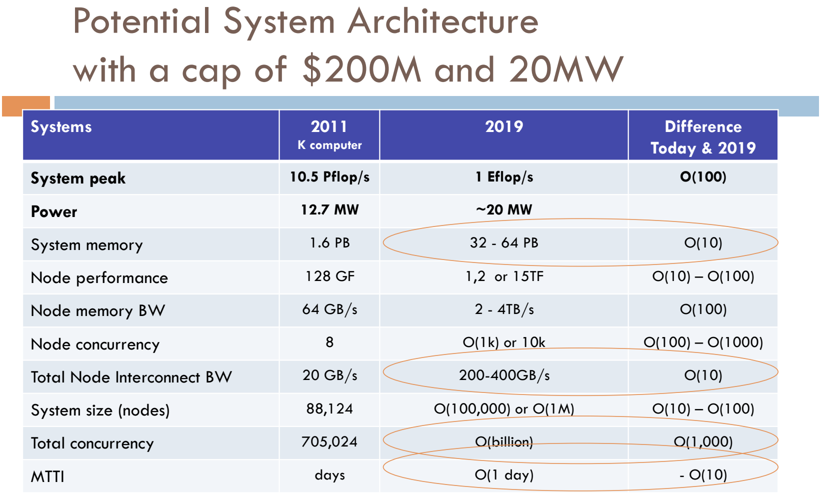
CCGrid – May 29, 2014

Outline

Outline

Outline

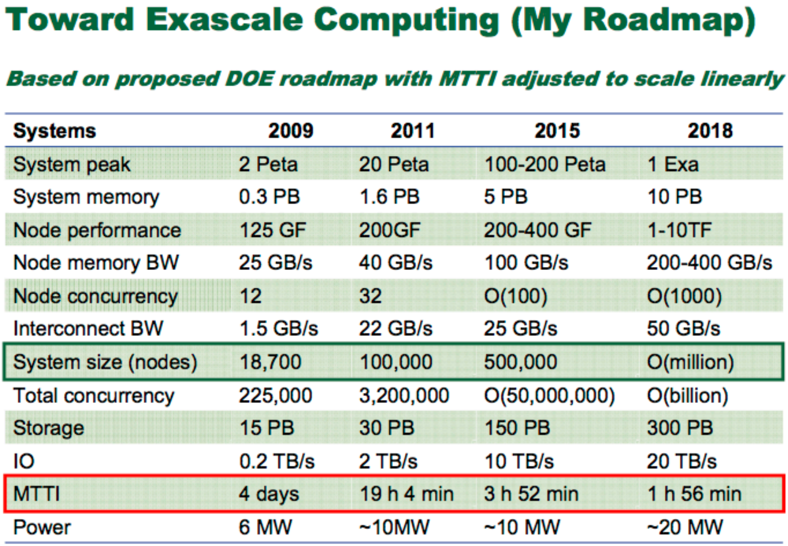
Exascale platforms (courtesy J. Dongarra)



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 ⇒ Shorter MTBF (Mean Time Between Failures)

Exascale platforms

- Hierarchica[illegible]
  - 10^5 or 10^6 node[illegible]
  - Each node equipped w[illegible] 10^4 [illegible] 10^3 cores
- Failure-prone

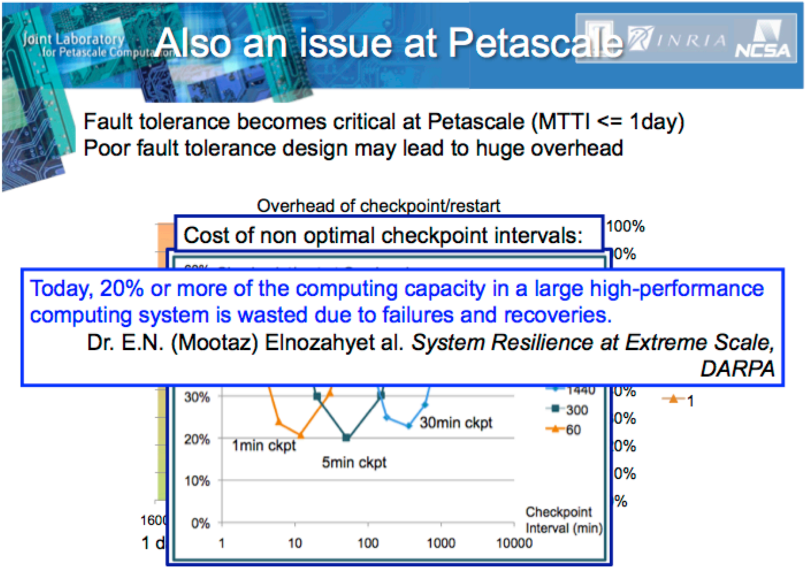
MTBF – o[illegible]node MTBF – [illegible]atform [illegible]f 10^6 nodes	1 year 30sec	10[illegible]ars 5m[illegible]	120 years 1h

Mor[illegible]odes = [illegible]een [illegible]lures)

Exascale

$\neq$ Petascale $\times 1000$

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



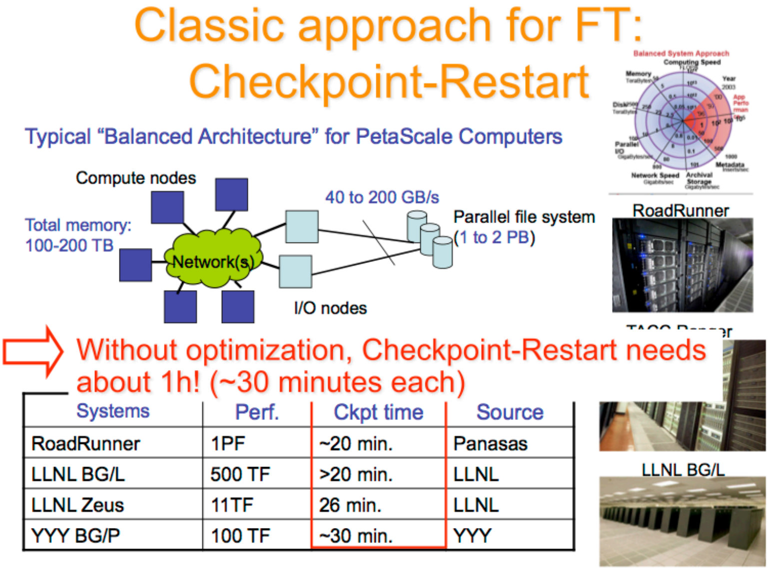
Also an issue at Petascale

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

Cost of non optimal checkpoint intervals:

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

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



Classic approach for FT: Checkpoint-Restart

Typical "Balanced Architecture" for PetaScale Computers

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

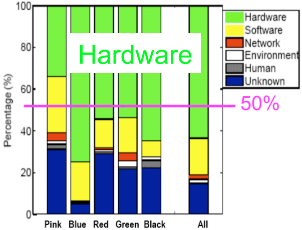
1. Introduction
   - Large-scale computing platforms
   - Faults and failures
2. Checkpointing
   - Coordinated checkpointing
   - Young/Daly's approximation
3. Models for faster checkpointing
   - Hierarchical checkpointing
   - In-memory checkpointing
   - Multilevel checkpointing
   - Checkpointing and prediction
   - Checkpointing and replication
4. Silent errors
   - Framework
   - ABFT for dense linear algebra kernels
   - Checkpointing and verification
5. 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



Relative frequency of root cause by system type.

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: fail-stop, unrecoverable, transient, silent data corruption (SDC)

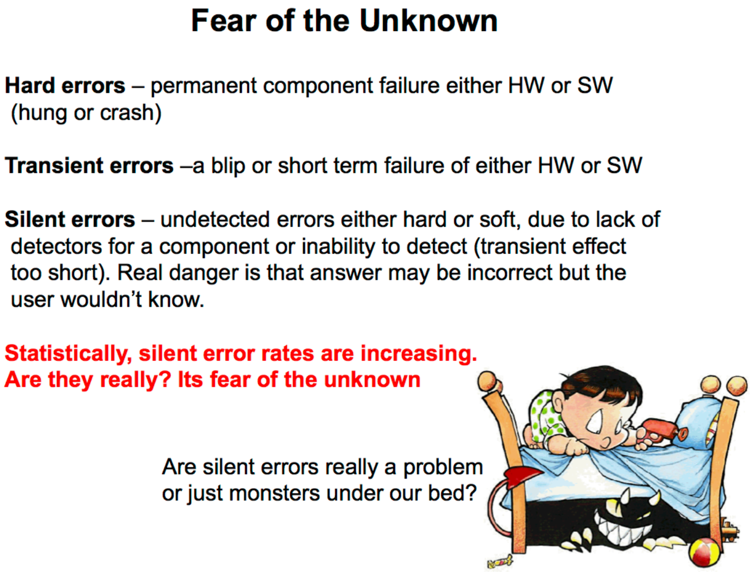
① Deal with faults that lead to application failures

Includes all hardware faults, and some software ones

Use *fault* and *failure* interchangeably

② Silent errors (SDC)

Should we be afraid? (courtesy Al Geist)



Fear of the Unknown

Hard errors – permanent component failure either HW or SW (hung or crash)

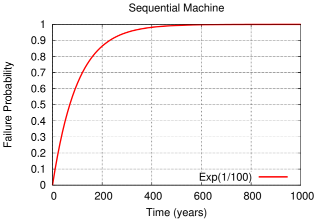
Transient errors –a blip or short term failure of either HW or SW

Silent errors – undetected errors either hard or soft, due to lack of detectors for a component or inability to detect (transient effect too short). Real danger is that answer may be incorrect but the user wouldn't know.

Statistically, silent error rates are increasing.
Are they really? Its fear of the unknown

Are silent errors really a problem or just monsters under our bed?

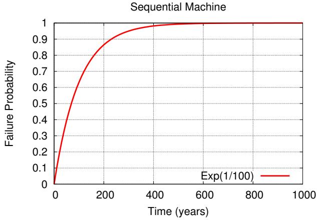
Failure distributions: (1) Exponential



$Exp(\lambda)$: 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 $= \frac{1}{\lambda}$

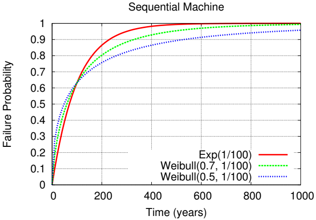
Failure distributions: (1) Exponential



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

- $\mathbb{P}(X \leq t) = 1 - e^{-\lambda t} dt$ (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) $\mu = \mathbb{E}(X) = \frac{1}{\lambda}$

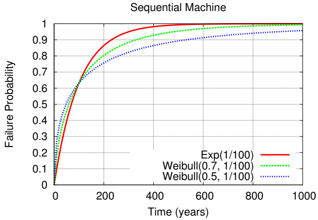
Failure distributions: (2) Weibull



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 $\mathit{Weibull}(k, \lambda)$ failure inter-arrival times:

- If $k < 1$: failure rate decreases with time
  "infant mortality": defective items fail early
- If $k = 1$: $\mathit{Weibull}(1, \lambda) = \mathit{Exp}(\lambda)$ 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_{ind} 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
  $\Rightarrow$ approach agnostic to granularity

- If the MTBF is μ_{ind} 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 $Exp(\lambda)$:

$$\min_{1..p}\left(Exp(\lambda)\right) = Exp(p\lambda)$$

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

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

Without rejuvenation (= real life)

- Rebooting only faulty processor
- Platform failure distribution
  $\Rightarrow$ superposition of p IID processor distributions

Theorem: $\mu_p = \frac{\mu_{\text{ind}}}{p}$ for arbitrary distributions

Values from the literature

- MTBF μ_{ind} 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`)

Outline

Outline

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

2 Checkpointing
- Coordinated checkpointing
- Young/Daly's approximation

3 Models for faster checkpointing
- Hierarchical checkpointing
- In-memory checkpointing
- Multilevel checkpointing
- Checkpointing and prediction
- Checkpointing and replication

4 Silent errors
- Framework
- ABFT for dense linear algebra kernels
- Checkpointing and verification

5 Conclusion

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

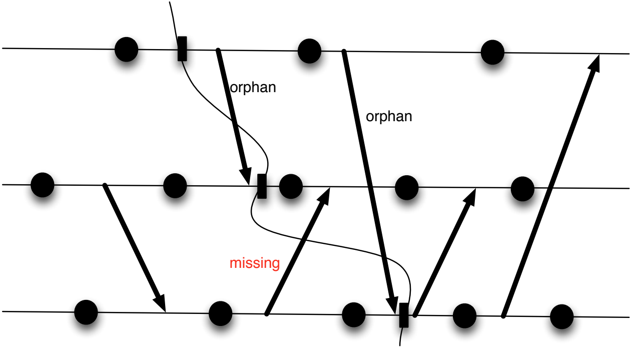
- local memory
- local disk (SSD, HDD)
- remote disk
  - Scalable Checkpoint Restart Library
    `http://scalablecr.sourceforge.net`

Checkpoint is valid when finished on reliable storage

Distributed Memory Storage

- In-memory checkpointing
- Disk-less checkpointing

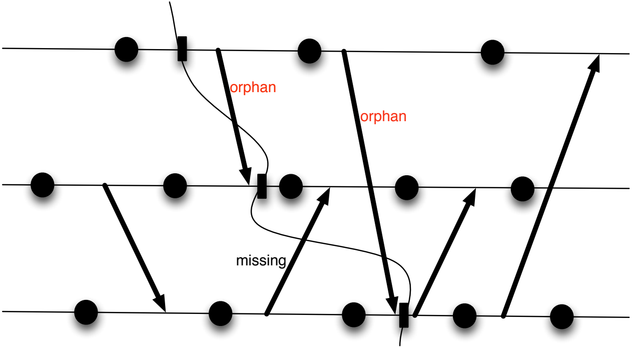
Coordinated checkpointing



Definition (Missing Message)

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

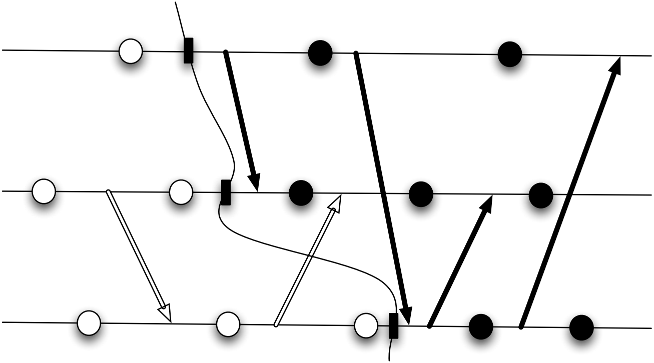
Coordinated checkpointing



Definition (Orphan Message)

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

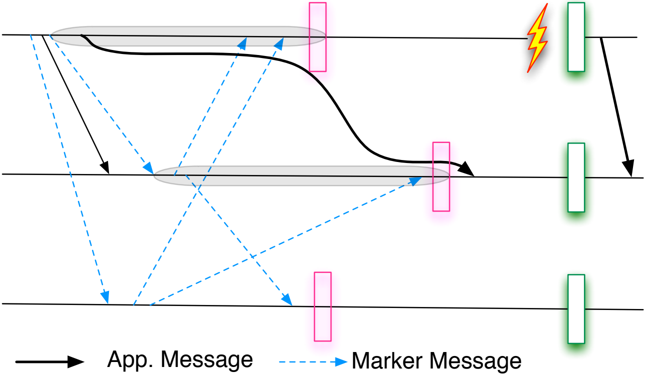
Coordinated checkpointing



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)

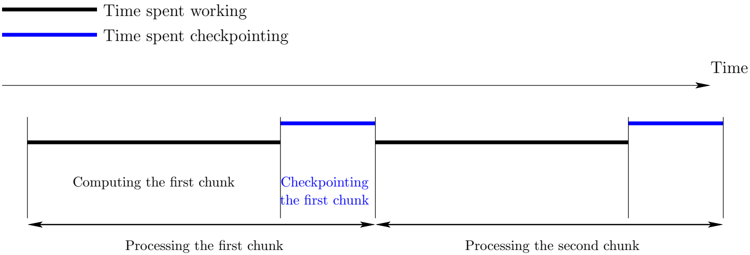
Coordinated checkpointing



- Silences the network during checkpoint
- Missing messages recorded

Outline

Checkpointing cost



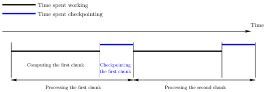
Blocking model: while a checkpoint is taken, no computation can be performed

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 ⇔ all processors available

Waste: fraction of time not spent for useful computations

Waste in fault-free execution



- $\textsc{Time}_{\mathsf{base}}$: application base time
- $\textsc{Time}_{\mathsf{FF}}$: with periodic checkpoints but failure-free

$$\textsc{Time}_{\mathsf{FF}} = \textsc{Time}_{\mathsf{base}} + \#checkpoints \times C$$

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

$$\textsc{Waste}[FF] = \frac{\textsc{Time}_{\mathsf{FF}} - \textsc{Time}_{\mathsf{base}}}{\textsc{Time}_{\mathsf{FF}}} = \frac{C}{T}$$

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 par failures

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

T_{lost}?

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

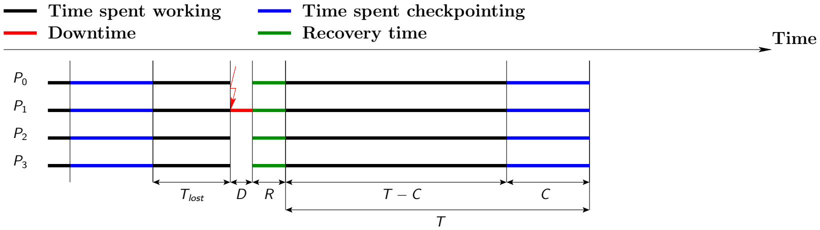
$$\textsc{Time}_{\text{final}} = \textsc{Time}_{\text{FF}} + N_{faults} \times T_{\text{lost}}$$

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

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

T_{lost}?

Computing T_{lost}



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

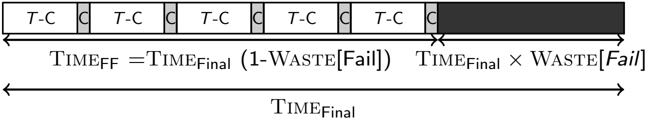
⇒ Instants when periods begin and failures strike are independent
⇒ Valid for all distribution laws, regardless of their particular shape

Waste due to failures

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

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

Total waste



$$\textsc{Waste} = \frac{\textsc{Time}_{\text{final}} - \textsc{Time}_{\text{base}}}{\textsc{Time}_{\text{final}}}$$

$$1 - \textsc{Waste} = (1 - \textsc{Waste}[FF])(1 - \textsc{Waste}[fail])$$

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

Waste minimization

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

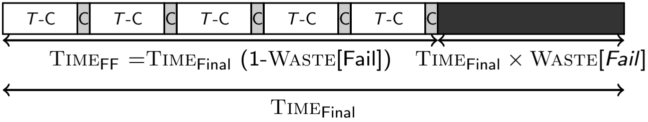
$$\textsc{Waste} = \frac{u}{T} + v + wT$$

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

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

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

Comparison with Young/Daly



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

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

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

Validity of the approach (1/3)

Technicalities

- $\mathbb{E}\left(N_{faults}\right) = \frac{\textsc{Time}_{\text{final}}}{\mu}$ 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 $\textsc{Waste}[FF] \leq 1$
- Enforce $D + R \leq \mu$ and bound T to get $\textsc{Waste}[fail] \leq 1$ 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?

(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

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

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

Lesson learnt?

(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	$\Rightarrow$ W TE$_{\text{opt}} = 17\%$
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 $= 100\%$

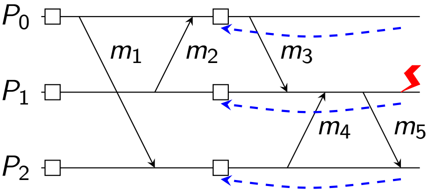
Outline

1. Introduction
   - Large-scale computing platforms
   - Faults and failures
2. Checkpointing
   - Coordinated checkpointing
   - Young/Daly's approximation
3. Models for faster checkpointing
   - Hierarchical checkpointing
   - In-memory checkpointing
   - Multilevel checkpointing
   - Checkpointing and prediction
   - Checkpointing and replication
4. Silent errors
   - Framework
   - ABFT for dense linear algebra kernels
   - Checkpointing and verification
5. Conclusion

Outline

Background: coordinated checkpointing protocols

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



☺ No risk of cascading rollbacks

☺ No need to log messages

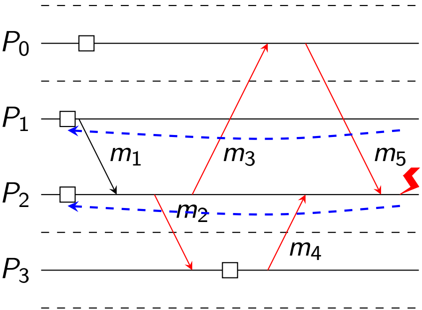
☹ All processors need to roll back

☹ Cost of synchronisation, I/O contention

☹ Rumor: May not scale to very large platforms

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

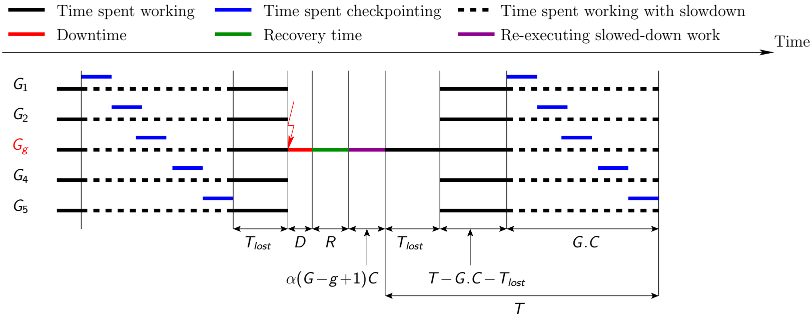


☹ Need to log inter-groups messages
- Slowdowns failure-free execution
- Increases checkpoint size/time

☺ Faster re-execution with logged messages

☺ Rumor: Should scale to very large platforms

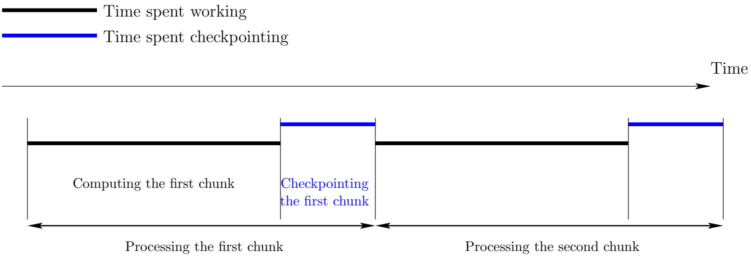
Hierarchical checkpointing



- Processors partitioned into G groups
- Each group includes q processors
- Inside each group: coordinated checkpointing
- Inter-group messages are logged

Four additional parameters $\alpha, \lambda, \rho, \beta$

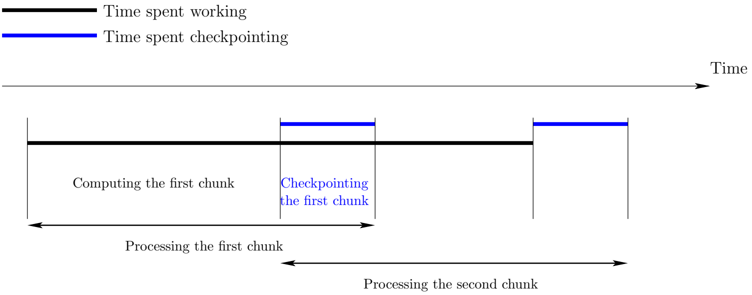
① Non-blocking checkpoint



Blocking model: checkpointing blocks all computations

Four additional parameters $\alpha, \lambda, \rho, \beta$

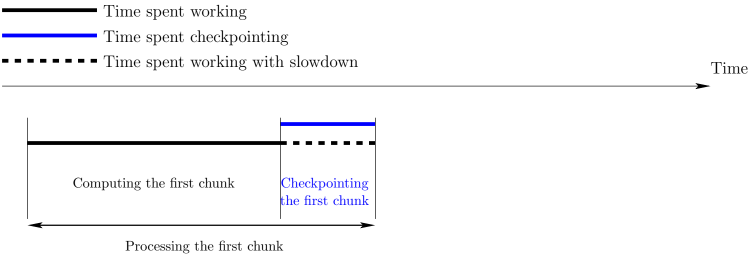
① Non-blocking checkpoint



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

Four additional parameters $\alpha, \lambda, \rho, \beta$

① Non-blocking checkpoint



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 ($0 \leq \alpha \leq 1$)

Four additional parameters $\alpha, \lambda, \rho, \beta$

② and ③ Impact of message logging 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: $\rho \approx 1.5$

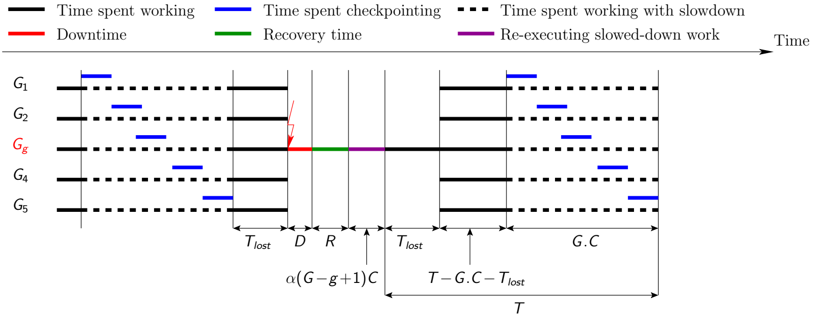
Four additional parameters $\alpha, \lambda, \rho, \beta$

④ Impact of message logging 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

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

Hierarchical checkpointing



Now we can compute the waste ☺

Three case studies

Coord-IO

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

Hierarch-IO

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

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

Hierarch-Port

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

Three applications

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

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	I/O Network Bandwidth (b_{io}) 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

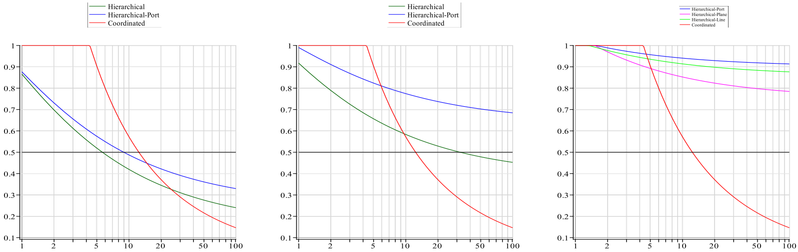
Name	Scenario	G ($C(q)$)	β for 2D-Stencil	β for Matrix-Product
Titan	Coord-IO	1 (2,048s)	/	/
	Hierarch-IO	136 (15s)	0.0001098	0.0004280
	Hierarch-Port	1,246 (1.6s)	0.0002196	0.0008561
K-Computer	Coord-IO	1 (14,688s)	/	/
	Hierarch-IO	296 (50s)	0.0002858	0.001113
	Hierarch-Port	17,626 (0.83s)	0.0005716	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

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



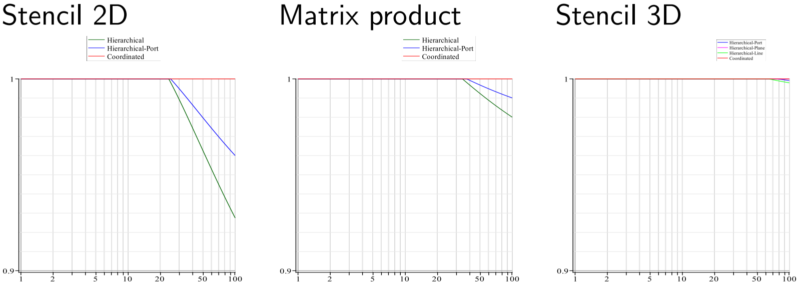
Stencil 2D

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

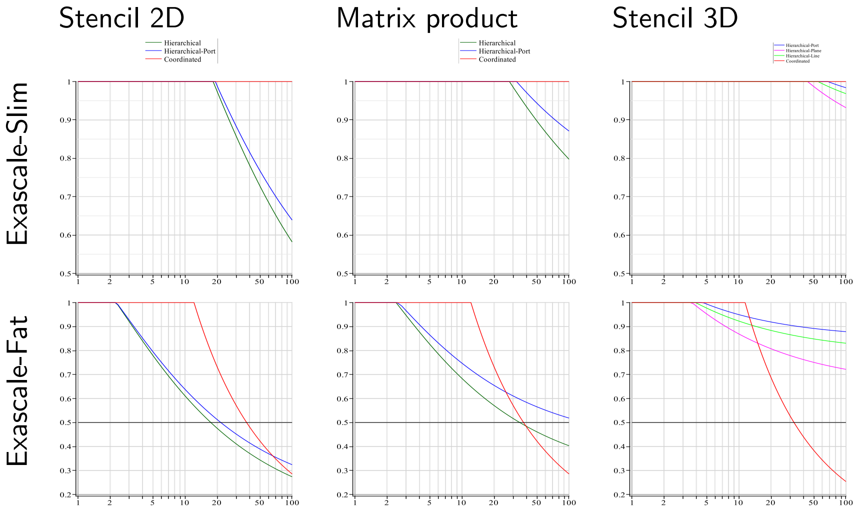
$\textsc{Waste} = 1$ for all scenarios!!!

Plotting formulas – Platform: Exascale

~~WASTE = [illegible] for all scenarios!!!~~

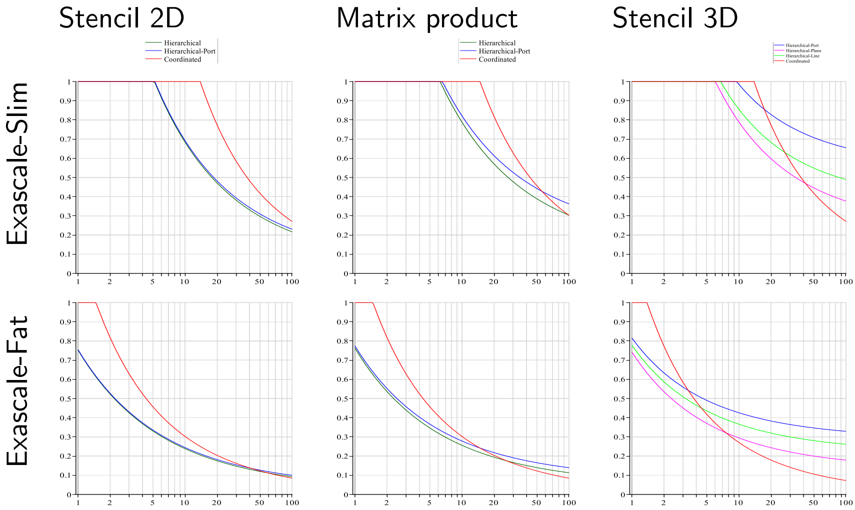
Goodbye Exascale?!

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



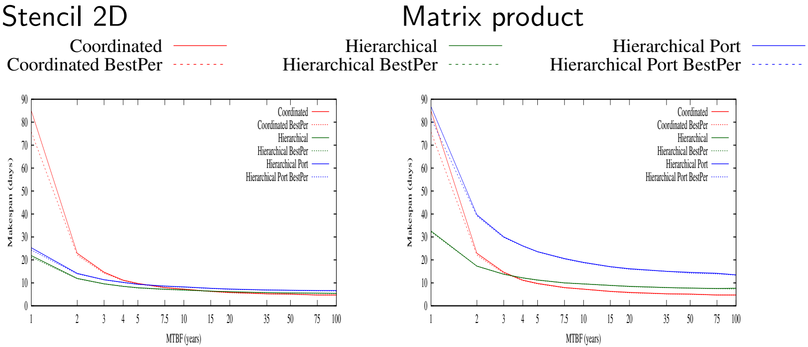
Waste as a function of processor MTBF μ_{ind}, $C = 1,000$

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



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

Simulations – Platform: Titan


Stencil 2D

Matrix product

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

Outline

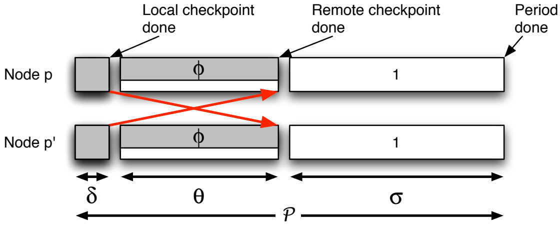
Motivation

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

- Stable storage: high cost

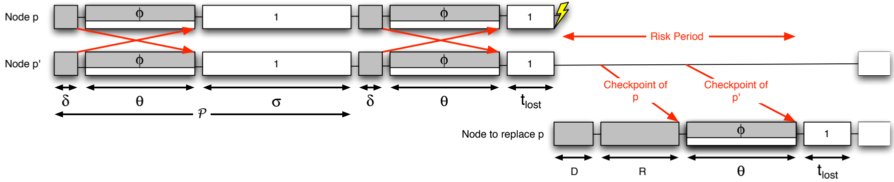
- Distributed in-memory storage:
  - Store checkpoints in local memory $\Rightarrow$ no centralized storage
    ☺ Much better scalability
  - Replicate checkpoints $\Rightarrow$ application survives single failure
    ☹ Still, risk of fatal failure in some (unlikely) scenarios

Double checkpoint algorithm (Kale et al., UIUC)



- 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

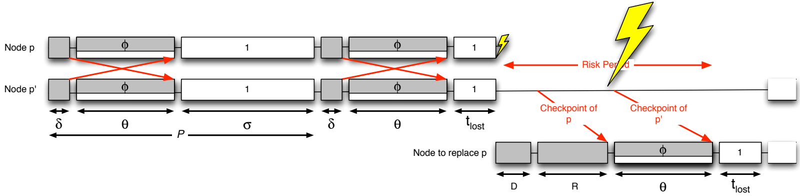
Failures



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

3 Models for faster checkpointing
- Hierarchical checkpointing
- In-memory checkpointing
- Multilevel checkpointing
- Checkpointing and prediction
- Checkpointing and replication

4 Silent errors
- Framework
- ABFT for dense linear algebra kernels
- Checkpointing and verification

5 Conclusion

Existing multi-level checkpoint toolkits

Scalable Checkpoint/Restart Library (SCR) – SC'10

① RAM disk / local disk

② Partner-copy / XOR encoding

③ Parallel File System (PFS), e.g., NFS

Fault Tolerance Interface (FTI) – SC'11

① Local disk: storing ckpt files in local disk

② Partner-copy: storing ckptt files in local disk & partner disk

③ Reed-Solomon encoding (RS-encoding)

④ Parallel File System (PFS), e.g., NFS

Outline

1. Introduction
   - Large-scale computing platforms
   - Faults and failures
2. Checkpointing
   - Coordinated checkpointing
   - Young/Daly's approximation
3. Models for faster checkpointing
   - Hierarchical checkpointing
   - In-memory checkpointing
   - Multilevel checkpointing
   - **Checkpointing and prediction**
   - Checkpointing and replication
4. Silent errors
   - Framework
   - ABFT for dense linear algebra kernels
   - Checkpointing and verification
5. 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

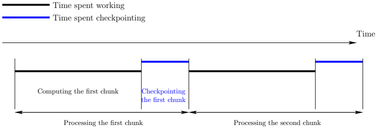
- *true positive*: predicted faults
- *false positive*: fault predictions that did not materialize as actual faults
- *false negative*: unpredicted faults

Algorithm

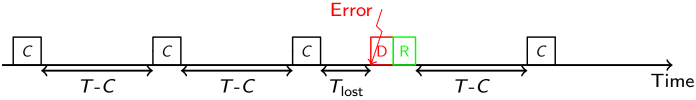
1. While no fault prediction is available:
   - checkpoints taken periodically with period T
2. When a fault is predicted at time t:
   - take a checkpoint ALAP (completion right at time t)
   - after the checkpoint, complete the execution of the period

Computing the waste

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



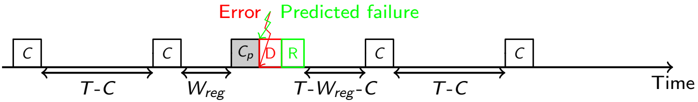
2. **Unpredicted faults:** $\frac{1}{\mu_{NP}}\left[D + R + \frac{T}{2}\right]$



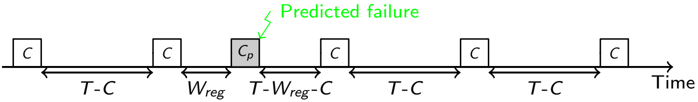
$$\textsc{Waste}[fail] = \frac{1}{\mu}\left[(1-r)\frac{T}{2} + D + R + \frac{r}{p}C\right] \Rightarrow T_{opt} \approx \sqrt{\frac{2\mu C}{1-r}}$$

Computing the waste

3. **Predictions:** $\frac{1}{\mu_P}\left[p(C+D+R)+(1-p)C\right]$



with actual fault (true positive)

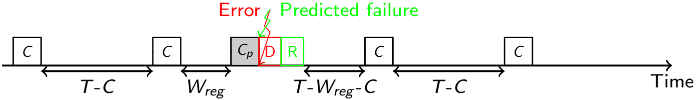


no actual fault (false negative)

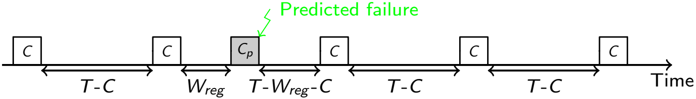
$$\text{WASTE}[fail]=\frac{1}{\mu}\left[(1-r)\frac{T}{2}+D+R+\frac{r}{p}C\right]\Rightarrow T_{opt}\approx\sqrt{\frac{2\mu C}{1-r}}$$

Computing the waste

③ **Predictions:** $\frac{1}{\mu_P}\left[p(C+D+R)+(1-p)C\right]$



with actual fault (true positive)



no actual fault (false negative)

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

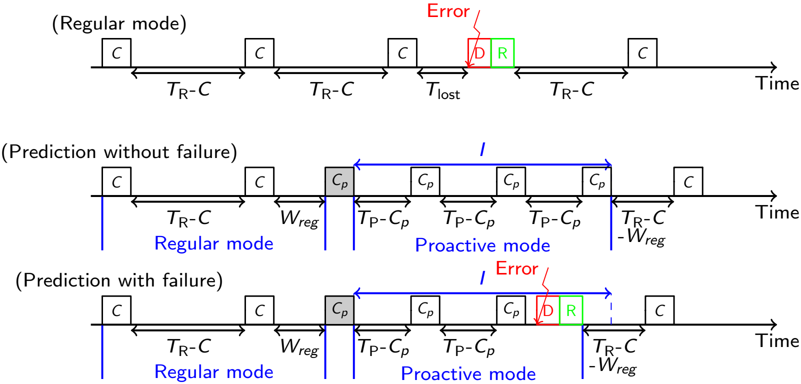
Refinements

- Use different value C_p for proactive checkpoints

- Avoid checkpointing too frequently for false negatives
  $\Rightarrow$ Only trust predictions with some fixed probability q
  $\Rightarrow$ 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
  $\Rightarrow$ Increase q when progressing
  $\Rightarrow$ Break-even point $\frac{C_p}{p}$

With prediction windows

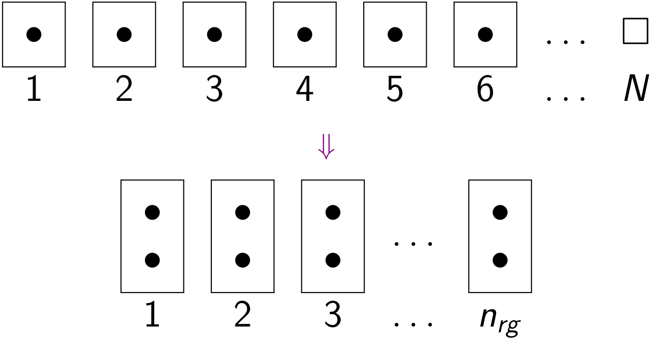


Gets too complicated! ☹

Outline

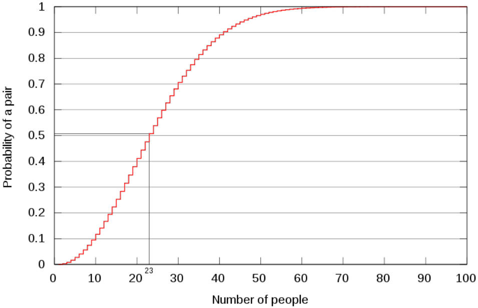
1. Introduction
   - Large-scale computing platforms
   - Faults and failures
2. Checkpointing
   - Coordinated checkpointing
   - Young/Daly's approximation
3. Models for faster checkpointing
   - Hierarchical checkpointing
   - In-memory checkpointing
   - Multilevel checkpointing
   - Checkpointing and prediction
   - Checkpointing and replication
4. Silent errors
   - Framework
   - ABFT for dense linear algebra kernels
   - Checkpointing and verification
5. Conclusion

PROCESS REPLICATION

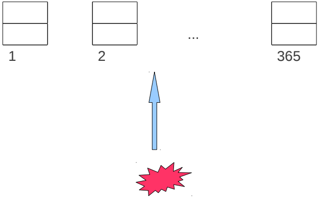


- Each process replicated $g \geq 2$ times → *replica-group*
- n_{rg} = number of replica-groups ($g \times n_{rg} = N$)
- Study for $g = 2$ by Ferreira et al., SC'2011

Analogy with birthday problem

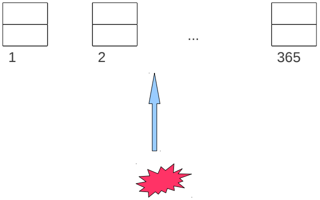


Analogy with birthday problem



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

Analogy with birthday problem

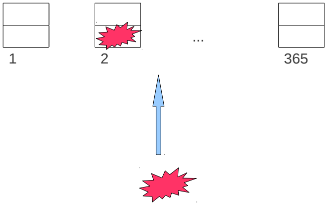


$n = n_{rg}$ bins, throw balls until one bin gets two balls

Expected number of balls to throw:

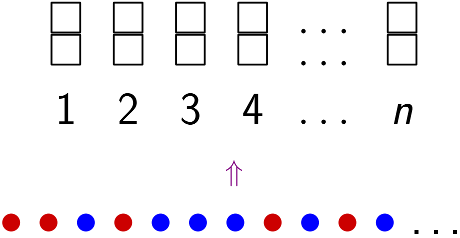
$$Birthday(n) = 1 + \int_0^{+\infty} e^{-x}(1 + x/n)^{n-1}dx$$

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 application down)
$MNFTI$ = expected number of balls to throw
until one bin gets one ball of each color

How can it help?

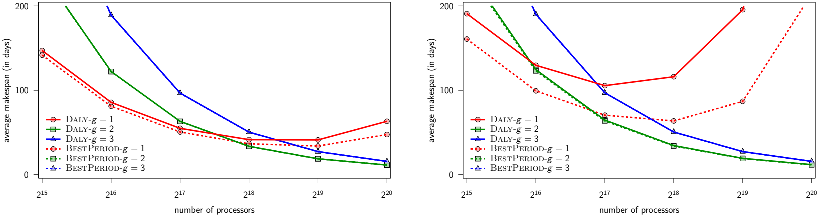
Trade-off

☹ By nature: replication → at most 50% machine efficiency
⇒ Reminds of TMR, *Triple Modular Redundancy*

☺ Allows to (virtually) increase MTBF dramatically

- fewer application failures
- larger checkpointing period
- less overhead

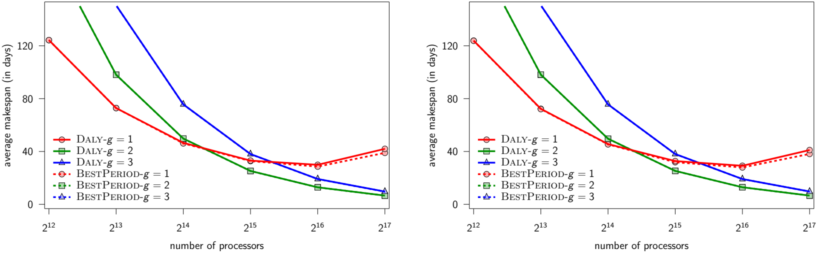
Simulations (1/3)



(a) $k = 0.70$ (b) $k = 0.50$

Weibull failures, $C = 600$ sec, $\mu_{\text{ind}} = 125$ years

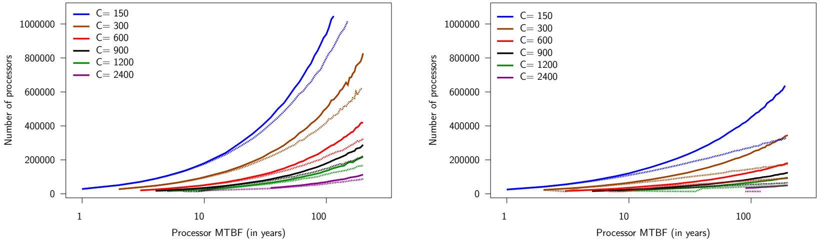
Simulations (2/3)



(a) LANL cluster 18 (b) LANL cluster 19

Log-based failures, $C = 600$ sec, $\mu_{\text{ind}} = 125$ years

Simulations (3/3)



(a) $k = 0.70$ (b) $k = 0.50$

Break-even point curves ($g = 2$), Weibull distributions

Replication better above curves!!!!!!

Outline

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

2 Checkpointing
- Coordinated checkpointing
- Young/Daly's approximation

3 Models for faster checkpointing
- Hierarchical checkpointing
- In-memory checkpointing
- Multilevel checkpointing
- Checkpointing and prediction
- Checkpointing and replication

4 Silent errors
- Framework
- ABFT for dense linear algebra kernels
- Checkpointing and verification

5 Conclusion

Outline

Definitions

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

Silent error detected only when the corrupt data is activated

- Includes some software faults, some hardware errors (soft errors in L1 cache), double bit flip
- Cannot always be corrected by ECC memory

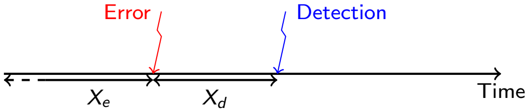
Quotes

- Soft Error: An unintended change in the state of an electronic device that alters the information that it stores without destroying its functionality, e.g. a bit flip caused by a cosmic-ray-induced neutron. (Hengartner et al., 2008)
- SDC occurs when incorrect data is delivered by a computing system to the user without any error being logged (Cristian Constantinescu, AMD)
- Silent errors are the black swan of errors (Marc Snir)

Application-specific methods

- ABFT: dense matrices / fail-stop, extended to sparse / silent. Limited to one error detection and/or correction in practice
- Asynchronous (chaotic) iterative methods (old work)
- Partial differential equations: use lower-order scheme as verification mechanism (detection only, Benson, Schmit and Schreiber)
- FT-GMRES: inner-outer iterations (Hoemmen and Heroux)
- PCG: orthogonalization check every k iterations, re-orthogonalization if problem detected (Sao and Vuduc)
- . . . Many others

General-purpose approach



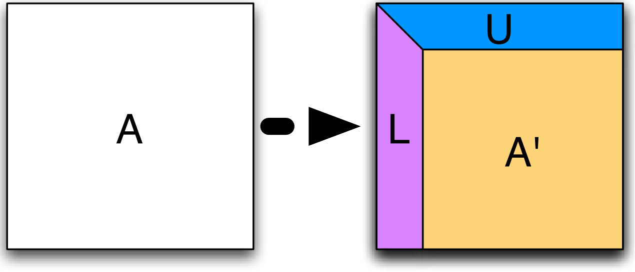
Error and detection latency

- Last checkpoint may have saved an already corrupted state

- Saving *k* checkpoints (Lu, Zheng and Chien):
    1. Which checkpoint to roll back to?
    2. Critical failure when all live checkpoints are invalid

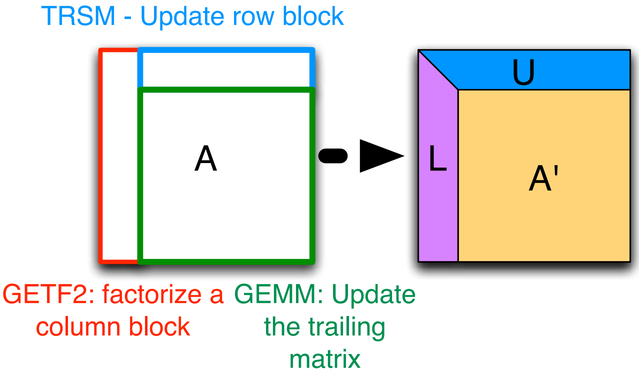
Outline

Tiled LU factorization



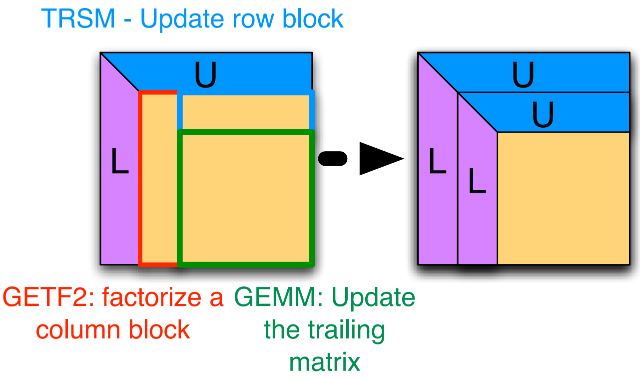
- Solve $A \cdot x = b$ (hard)
- Transform A into a LU factorization
- Solve $L \cdot y = B \cdot b$, then $U \cdot x = y$

Tiled LU factorization



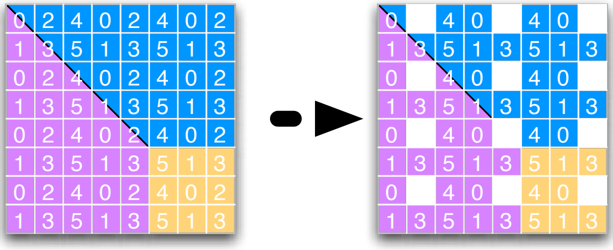
- Solve $A \cdot x = b$ (hard)
- Transform A into a LU factorization
- Solve $L \cdot y = B \cdot b$, then $U \cdot x = y$

Tiled LU factorization



- 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



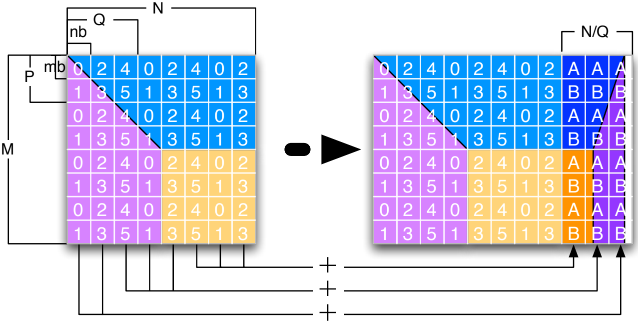
Failure of rank 2

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

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

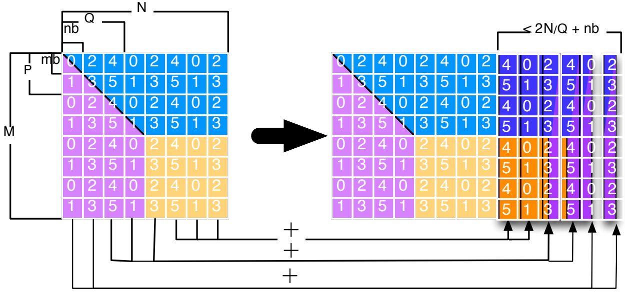
- 2D Block Cyclic Distribution (here 2 × 3)
- A single failure ⇒ many data lost

Algorithm Based Fault Tolerant LU decomposition



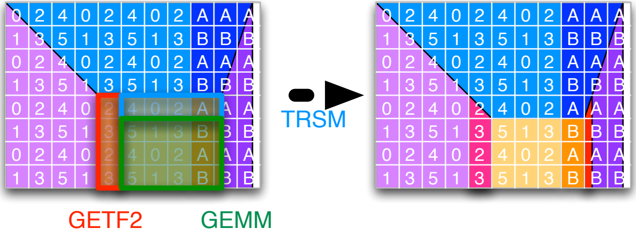
- Checksum: invertible operation on row/column data
  - Checksum replication avoided by dedicating additional computing resources to checksum storage

Algorithm Based Fault Tolerant LU decomposition



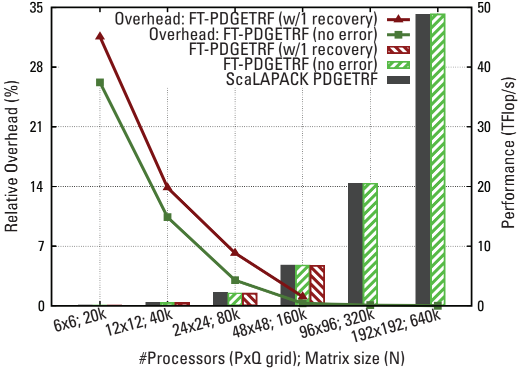
- Checksum: invertible operation on row/column data
  - Checksum blocks are doubled, to allow recovery when data and checksum are lost together (no extra resource needed)

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 Checkpointing
- Coordinated checkpointing
- Young/Daly's approximation

3 Models for faster checkpointing
- Hierarchical checkpointing
- In-memory checkpointing
- Multilevel checkpointing
- Checkpointing and prediction
- Checkpointing and replication

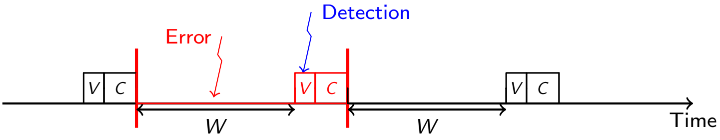
4 Silent errors
- Framework
- ABFT for dense linear algebra kernels
- Checkpointing and verification

5 Conclusion

Coupling checkpointing and verification

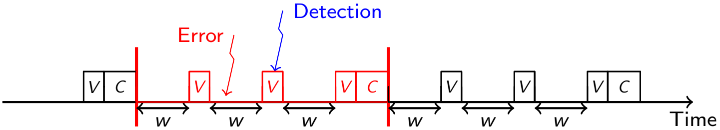
- Verification mechanism of cost V
- Silent errors detected only when verification is executed
- Approach agnostic of the nature of verification mechanism (checksum, error correcting code, coherence tests, etc)
- Fully general-purpose
  (application-specific information, if available, can always be used to decrease V)

Base pattern (and revisiting Young/Daly)



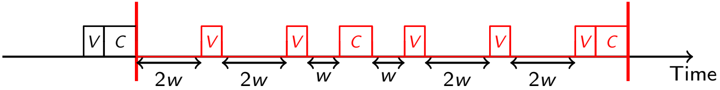
	Fail − stop(classical)	Silent errors
Pattern	$T = W + C$	$S = W + V + C$
$\textsc{Waste}[FF]$	$\frac{C}{T}$	$\frac{V+C}{S}$
$\textsc{Waste}[fail]$	$\frac{1}{\mu}(D + R + \frac{W}{2})$	$\frac{1}{\mu}(R + W + V)$
Optimal	$T_{\text{opt}} = \sqrt{2C\mu}$	$S_{\text{opt}} = \sqrt{(C + V)\mu}$
$\textsc{Waste}_{\text{opt}}$	$\sqrt{\frac{2C}{\mu}}$	$2\sqrt{\frac{C+V}{\mu}}$

With $p = 1$ checkpoint and $q = 3$ verifications



Base Pattern	$p = 1, q = 1$	$\text{WASTE}_{\text{opt}} = 2\sqrt{\frac{C+V}{\mu}}$
New Pattern	$p = 1, q = 3$	$\text{WASTE}_{\text{opt}} = 2\sqrt{\frac{4(C+3V)}{6\mu}}$

With p checkpoints and q verifications, $p \leq q$



- BalancedAlgorithm optimal when $C, R, V \ll \mu$
- Keep only 2 checkpoints in memory/storage
- Closed-form formula for $\text{Waste}_{\text{opt}}$
- Given C and V, choose optimal pattern

Outline

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'13 tutorial with Thomas Hérault. Available at

`http://graal.ens-lyon.fr/~yrobert/`

Thanks

INRIA & ENS Lyon

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

Univ. Tennessee Knoxville

- George Bosilca
- Aurélien Bouteiller
- Jack Dongarra
- Thomas Hérault (joint tutorial at SC'13)

Others

- Franck Cappello, Argonne National Lab.
- Henri Casanova, Univ. Hawai'i