Fault-Tolerance Techniques for Computing at Scale

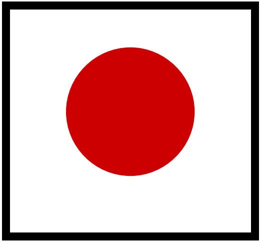
Yves Robert

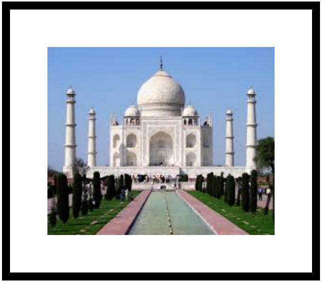
ENS Lyon & Institut Universitaire de France
University of Tennessee Knoxville

yves.robert@ens-lyon.fr

http://graal.ens-lyon.fr/~yrobert/hipc2014.pdf

HiPC – December 20, 2014





Outline

Outline

Exascale platforms (courtesy Jack 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

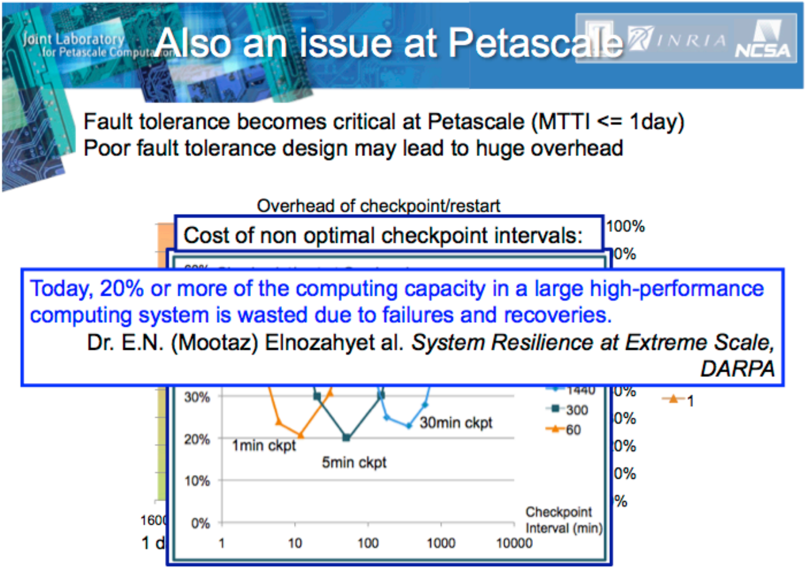
Exascale platforms

- 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

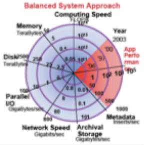
More nodes ⇒ Shorter MTBF (Mean Time Between Failures)

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

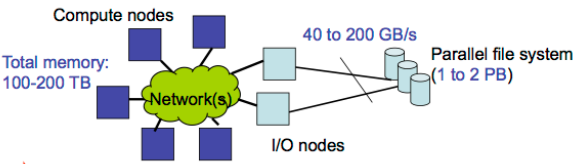


Even for today's platforms (courtesy F. Cappello)

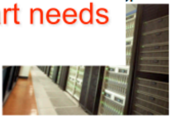
Classic approach for FT: Checkpoint-Restart



Typical "Balanced Architecture" for PetaScale Computers



RoadRunner



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

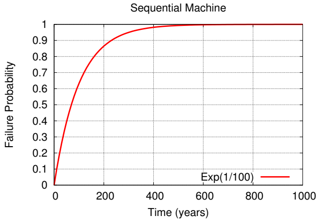
Systems	Perf.	Ckpt time	Source
RoadRunner	1PF	~20 min.	Panasas
LLNL BG/L	500 TF	>20 min.	LLNL
LLNL Zeus	11TF	26 min.	LLNL
YYY BG/P	100 TF	~30 min.	YYY

LLNL BG/L

Error sources

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

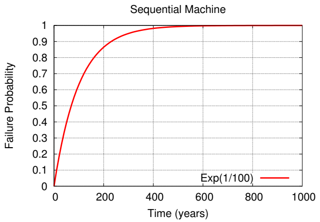
Failure distributions: (1) Exponential



Exp(λ): 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 $= \frac{1}{\lambda}$

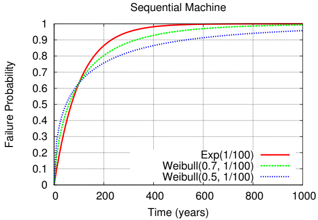
Failure distributions: (1) Exponential



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) $\mu = \mathbb{E}(X) = \frac{1}{\lambda}$

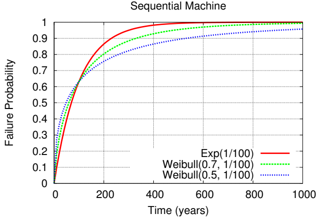
Failure distributions: (2) Weibull



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 $Weibull(k, \lambda)$ 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: (3) with several processors

- If the MTBF is μ with one processor, what is its value with p processors?
- Processor (or node): any entity subject to failures
  $\Rightarrow$ approach agnostic to granularity
- Platform failure distribution
  $\Rightarrow$ superposition of p IID processor distributions

Failure distributions: (3) with several processors

- If th
  what

Too complicated ! ☹ ☹ ☹

- Processor (or ode): any entity subject to lures
  $\Rightarrow$ approa agnostic to granularity
- Platfo failure distribution
  $\Rightarrow$ perposition of p IID processor distributions

Failure distributions: (3) with several processors

Theorem:

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

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`)

Outline

1. Introduction
2. Checkpointing
   - Coordinated checkpointing
   - Young/Daly's approximation
   - Assessing protocols at scale
   - In-memory checkpointing
   - Failure Prediction
   - Replication
3. ABFT for dense linear algebra kernels
4. Silent errors
5. Conclusion

Outline

Maintaining redundant information

Goal

- General Purpose Fault Tolerance Techniques: work despite the application behavior
- Two adversaries: Failures & Application
- Use automatically computed redundant information
  - At given instants: checkpoints
  - At any instant: replication
  - Or 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

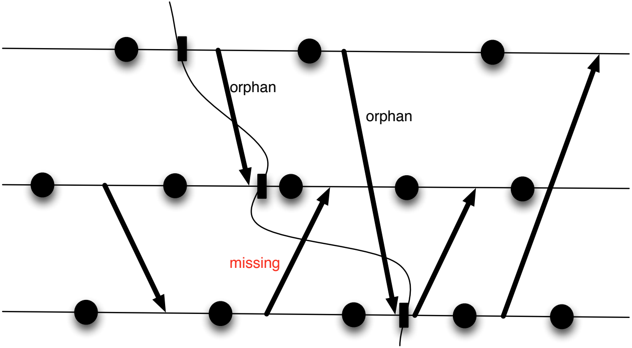
- local memory
- local disk (SSD, HDD)
- remote disk
  - Scalable Checkpoint Restart Library
    `http://scalablecr.sourceforge.net`

Checkpoint is valid when finished on reliable storage

Distributed memory storage

- In-memory checkpointing
- Disk-less checkpointing

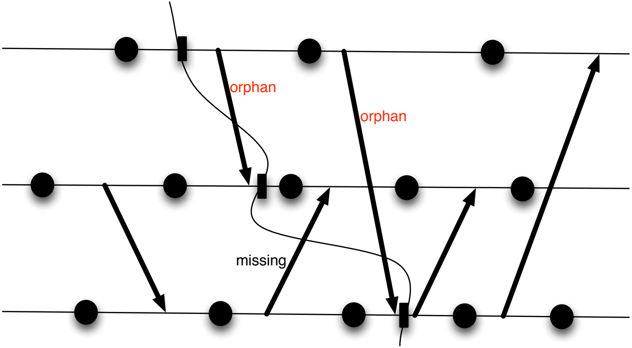
Coordinated checkpointing



Definition (Missing Message)

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

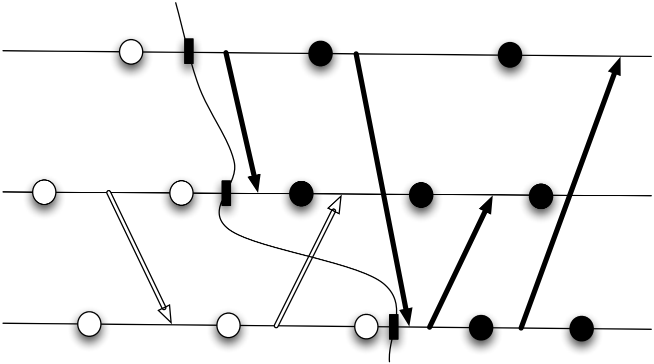
Coordinated checkpointing



Definition (Orphan Message)

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

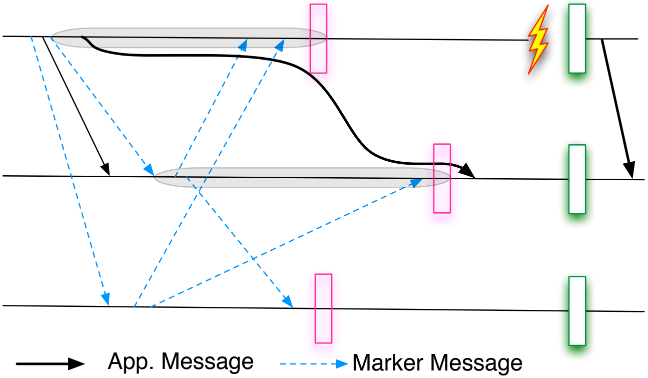
Coordinated checkpointing



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)

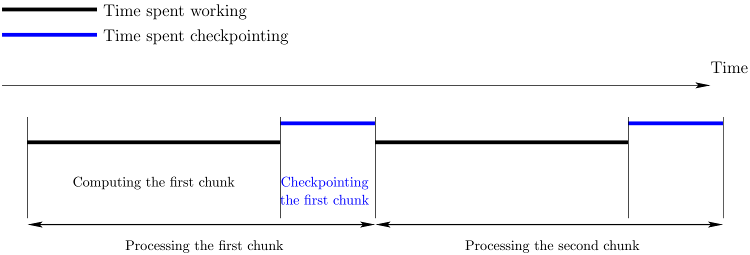
Coordinated checkpointing



- Silences the network during checkpoint
- Missing messages recorded

Outline

Periodic checkpointing



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

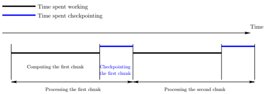
Framework

- 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 and MTBF $\mu = \frac{\mu_{ind}}{p}$
  - coordinated checkpointing
  - tightly-coupled application
  - progress $\Leftrightarrow$ all processors available

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

Waste: fraction of time not spent for useful computations

Waste in fault-free execution



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

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

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

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

Waste due to failures

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

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

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

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

T_{lost}?

Waste due to failures

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

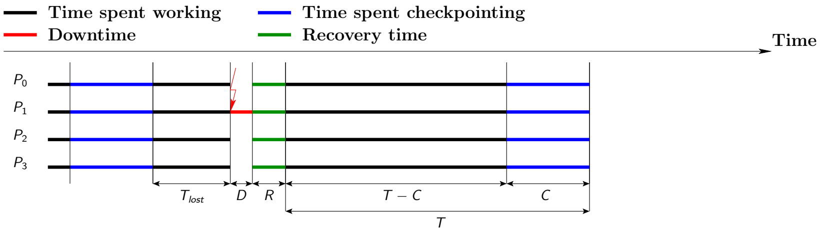
$$\textsc{Time}_{\mathsf{final}} = \textsc{Time}_{\mathsf{FF}} + N_{faults} \times T_{\mathsf{lost}}$$

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

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

T_{lost}?

Computing T_{lost}



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

Rationale

⇒ Instants when periods begin and failures strike are independent

⇒ Approximation used for all distribution laws

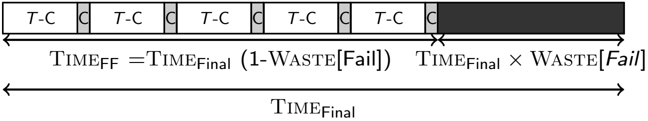
⇒ Exact for Exponential and uniform distributions

Waste due to failures

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

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

Total waste



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

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

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

Waste minimization

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

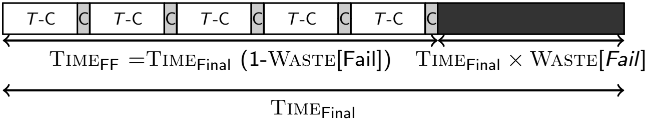
$$\textsc{Waste} = \frac{u}{T} + v + wT$$

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

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

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

Comparison with Young/Daly



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

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

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

Validity of the approach

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

Wrap up

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

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

- Approach surprisingly robust ☺

Lesson learnt for fail-stop failures

(Not so) Secret data

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

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

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

Lesson learnt for fail-stop failures

([illegible] so) Secret data

- Tsuba[illegible] 962 failures during last 18 months so [illegible] 13 hrs
- Blue Waters: 2-[illegible]de failures per day
- Titan: a few failures pe[illegible]
- Tianhe[illegible]

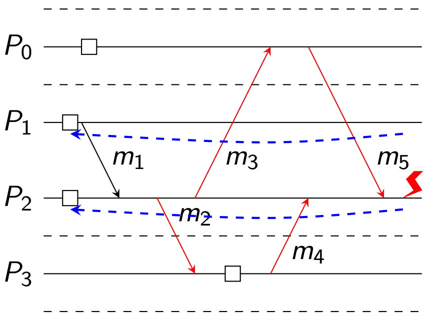
Exascale $\neq$ Petascale $\times 1000$
Need more reliable components
Need to checkpoint faster

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

Outline

Hierarchical checkpointing

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



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

☺ Faster re-execution with logged messages

Which checkpointing protocol to use?

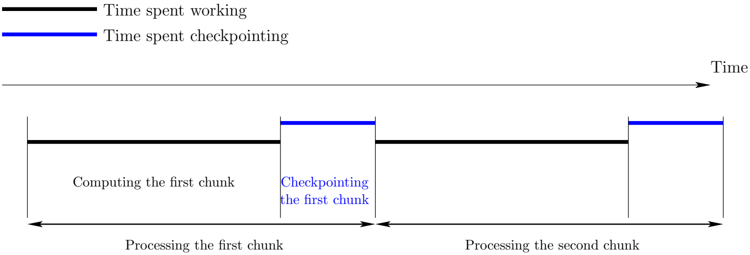
Coordinated checkpointing

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

Hierarchical checkpointing

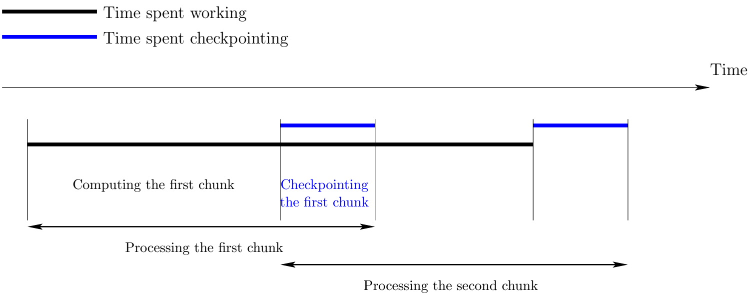
- ☹ Need to log inter-groups messages
  - Slowdowns failure-free execution
  - Increases checkpoint size/time
- ☺ Only processors from failed group need to roll back
- ☺ Faster re-execution with logged messages
- ☺ Rumor: Should scale to very large platforms

Blocking vs. non-blocking



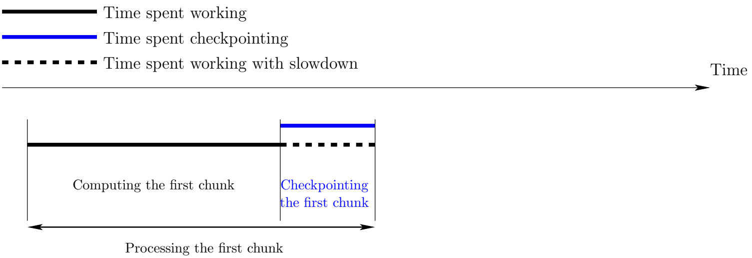
Blocking model: checkpointing blocks all computations

Blocking vs. non-blocking



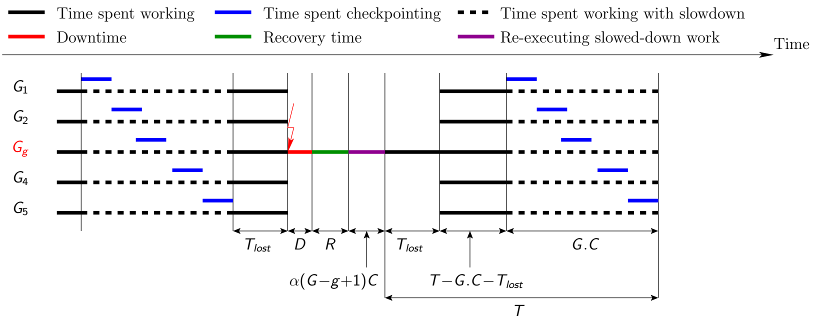
Non-blocking model: checkpointing has no impact on computations (e.g., first copy state to RAM, then copy RAM to disk)

Blocking vs. non-blocking



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

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

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

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

Accounting for message logging: Impact on checkpoint size

- Inter-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

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

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

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

Four platforms: basic characteristics

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

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

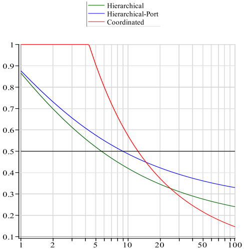
Checkpoint time

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

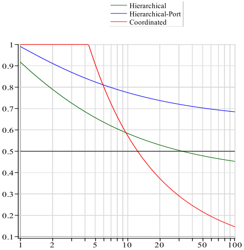
- Large time to dump the memory
- Using 1%C
- Comparing with 0.1%C for exascale platforms
- $\alpha = 0.3$, $\lambda = 0.98$ and $\rho = 1.5$

Plotting formulas – Platform: Titan

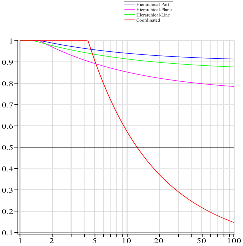
Stencil 2D



Matrix product



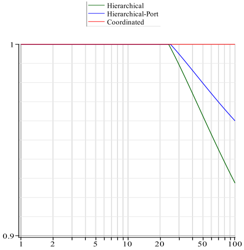
Stencil 3D



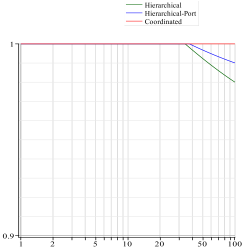
Waste as a function of processor MTBF μ_{ind}

Platform: K-Computer

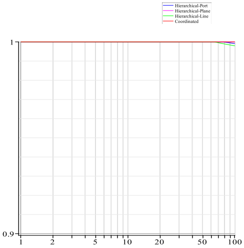
Stencil 2D



Matrix product



Stencil 3D



Waste as a function of processor MTBF μ_{ind}

Plotting formulas – Platform: Exascale

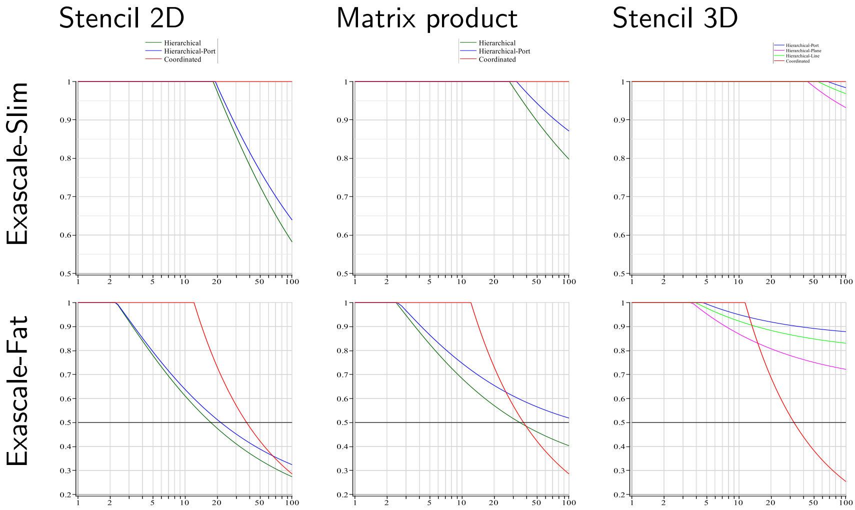
$\textsc{Waste} = 1$ for all scenarios!!!

Plotting formulas – Platform: Exascale

WASTE = 1 for all scenarios!!!

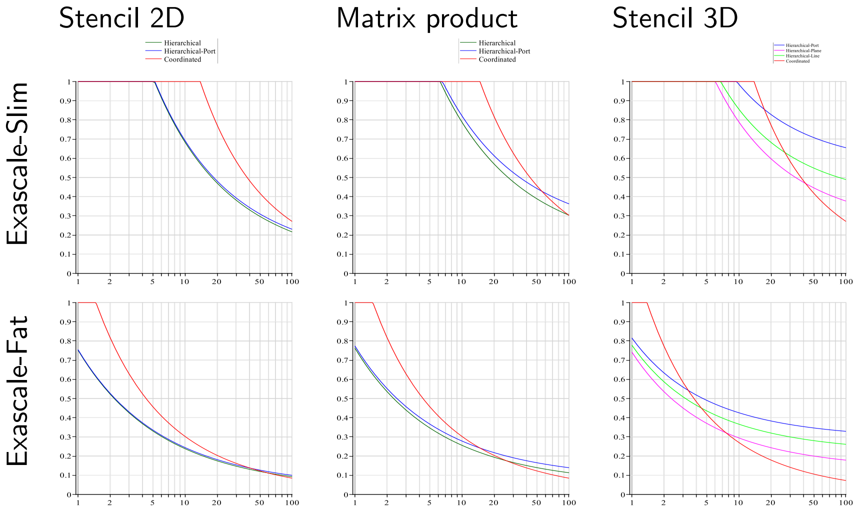
Goodbye Exascale?!

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



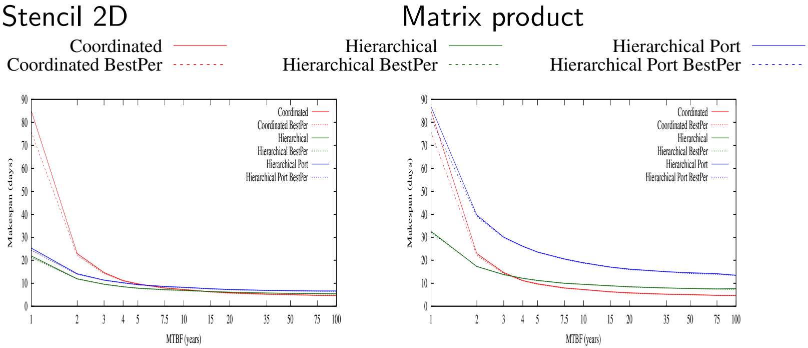
Waste as a function of processor MTBF μ_{ind}, $C = 1,000$

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



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

Simulations – Platform: Titan

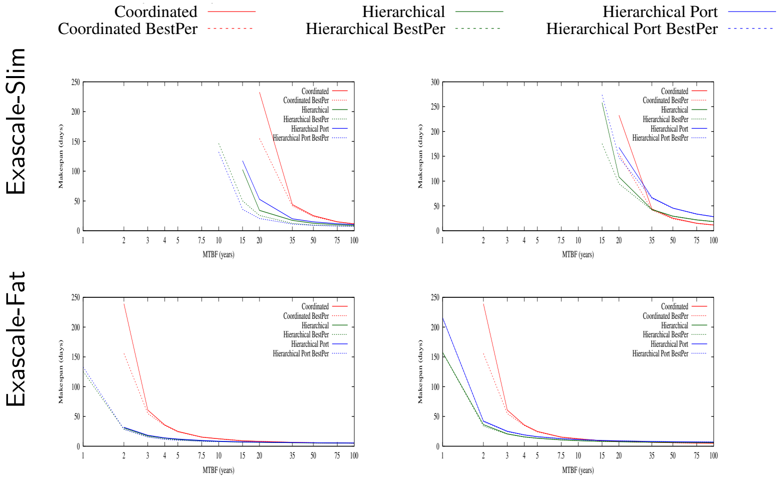


Makespan (in days) as a function of processor MTBF μ_{ind}

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

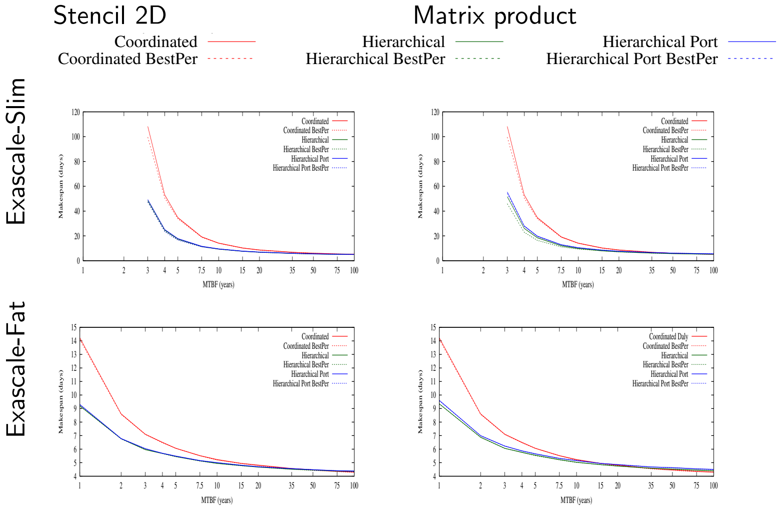
Stencil 2D

Matrix product



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

Simulations – Platform: Exascale with $C = 100$



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

Outline

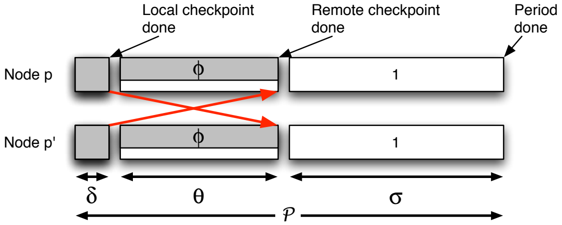
Motivation

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

- Stable storage: high cost

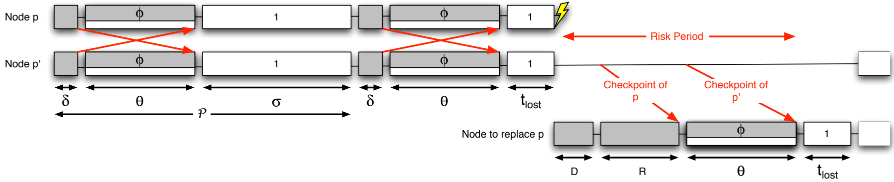
- Distributed in-memory storage:
  - Store checkpoints in local memory $\Rightarrow$ no centralized storage
    ☺ Much better scalability
  - Replicate checkpoints $\Rightarrow$ application survives single failure
    ☹ Still, risk of fatal failure in some (unlikely) scenarios

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

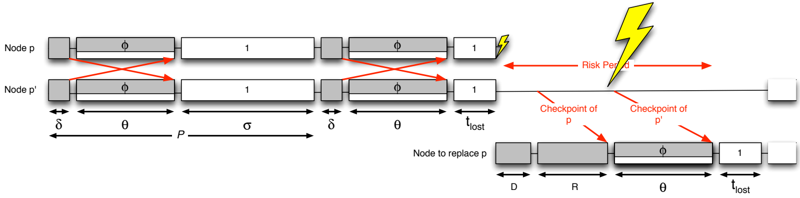
Failures



- 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

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

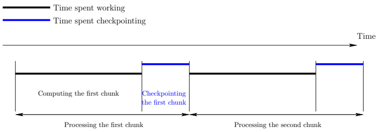
- *true positive*: predicted faults
- *false positive*: fault predictions that did not materialize as actual faults
- *false negative*: unpredicted faults

Algorithm

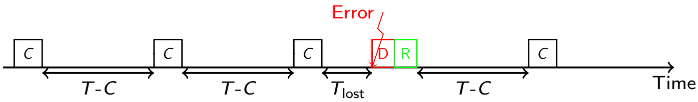
1. While no fault prediction is available:
   - checkpoints taken periodically with period T
2. When a fault is predicted at time t:
   - take a checkpoint ALAP (completion right at time t)
   - after the checkpoint, complete the execution of the period

Computing the waste

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

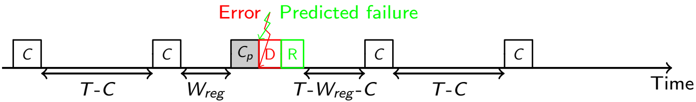


2. **Unpredicted faults:** $\frac{1}{\mu_{NP}}\left[D + R + \frac{T}{2}\right]$

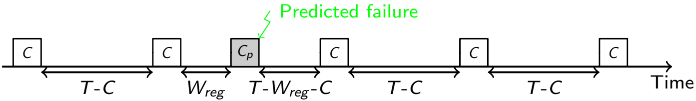


Computing the waste

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



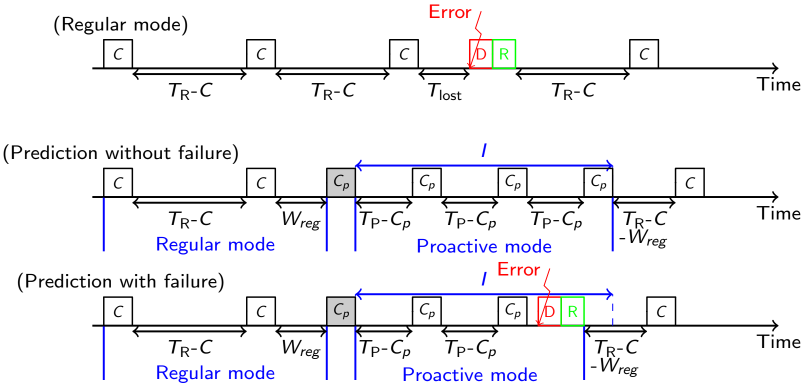
with actual fault (true positive)



no actual fault (false negative)

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

With prediction windows



Gets too complicated! ☹

Outline

Replication

- Systematic replication: efficiency $< 50\%$
- Can replication+checkpointing be more efficient than checkpointing alone?
- Study by Ferreira et al. [SC'2011]: yes

Model by Ferreira et al. [SC' 2011]

- Parallel application comprising N processes
- Platform with $p_{total} = 2N$ processors
- Each process replicated $\rightarrow$ N *replica-groups*
- When a replica is hit by a failure, it is not restarted
- Application fails when both replicas in one replica-group have been hit by failures

The birthday problem

Classical formulation

What is the probability, in a set of m people, that two of them have same birthday ?

Relevant formulation

What is the average number of people required to find a pair with same birthday?

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

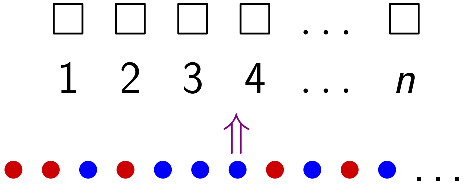
The analogy

Two people with same birthday

$\equiv$

Two failures hitting same replica-group

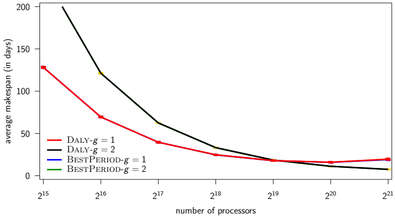
Correct analogy



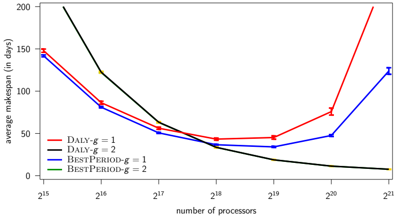
$N = n_{rg}$ bins, red and blue balls

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

Forward-Recovery

Backward Recovery

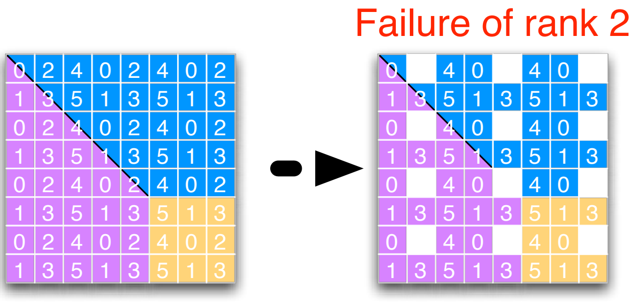
- Rollback / Backward Recovery: returns in the history to recover from failures.
- Spends time to re-execute computations
- Rebuilds states already reached
- Typical: checkpointing techniques

Forward-Recovery

Forward Recovery

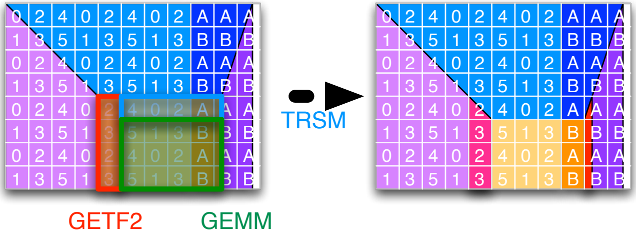
- Forward Recovery: proceeds without returning.
- Pays additional costs during (failure-free) computation to maintain consistent redundancy
- Or pays additional computations when failures happen
- General technique: Replication
- Application-Specific techniques: Iterative algorithms with fixed point convergence, ABFT, ...

Tiled LU factorization



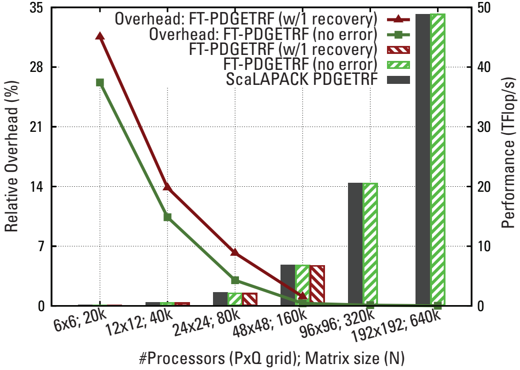
- 2D Block Cyclic Distribution (here 2×3)
- A single failure $\Rightarrow$ many data lost

Algorithm Based Fault Tolerant LU decomposition



- Checksum: invertible operation on row/column data
  - 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

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)

Should we be afraid? (courtesy Al Geist)

Fear of the Unknown

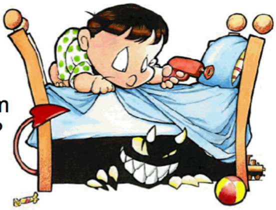
Hard errors – permanent component failure either HW or SW (hung or crash)

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?



Probability distributions for silent errors

?

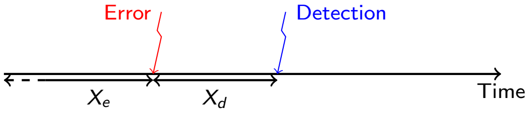
Theorem: $\mu_p = \frac{\mu_{\text{ind}}}{p}$ for arbitrary distributions

Probability distributions for silent errors

?

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

General-purpose approach

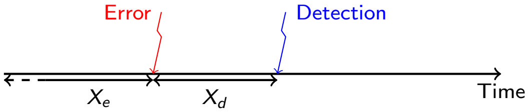


Error and detection latency

- Last checkpoint may have saved an already corrupted state

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

General-purpose approach



Error and detection latency

- Last checkpoint may have saved an already corrupted state
- Saving k checkpoints (Lu, Zheng and Chien):
  1. Critical failure when all live checkpoints are invalid
     Assume unlimited storage resources
  2. Which checkpoint to roll back to?
     Assume verification mechanism

Limitation of the model

It is not clear how to detect when the error has occurred (hence to identify the last valid checkpoint) ☹ ☹ ☹

Need a verification mechanism to check the correctness of the checkpoints. This has an additional cost!

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)

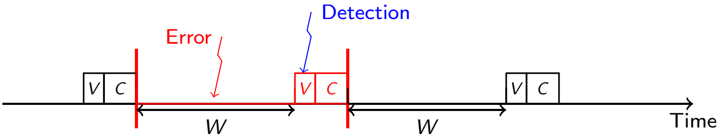
On-line ABFT scheme for PCG

Compute $r^{(0)} = b - Ax^{(0)}, z^{(0)} = M^{-1}r^{(0)}, p^{(0)} = z^{(0)}$,
    and $\rho_0 = {r^{(0)}}^T z^{(0)}$ for some initial guess $x^{(0)}$
checkpoint: A, M, and b
for $i = 0, 1, \ldots$
    if ((i>0) and ($i\%d = 0$))
        if ($\frac{{p^{(i+1)}}^T q^{(i)}}{||p^{(i+1)}||.||q^{(i)}||} > 10^{-10}$
          or $\frac{||r^{(i+1)} + Ax^{(i+1)} - b||}{||b||.||A||} > 10^{-10}$)
            recover: A, M, b, i, ρ_i,
                $p^{(i)}$, $x^{(i)}$, and $r^{(i)}$.
        else if ($i\%(cd) = 0$)
            checkpoint: i, ρ_i, $p^{(i)}$, and $x^{(i)}$
        endif
    endif
    $q^{(i)} = Ap^{(i)}$
    $\alpha_i = \rho_i / {p^{(i)}}^T q^{(i)}$
    $x^{(i+1)} = x^{(i)} + \alpha_i p^{(i)}$
    $r^{(i+1)} = r^{(i)} - \alpha_i q^{(i)}$
    solve $Mz^{(i+1)} = r^{(i+1)}$, where $M = M^T$
    $\rho_{i+1} = {r^{(i+1)}}^T z^{(i+1)}$
    $\beta_i = \rho_{i+1} / \rho_i$
    $p^{(i+1)} = z^{(i+1)} + \beta_i p^{(i)}$
    check convergence; continue if necessary
end

Zizhong Chen, PPoPP'13

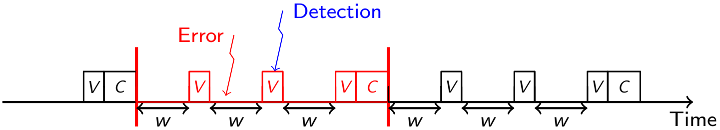
- Iterate PCG
  Cost: SpMV, preconditioner solve, 5 linear kernels
- Detect soft errors by checking orthogonality and residual
- Verification every *d* iterations
  Cost: scalar product+SpMV
- Checkpoint every *c* iterations
  Cost: three vectors, or two vectors + SpMV at recovery
- Experimental method to choose *c* and *d*

Base pattern (and revisiting Young/Daly)



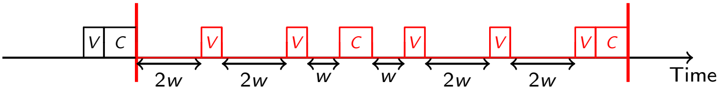
	Fail-stop (classical)	Silent errors
Pattern	$T = W + C$	$S = W + V + C$
$\textsc{Waste}[FF]$	$\frac{C}{T}$	$\frac{V+C}{S}$
$\textsc{Waste}[fail]$	$\frac{1}{\mu}(D + R + \frac{W}{2})$	$\frac{1}{\mu}(R + W + V)$
Optimal	$T_{\text{opt}} = \sqrt{2C\mu}$	$S_{\text{opt}} = \sqrt{(C+V)\mu}$
$\textsc{Waste}[opt]$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

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



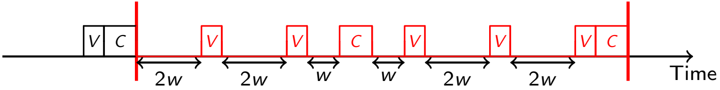
Base Pattern	$p = 1, q = 1$	$\textsc{Waste}[opt] = 2\sqrt{\frac{C+V}{\mu}}$
New Pattern	$p = 1, q = 3$	$\textsc{Waste}[opt] = 2\sqrt{\frac{4(C+3V)}{6\mu}}$

BalancedAlgorithm



- p checkpoints and q verifications, $p \leq q$
- $p = 2,\ q = 5,\ S = 2C + 5V + W$
- $W = 10w$, six chunks of size w or $2w$
- May store invalid checkpoint (error during third chunk)
- After successful verification in fourth chunk, preceding checkpoint is valid
- Keep only two checkpoints in memory and avoid any fatal failure

BalancedAlgorithm



① (proba $2w/W$) $T_{\text{lost}} = R + 2w + V$

② (proba $2w/W$) $T_{\text{lost}} = R + 4w + 2V$

③ (proba w/W) $T_{\text{lost}} = 2R + 6w + C + 4V$

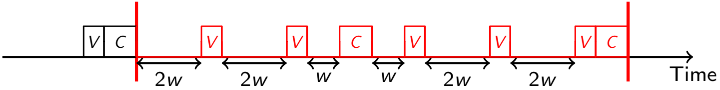
④ (proba w/W) $T_{\text{lost}} = R + w + 2V$

⑤ (proba $2w/W$) $T_{\text{lost}} = R + 3w + 2V$

⑥ (proba $2w/W$) $T_{\text{lost}} = R + 5w + 3V$

$$\textsc{Waste}[opt] \approx 2\sqrt{\frac{7(2C+5V)}{20\mu}}$$

Results



- BALANCEDALGORITHM optimal when $C, R, V \ll \mu$
- Keep only 2 checkpoints in memory/storage
- Closed-form formula for WASTE[*opt*]
- Given C and V, choose optimal pattern
- Gain of up to 20% over base pattern

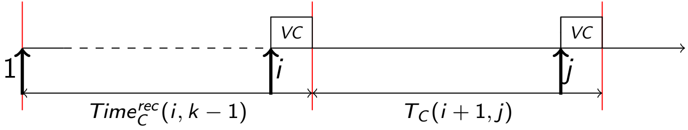
Application-specific methods

- 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

Dynamic programming for linear chains of tasks

- $\{T_1, T_2, \ldots, T_n\}$: linear chain of n tasks
- Each task T_i fully parametrized:
  - w_i computational weight
  - C_i, R_i, V_i : checkpoint, recovery, verification
- Error rates:
  - λ^F rate of fail-stop errors
  - λ^S rate of silent errors

VC-ONLY



$$\min_{0 \leq k < n} Time_C^{rec}(n, k)$$

$$Time_C^{rec}(j, k) = \min_{k \leq i < j} \{ Time_C^{rec}(i, k-1) + T_C^{SF}(i+1, j) \}$$

$$T_C^{SF}(i,j) = p_{i,j}^F \left(T_{lost_{i,j}} + R_{i-1} + T_C^{SF}(i,j) \right)$$
$$+ \left(1 - p_{i,j}^F\right) \left(\sum_{\ell=i}^{j} w_\ell + V_j + p_{i,j}^S \left(R_{i-1} + T_C^{SF}(i,j) \right) + \left(1 - p_{i,j}^S\right) C_j \right)$$

Extensions

- VC-ONLY and VC+V
- Different speeds with DVFS, different error rates
- Different execution modes
- Optimize for time or for energy consumption

Current research

- Use verification to correct some errors (ABFT)
- Imprecise verifications (a.k.a. recall and prediction)

Outline

A few questions

Silent errors

- Error rate? MTBE?
- Selective reliability?
- New algorithms beyond iterative? matrix-product, FFT, ...

Resilient research on resilience

Models needed to assess techniques at scale
without bias ☺

Conclusion

General Purpose Fault Tolerance

- 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'14 tutorial. Available at
`http://graal.ens-lyon.fr/~yrobert/`

Bibliography

Exascale
• Toward Exascale Resilience, Cappello F. et al., IJHPCA 23, 4 (2009)
• The International Exascale Software Roadmap, Dongarra, J., Beckman, P. et al., IJHPCA 25, 1 (2011)

ABFT Algorithm-based fault tolerance applied to high performance computing, Bosilca G. et al., JPDC 69, 4 (2009)

Coordinated Checkpointing Distributed snapshots: determining global states of distributed systems, Chandy K.M., Lamport L., ACM Trans. Comput. Syst. 3, 1 (1985)

Message Logging A survey of rollback-recovery protocols in message-passing systems, Elnozahy E.N. et al., ACM Comput. Surveys 34, 3 (2002)

Replication Evaluating the viability of process replication reliability for exascale systems, Ferreira K. et al, SC'2011

Models
• Checkpointing strategies for parallel jobs, Bougeret M. et al., SC'2011
• Unified model for assessing checkpointing protocols at extreme-scale, Bosilca G et al., INRIA RR-7950, 2012

Thanks

INRIA & ENS Lyon

- Anne Benoit & Frédéric Vivien
- PhD students (Guillaume Aupy, Aurélien Cavelan, Hongyang Sun, Dounia Zaidouni)

Univ. Tennessee Knoxville

- George Bosilca, Aurélien Bouteiller & Thomas Hérault (joint tutorial at SC'14)
- Jack Dongarra

Elsewhere

- Franck Cappello & Marc Snir, Argonne National Lab.
- Henri Casanova, Univ. Hawai'i
- Saurabh K. Raina, Jaypee IIT, Noida, India