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/

Sophia Tech, November 5, 2013

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)

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

Exascale platforms

- Hierarchical
  - 10^5 or 10^6 nodes
  - Each node equipped with 10^4 or 10^3 cores

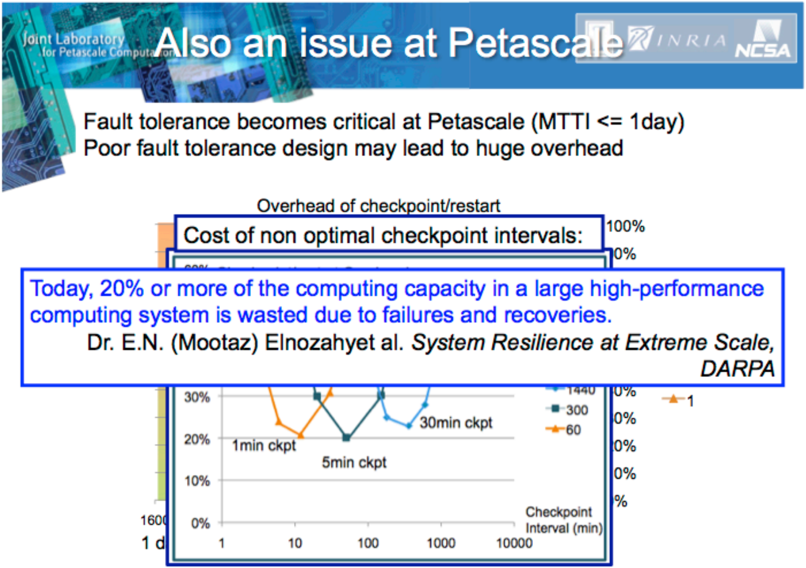
- Failure-prone

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)

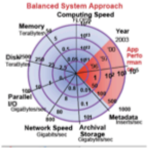
Theorem: $\mu_p = \frac{\mu}{p}$ for arbitrary distributions

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

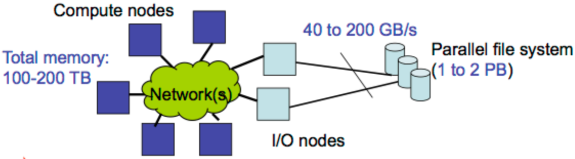


Even for today's platforms (courtesy F. Cappello)

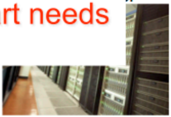
Classic approach for FT: Checkpoint-Restart



Typical "Balanced Architecture" for PetaScale Computers



RoadRunner



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

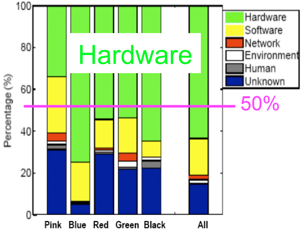
Systems	Perf.	Ckpt time	Source
RoadRunner	1PF	~20 min.	Panasas
LLNL BG/L	500 TF	>20 min.	LLNL
LLNL Zeus	11TF	26 min.	LLNL
YYY BG/P	100 TF	~30 min.	YYY

LLNL BG/L

Error sources (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):



Relative frequency of root cause by system type.

- 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.

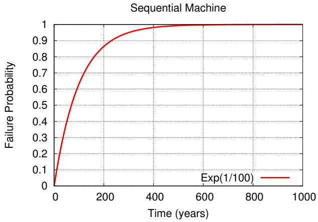
Hardware errors, Disks, processors, memory, network

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

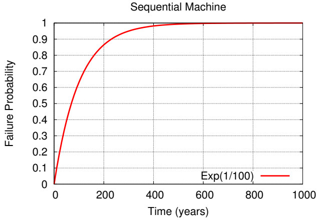
Failure distributions: (1) Exponential



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

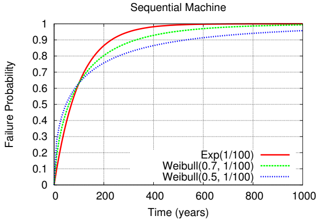
Failure distributions: (1) Exponential



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

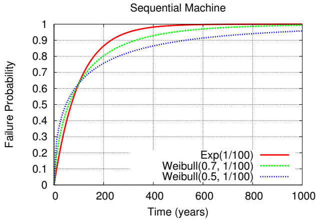
Failure distributions: (2) Weibull



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

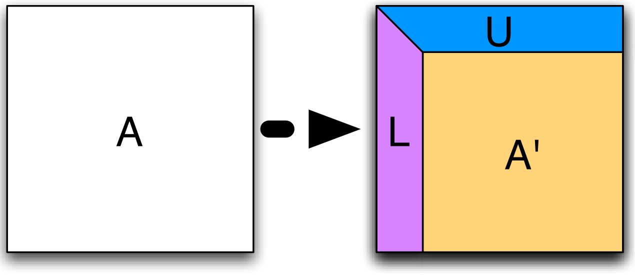
Values from the literature

- 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

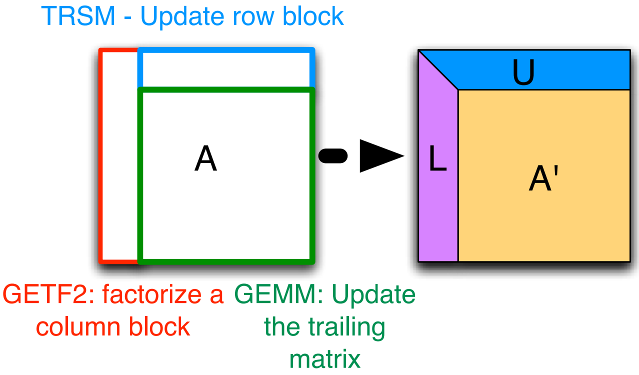
Outline

Tiled LU factorization



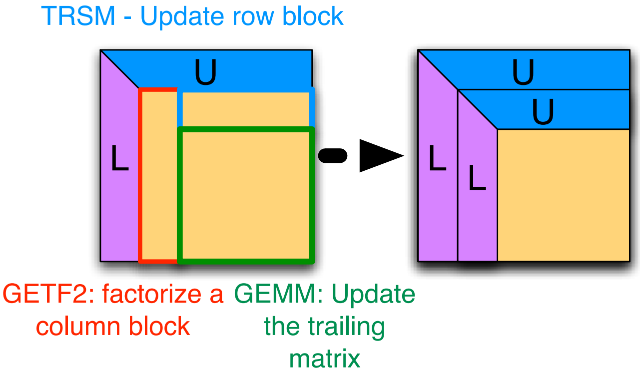
- Solve $A \cdot x = b$ (hard)
- Transform A into a LU factorization
- Solve $L \cdot y = B \cdot b$, then $U \cdot x = y$

Tiled LU factorization



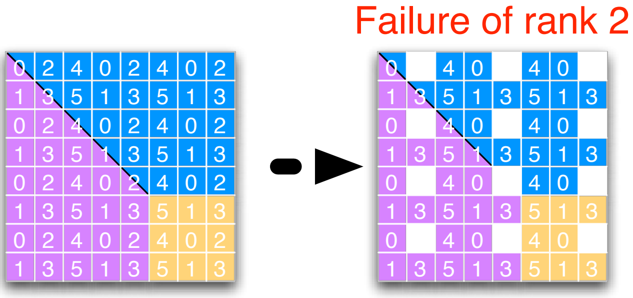
- Solve $A \cdot x = b$ (hard)
- Transform A into a LU factorization
- Solve $L \cdot y = B \cdot b$, then $U \cdot x = y$

Tiled LU factorization



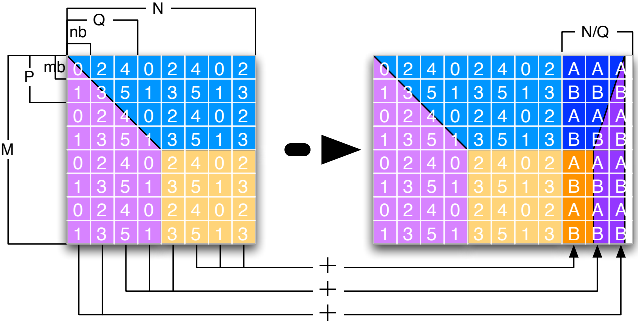
- Solve $A \cdot x = b$ (hard)
- Transform A into a LU factorization
- Solve $L \cdot y = B \cdot b$, then $U \cdot x = y$

Tiled LU factorization



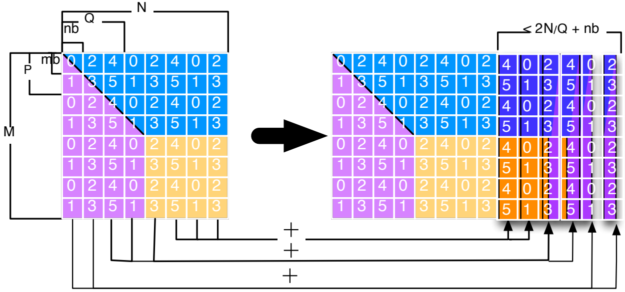
- 2D Block Cyclic Distribution (here 2×3)
- A single failure $\Rightarrow$ many data lost

Algorithm Based Fault Tolerant LU decomposition



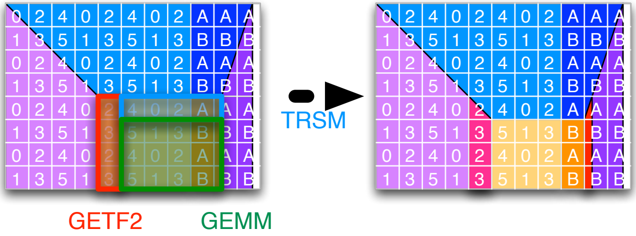
- Checksum: invertible operation on row/column data
  - Checksum replication avoided by dedicating additional computing resources to checksum storage

Algorithm Based Fault Tolerant LU decomposition



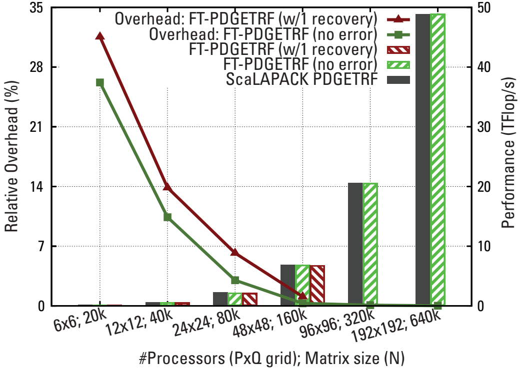
- Checksum: invertible operation on row/column data
  - Checksum blocks are doubled, to allow recovery when data and checksum are lost together (no extra resource needed)

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



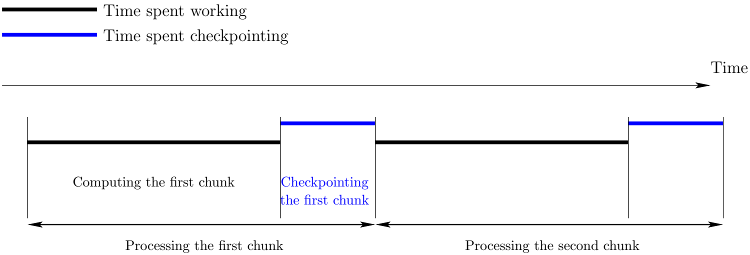
MPI-Next ULFM Performance

- Open MPI with ULFM; Kraken supercomputer;

Outline

Outline

Checkpointing cost



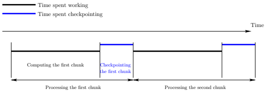
Blocking model: while a checkpoint is taken, no computation can be performed

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 $\Leftrightarrow$ 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

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

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

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

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

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

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

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

T_{lost}?

Waste due to failures

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

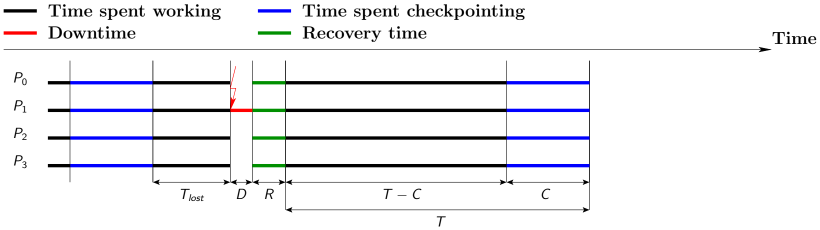
$$\text{TIME}_{\text{final}} = \text{TIME}_{\text{FF}} + N_{faults} \times T_{\text{lost}}$$

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

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

T_{lost}?

Computing T_{lost}



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

⇒ Instants when periods begin and failures strike are independent

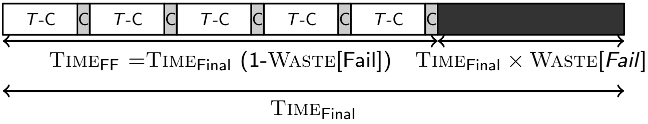
⇒ Valid for all distribution laws, regardless of their particular shape

Waste due to failures

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

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

Total waste



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

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

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

Waste minimization

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

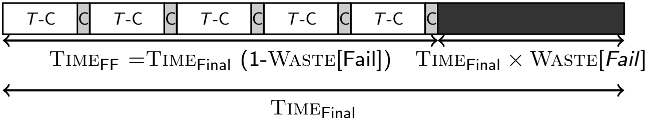
$$\text{WASTE} = \frac{u}{T} + v + wT$$

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

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

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

Comparison with Young/Daly



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

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

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

How accurate?

- 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 ☺

Outline

Expected execution time for a single chunk

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

Recursive Approach

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

Expected execution time for a single chunk

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

Recursive Approach

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

Expected execution time for a single chunk

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

Recursive Approach

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

Expected execution time for a single chunk

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

Recursive Approach

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

Expected execution time for a single chunk

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

Recursive Approach

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

Expected execution time for a single chunk

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

Recursive Approach

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

Expected execution time for a single chunk

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

Recursive Approach

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

Expected execution time for a single chunk

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

Recursive Approach

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

Expected execution time for a single chunk

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

Recursive Approach

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

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

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

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

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

Checkpointing a sequential job

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

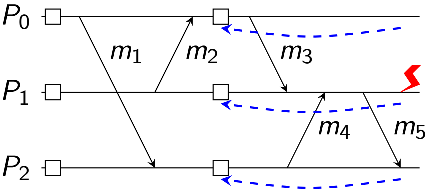
$$\mathbb{E}_{opt}(T(W)) = K^* \left(e^{\lambda R}\left(\frac{1}{\lambda} + D\right)\right)\left(e^{\lambda(\frac{W}{K^*} + C)} - 1\right)$$

- Can also use Daly's second-order approximation

Outline

Background: coordinated checkpointing protocols

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



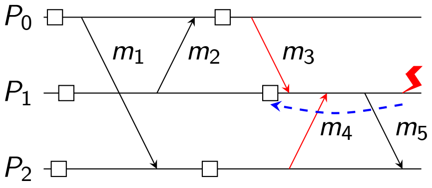
☺ No risk of cascading rollbacks

☺ No need to log messages

☹ All processors need to roll back

Background: message logging protocols

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



☺ No cascading rollbacks

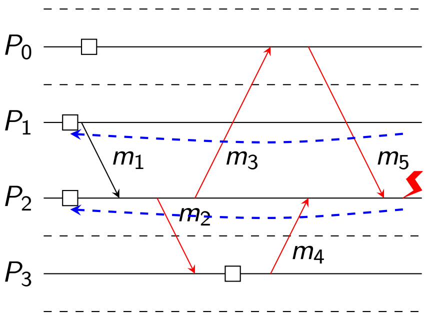
☺ Number of processes to roll back

☹ 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
- 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

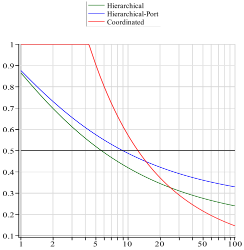
Four platforms: basic characteristics

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

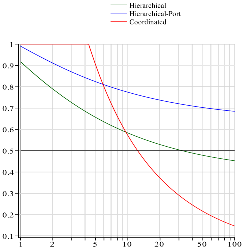
Name	Scenario	G ($C(q)$)	β for 2D-Stencil	β for Matrix-Product
Titan	Coord-IO	1 (2,048s)	/	/
	Hierarch-IO	136 (15s)	0.0001098	0.0004280
	Hierarch-Port	1,246 (1.6s)	0.0002196	0.0008561
K-Computer	Coord-IO	1 (14,688s)	/	/
	Hierarch-IO	296 (50s)	0.0002858	0.001113
	Hierarch-Port	17,626 (0.83s)	0.0005716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

Plotting formulas – Platform: Titan

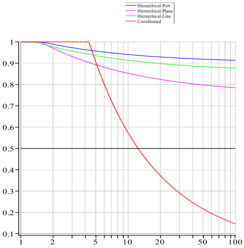
Stencil 2D



Matrix product



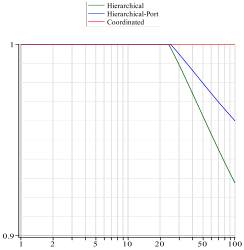
Stencil 3D



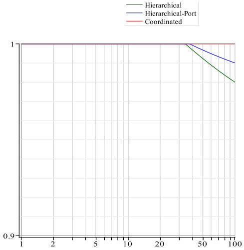
Waste as a function of processor MTBF μ_{ind}

Platform: K-Computer

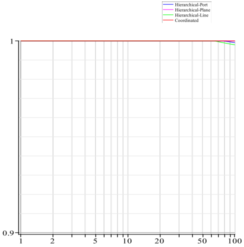
Stencil 2D



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!!!

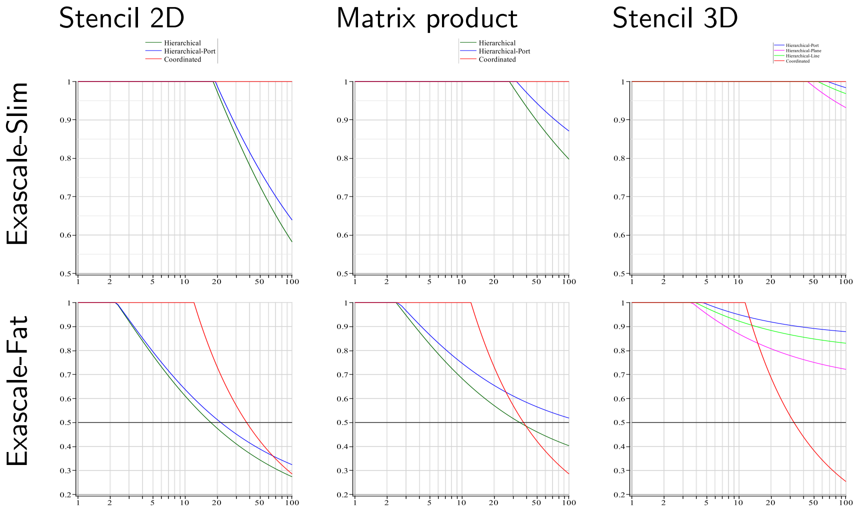
Goodbye Exascale?!

Checkpoint time

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

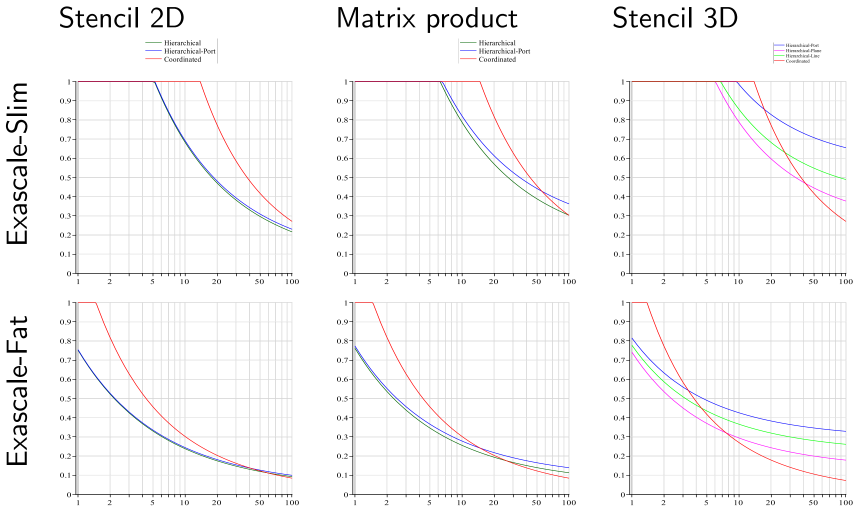
- Large time to dump the memory
- Using $1\%C$, and even $0.1\%C$ for Exascale platforms?

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 $C = 100$

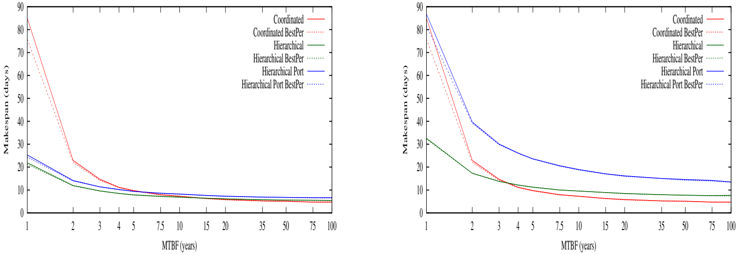


Waste as a function of processor MTBF μ_{ind}, $C = 100$

Simulations – Platform: Titan

Stencil 2D | Matrix product

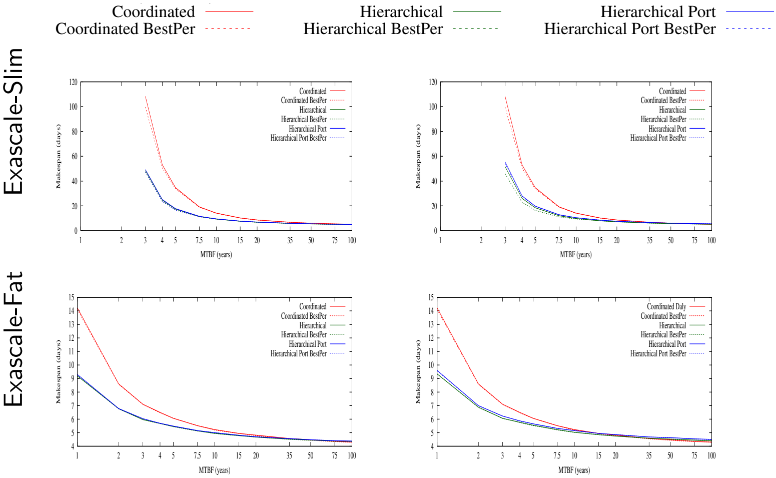
Coordinated —— Hierarchical —— Hierarchical Port ——
Coordinated BestPer - - - Hierarchical BestPer - - - Hierarchical Port BestPer - - -



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

Simulations – Platform: Exascale with $C = 100$

Stencil 2D | Matrix product



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

Outline

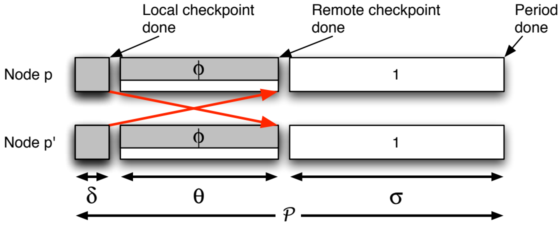
Motivation

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

- Stable storage: high cost

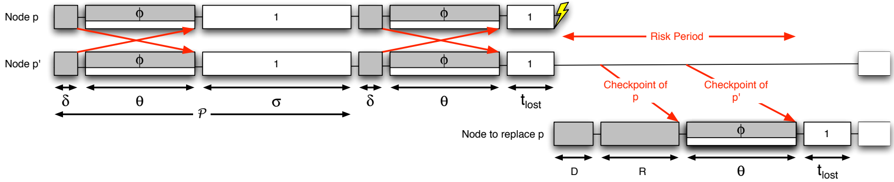
- Distributed in-memory storage:
  - Store checkpoints in local memory $\Rightarrow$ no centralized storage
    ☺ Much better scalability
  - Replicate checkpoints $\Rightarrow$ application survives single failure
    ☹ Still, risk of fatal failure in some (unlikely) scenarios

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

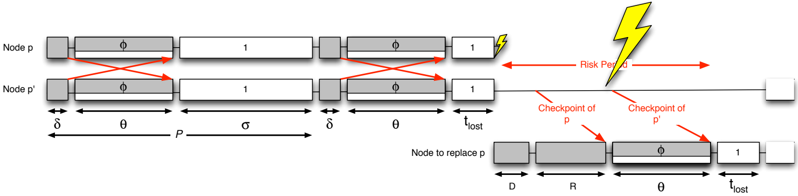
Failures



- 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

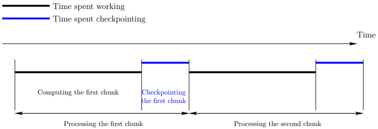
- *true positive*: predicted faults
- *false positive*: fault predictions that did not materialize as actual faults
- *false negative*: unpredicted faults

Algorithm

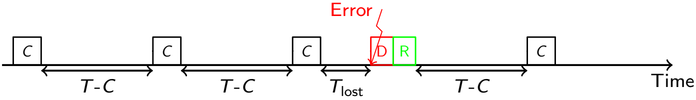
1. While no fault prediction is available:
   - checkpoints taken periodically with period T
2. When a fault is predicted at time t:
   - take a checkpoint ALAP (completion right at time t)
   - after the checkpoint, complete the execution of the period

Computing the waste

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



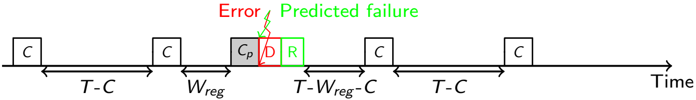
2. **Unpredicted faults:** $\frac{1}{\mu_{NP}}\left[D + R + \frac{T}{2}\right]$



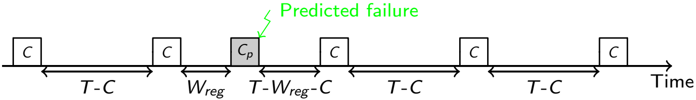
$$\text{WASTE}[fail] = \frac{1}{\mu}\left[(1-r)\frac{T}{2} + D + R + \frac{r}{p}C\right] \Rightarrow T_{opt} \approx \sqrt{\frac{2\mu C}{1-r}}$$

Computing the waste

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



with actual fault (true positive)

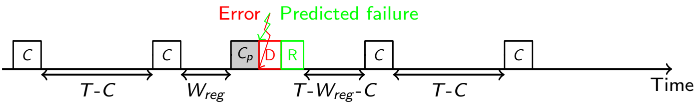


no actual fault (false negative)

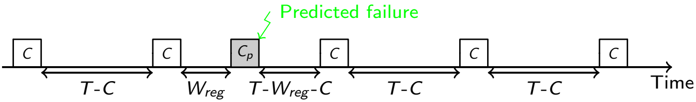
$$\textsc{Waste}[fail] = \frac{1}{\mu}\left[(1-r)\frac{T}{2} + D + R + \frac{r}{p}C\right] \Rightarrow T_{opt} \approx \sqrt{\frac{2\mu C}{1-r}}$$

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)

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

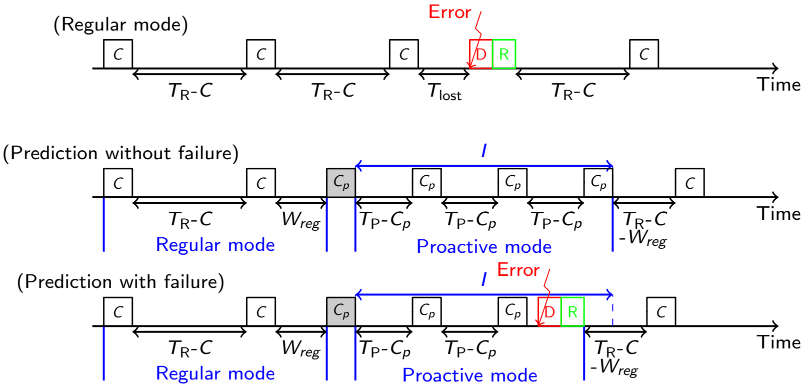
Refinements

- 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

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?

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

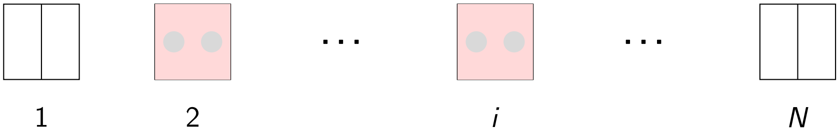
The analogy

Two people with same birthday

$\equiv$

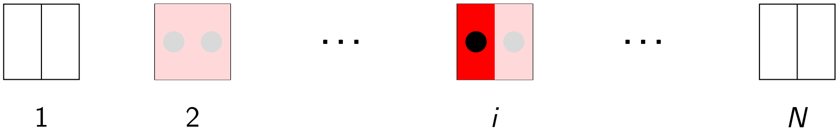
Two failures hitting same replica-group

Differences with birthday problem



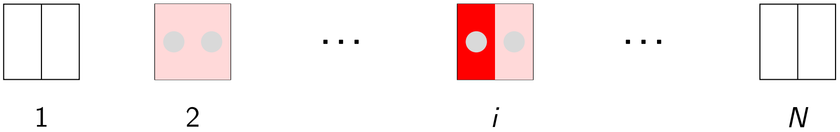
- N processes; each replicated twice
- Uniform distribution of failures
- First failure: each replica-group has probability $1/N$ to be hit
- Second failure

Differences with birthday problem



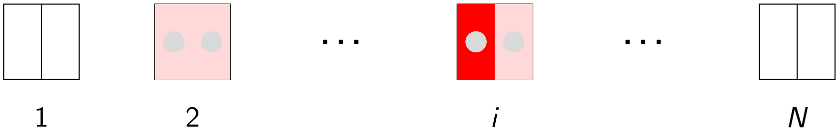
- N processes; each replicated twice
- Uniform distribution of failures
- First failure: each replica-group has probability $1/N$ to be hit
- Second failure

Differences with birthday problem



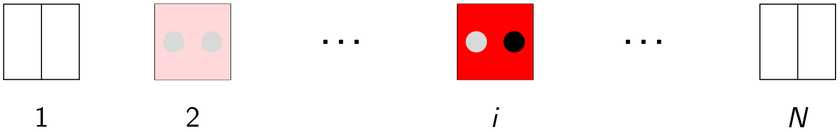
- N processes; each replicated twice
- Uniform distribution of failures
- First failure: each replica-group has probability $1/N$ to be hit
- Second failure: can failed PE be hit?

Differences with birthday problem



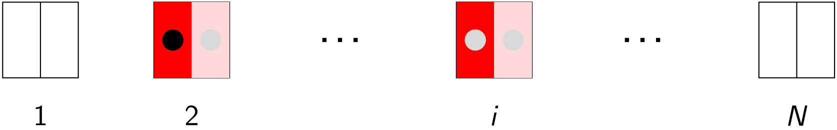
- N processes; each replicated twice
- Uniform distribution of failures
- First failure: each replica-group has probability $1/N$ to be hit
- Second failure cannot hit failed PE
  - Failure uniformly distributed over $2N - 1$ PEs
  - Probability that replica-group i is hit by failure: $1/(2N - 1)$
  - Probability that replica-group $\neq i$ is hit by failure: $2/(2N - 1)$
  - Failure not uniformly distributed over replica-groups: this is not the birthday problem

Differences with birthday problem



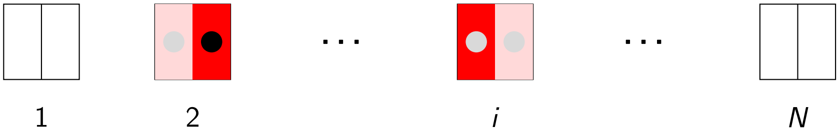
- N processes; each replicated twice
- Uniform distribution of failures
- First failure: each replica-group has probability $1/N$ to be hit
- Second failure cannot hit failed PE
  - Failure uniformly distributed over $2N - 1$ PEs
  - Probability that replica-group i is hit by failure: $1/(2N - 1)$
  - Probability that replica-group $\neq i$ is hit by failure: $2/(2N - 1)$
  - Failure not uniformly distributed over replica-groups: this is not the birthday problem

Differences with birthday problem



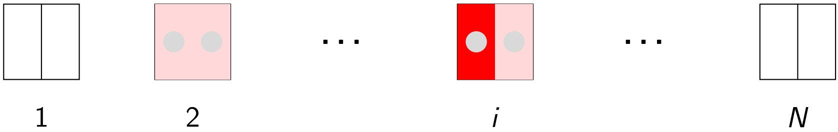
- N processes; each replicated twice
- Uniform distribution of failures
- First failure: each replica-group has probability $1/N$ to be hit
- Second failure cannot hit failed PE
  - Failure uniformly distributed over $2N - 1$ PEs
  - Probability that replica-group i is hit by failure: $1/(2N - 1)$
  - Probability that replica-group $\neq i$ is hit by failure: $2/(2N - 1)$
  - Failure not uniformly distributed over replica-groups: this is not the birthday problem

Differences with birthday problem



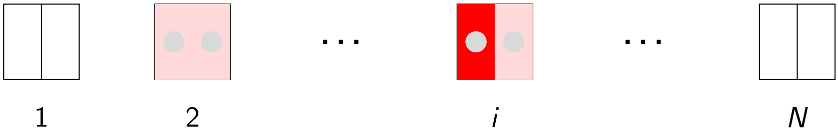
- N processes; each replicated twice
- Uniform distribution of failures
- First failure: each replica-group has probability $1/N$ to be hit
- Second failure cannot hit failed PE
  - Failure uniformly distributed over $2N - 1$ PEs
  - Probability that replica-group i is hit by failure: $1/(2N - 1)$
  - Probability that replica-group $\neq i$ is hit by failure: $2/(2N - 1)$
  - Failure not uniformly distributed over replica-groups: this is not the birthday problem

Differences with birthday problem



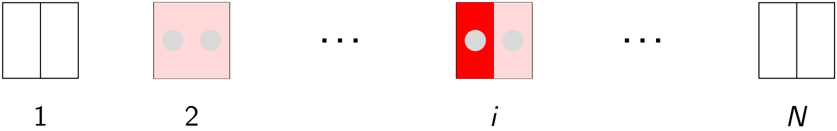
- N processes; each replicated twice
- Uniform distribution of failures
- First failure: each replica-group has probability $1/N$ to be hit
- Second failure cannot hit failed PE
  - Failure uniformly distributed over $2N-1$ PEs
  - Probability that replica-group i is hit by failure: $1/(2N-1)$
  - Probability that replica-group $\neq i$ is hit by failure: $2/(2N-1)$
  - Failure not uniformly distributed over replica-groups: this is not the birthday problem

Differences with birthday problem



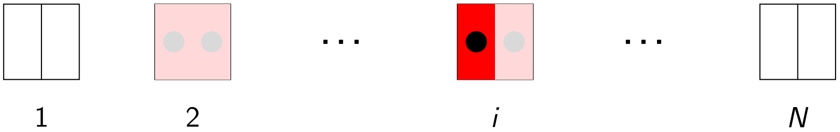
- N processes; each replicated twice
- Uniform distribution of failures
- First failure: each replica-group has probability $1/N$ to be hit
- Second failure can hit failed PE

Differences with birthday problem



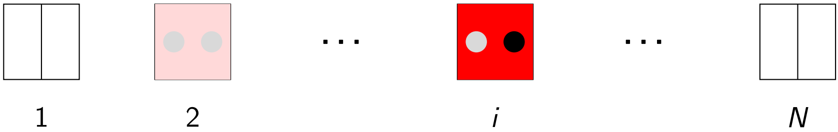
- N processes; each replicated twice
- Uniform distribution of failures
- First failure: each replica-group has probability $1/N$ to be hit
- Second failure can hit failed PE
  - Suppose failure hits replica-group i
  - If failure hits failed PE: application survives
  - If failure hits running PE: application killed
  - Not all failures hitting the same replica-group are equal: this is not the birthday problem

Differences with birthday problem



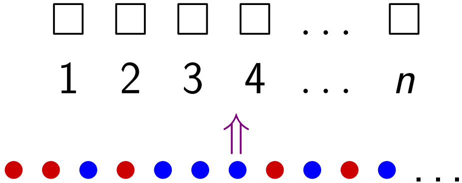
- N processes; each replicated twice
- Uniform distribution of failures
- First failure: each replica-group has probability $1/N$ to be hit
- Second failure can hit failed PE
  - Suppose failure hits replica-group i
  - If failure hits failed PE: application survives
  - If failure hits running PE: application killed
  - Not all failures hitting the same replica-group are equal: this is not the birthday problem

Differences with birthday problem



- N processes; each replicated twice
- Uniform distribution of failures
- First failure: each replica-group has probability $1/N$ to be hit
- Second failure can hit failed PE
  - Suppose failure hits replica-group i
  - If failure hits failed PE: application survives
  - If failure hits running PE: application killed
  - Not all failures hitting the same replica-group are equal: this is not the birthday problem

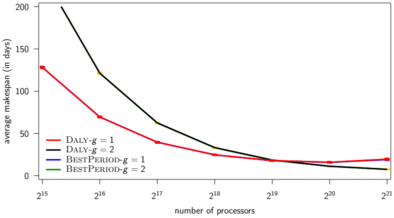
Correct analogy



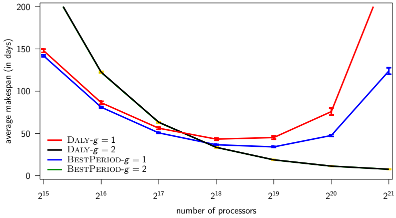
$N = n_{rg}$ bins, red and blue balls

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

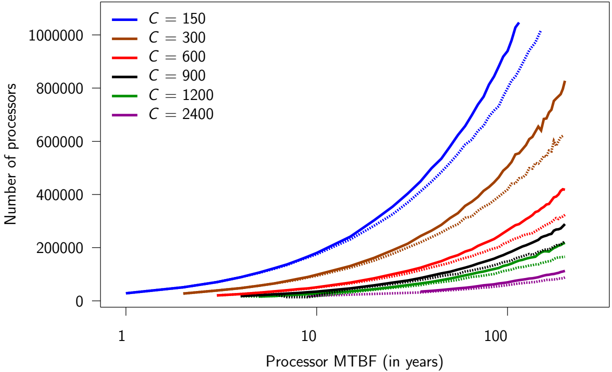


(b) Weibull, $k = 0.7$

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

Weibull distribution with $k = 0.7$

Dashed line: Ferreira et al. Solid line: Correct analogy



- Study by Ferrreira et al. favors replication
- Replication beneficial if small μ + large C + big p_{total}

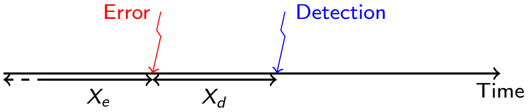
Outline

Silent errors

- Many types of faults: software error, hardware malfunction, memory corruption
- Many possible behaviors: silent, transient, unrecoverable
- Consider silent errors here
- This includes some software faults, some hardware errors (soft errors in L1 cache), bit flips (cosmic radiations)
- Silent error detected when corrupt data is activated

Detection latency

- Instantaneous error detection ⇒ fail-stop failures
- Silent errors (data corruption) ⇒ detection latency

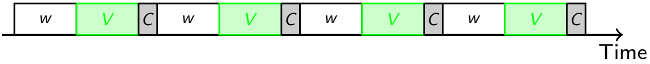


Error and detection latency

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

Coupling checkpointing and verification

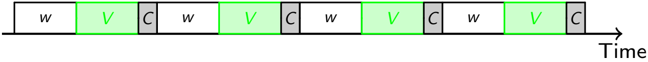
- Verification mechanism of cost V
- Simplest idea: verify work before each checkpoint



V large compared to $w \Rightarrow$ large $\text{W}\small{\text{ASTE}}_{\text{ff}}$, can we improve that?

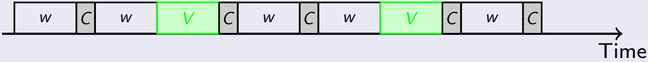
Coupling checkpointing and verification

- Verification mechanism of cost V
- Simplest idea: verify work before each checkpoint



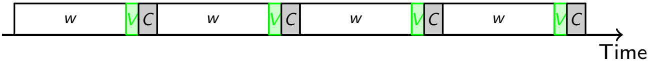
V large compared to $w \Rightarrow$ large WASTE_{ff}, can we improve that?

Is this better?



Coupling checkpointing and verification

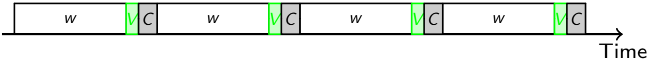
- Verification mechanism of cost V
- Simplest idea: verify work before each checkpoint



V small in front of w $\Rightarrow$ large $\text{WASTE}_{\text{fail}}$, can we improve that?

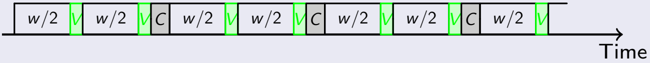
Coupling checkpointing and verification

- Verification mechanism of cost V
- Simplest idea: verify work before each checkpoint

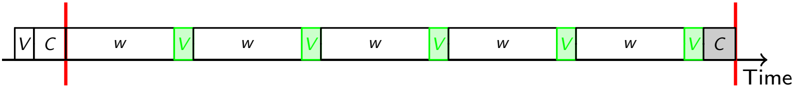


V small in front of $w \Rightarrow$ large $\text{WASTE}_{\text{fail}}$, can we improve that?

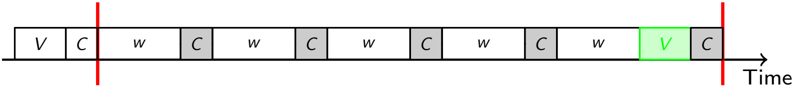
Is this better?



Coupling checkpointing and verification



Small cost V: 5 verifications for 1 checkpoint

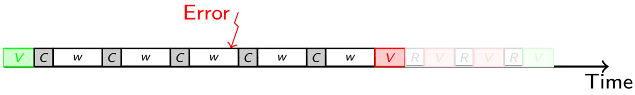


Large cost V: 5 checkpoints for 1 verification

More complicated periodic patterns? Different-size chunks?

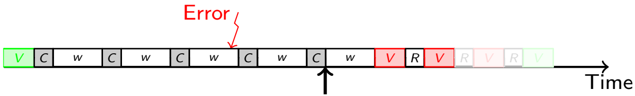
k checkpoints for 1 verification

Where did the error strike?



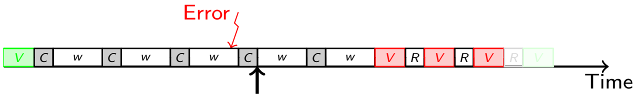
k checkpoints for 1 verification

Where did the error strike?



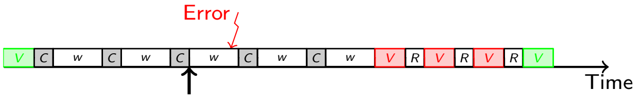
k checkpoints for 1 verification

Where did the error strike?



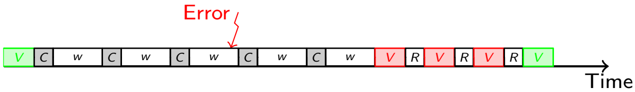
k checkpoints for 1 verification

Where did the error strike?



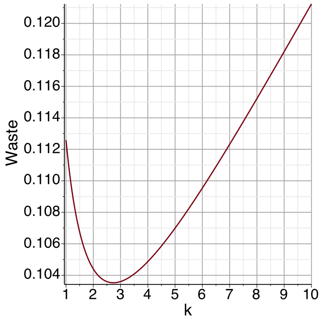
k checkpoints for 1 verification

Where did the error strike?



$$\text{Re-Exec} = 2(w + C) + (w + V)$$

k checkpoints for 1 verification



Waste as function of k, using optimal period
($V = 100s$, $C = R = 6s$ and $\mu = \frac{10 years}{10^5}$)

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)

UT Knoxville

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

Others

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