Two-level checkpointing and partial verifications for linear task graphs

Anne Benoit, Aurélien Cavelan, Yves Robert and Hongyang Sun

ENS Lyon, France

Anne.Benoit@ens-lyon.fr
http://graal.ens-lyon.fr/~abenoit

6th Int. Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS15) @ SC'15

November 15, 2015, Austin, TX

Computing at Exascale

Exascale platform:

- 10^5 or 10^6 nodes, each equipped with 10^2 or 10^3 cores
- Shorter Mean Time Between Failures (MTBF) μ

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

MTBF (individual node)	1 year	10 years	120 years
MTBF (platform of 10^6 nodes)	30 sec	5 mn	1 h

Need more reliable components!!

Need more resilient techniques!!!

Computing at Exascale

Exascale platform:

- 10^5 or 10^6 nodes, each equipped with 10^2 or 10^3 cores
- Shorter Mean Time Between Failures (MTBF) μ

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

MTBF (individual node)	1 year	10 years	120 years
MTBF (platform of 10^6 nodes)	30 sec	5 mn	1 h

Need more reliable components!!

Need more resilient techniques!!!

Two main sources of errors

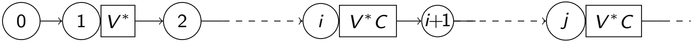
- Fail-stop errors: instantaneous error detection, e.g., resource crash
- Silent errors (aka silent data corruptions), e.g., soft faults in L1 cache, ALU, double bit flip
  - Silent error is detected only when corrupted data is activated, which could happen long after its occurrence ☹
  - Detection latency is problematic

  - Before each checkpoint, run some verification mechanism (checksum, ECC, coherence tests, TMR, etc)
  - Silent error is detected by verification
    $\Rightarrow$ checkpoint always valid ☺

Verified checkpoints, rollback and recovery

One step further and partial verifications

- Perform several verifications before each checkpoint:
  - Pro: silent error is detected earlier in the pattern ☺
  - Con: additional overhead in error-free executions ☹

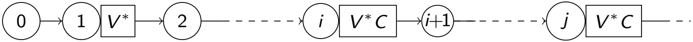


- Guaranteed/perfect verifications (V^*) can be very expensive! Partial verifications (V) are available for many HPC applications!
  - Lower accuracy: recall $r = \frac{\#\text{detected errors}}{\#\text{total errors}} < 1$ ☹
  - Much lower cost, i.e., $V < V^*$ ☺

How many intermediate verifications to use and the positions?

One step further and partial verifications

- Perform several verifications before each checkpoint:
    - Pro: silent error is detected earlier in the pattern ☺
    - Con: additional overhead in error-free executions ☹



- Guaranteed/perfect verifications (V^*) can be very expensive! Partial verifications (V) are available for many HPC applications!
    - Lower accuracy: recall $r = \frac{\#\text{detected errors}}{\#\text{total errors}} < 1$ ☹
    - Much lower cost, i.e., $V < V^*$ ☺

How many intermediate verifications to use and the positions?

Two-level checkpointing

- Silent errors: use of a lightweight mechanism of in-memory checkpoints C_M
- Local copies lost in case of fail-stop errors: use (less frequent) copies on stable storage (classical disk checkpoints) C_D
- Always C_M before C_D: little overhead, enforced in practice
- Always V^* before C_M: all checkpoints are valid
- Verifications, memory copies and I/O transfers protected from errors

Outline

1. Problem statement
2. Theoretical analysis
3. Performance evaluation
4. Conclusion

Application and errors

- Linear chain of tasks $T_1, T_2, \ldots, T_n$
- Each task T_i has a weight w_i (computational load)
- $W_{i,j} = \sum_{k=i+1}^{j} w_k$: time to execute tasks T_{i+1} to T_j

- Subject to fail-stop and silent errors, independent and following a *Poisson process* with arrival rates λ_f and λ_s
- $p^f_{i,j} = 1 - e^{-\lambda_f W_{i,j}}$: probability of having at least a fail-stop error while executing T_{i+1} to T_j
- $p^s_{i,j} = 1 - e^{-\lambda_s W_{i,j}}$: idem for silent errors

Resilience parameters and objective

- Cost of disk checkpointing C_D, cost of disk recovery R_D
- Cost of memory checkpointing C_M, cost of memory recovery R_M
- For simplicity, R_M included in R_D

- Cost V^* for guaranteed verification
- V for partial verification, with recall r, and $g = 1 - r$ is the proportion of undetected errors

⇒ Decide where to place disk checkpoints, memory checkpoints, guaranteed verifications and partial verifications, in order to minimize the **expected execution time** (or makespan) of the application

Outline

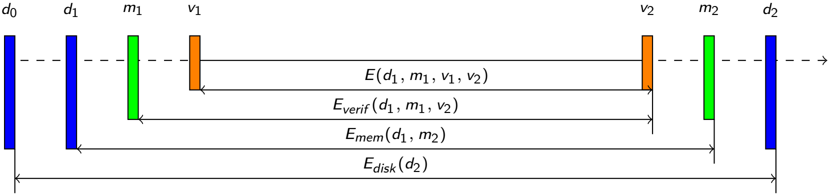
1. Problem statement
2. Theoretical analysis
3. Performance evaluation
4. Conclusion

Dynamic programming algorithm

Several dynamic programming levels:

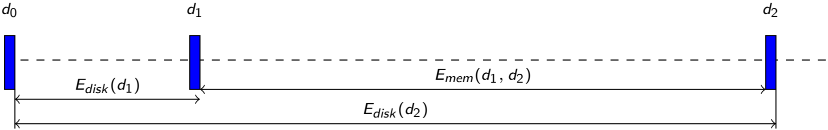
- First decide where to place disk checkpoints
- Then memory checkpoints between any two disk checkpoints
- And finally, guaranteed or partial verifications between any two memory checkpoints

- Compute the expected execution time between any two verifications



Without partial verifications

Placing disk checkpoints:



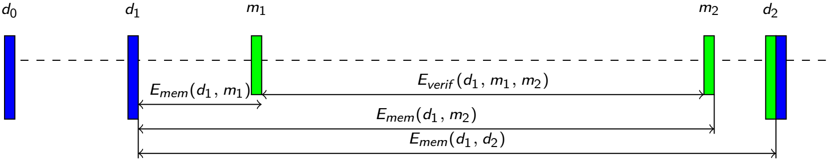
- $E_{disk}(d_2)$: expected time needed to successfully execute tasks T_1 to T_{d_2}, where T_{d_2} is followed by $V^* C_M C_D$:

$$E_{disk}(d_2) = \min_{0 \le d_1 < d_2} \{E_{disk}(d_1) + E_{mem}(d_1, d_2) + C_D\}$$

- Objective: $E_{disk}(n)$
- Initialization: $E_{disk}(0) = 0$

Without partial verifications

Placing memory checkpoints:



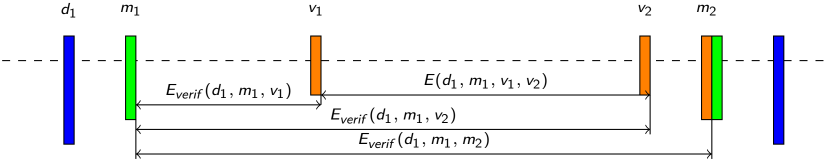
- $E_{mem}(d_1, m_2)$: expected time needed to successfully execute tasks T_{d_1+1} to T_{m_2}, where T_{d_1} is followed by $V^* C_M C_D$ and T_{m_2} is followed by $V^* C_M$:

$$E_{mem}(d_1, m_2) = \min_{d_1 \le m_1 < m_2} \{E_{mem}(d_1, m_1) + E_{verif}(d_1, m_1, m_2) + C_M\}$$

- Initialization: $E_{mem}(d_1, d_1) = 0$

Without partial verifications

Placing additional guaranteed verifications:



- $E_{verif}(d_1, m_1, v_2)$: expected time needed to successfully execute tasks T_{m_1+1} to T_{v_2}, where T_{d_1} is followed by $V^* C_M C_D$, T_{m_1} is followed by $V^* C_M$, T_{v_2} is followed by V^*:

$$E_{verif}(d_1, m_1, v_2) = \min_{m_1 \le v_1 < v_2} \{E_{verif}(d_1, m_1, v_1) + E(d_1, m_1, v_1, v_2)\}$$

- Initialization: $E_{verif}(d_1, m_1, m_1) = 0$

Without partial verifications

Expected execution time between two verifications $E(d_1, m_1, v_1, v_2)$, knowing positions of last C_D and last C_M:

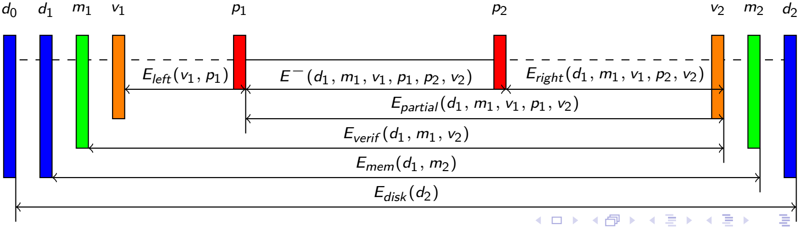
- If $p^f_{v_1,v_2}$, recover from C_D
- Otherwise, if $p^s_{v_1,v_2}$, detect error at v_2 and recover from C_M

$$\begin{aligned} & E(d_1, m_1, v_1, v_2) = \\ & p^f_{v_1,v_2} \left(T^{\text{lost}}_{v_1,v_2} + R_D + E_{mem}(d_1, m_1) + E_{verif}(d_1, m_1, v_1) + E(d_1, m_1, v_1, v_2) \right) \\ & + \left(1 - p^f_{v_1,v_2}\right) \Big(W_{v_1,v_2} + V^* \\ & \qquad + p^s_{v_1,v_2} \left(R_M + E_{verif}(d_1, m_1, v_1) + E(d_1, m_1, v_1, v_2) \right) \Big) \end{aligned}$$

- Compute $T^{\text{lost}}_{v_1,v_2} = \frac{1}{\lambda_f} - \frac{W_{v_1,v_2}}{e^{\lambda_f W_{v_1,v_2}} - 1}$ and simplify

And with partial verifications?

- Probability g that error remains undetected after partial verification
- Need to account fo time lost executing following tasks until error is detected: compute first values at the right of the current interval
- $E_{partial}(d_1, m_1, v_1, p_1, v_2)$: expected time needed to execute all tasks T_{p_1+1} to T_{v_2}, tries all positions p_2 for next partial verification
- $E_{partial}(d_1, m_1, v_1, p_1, v_2)$ calls recursively $E_{partial}(d_1, m_1, v_1, p_2, v_2)$
- To compute $E^-(d_1, m1, v_1, p_1, p_2, v_2)$, need to know $E_{left}(v_1, p_1)$ and $E_{right}(d_1, m_1, v_1, p_2, v_2)$; E_{right} can be computed, and E_{left} accounted for separately (independent on nb of partial verifs)



Outline

1. Problem statement
2. Theoretical analysis
3. Performance evaluation
4. Conclusion

Simulation settings

- Identical recovery and checkpoint costs: $R_D = C_D$ and $R_M = C_M$
- $V^* = C_M$ (check all data in memory), $V = \frac{V^*}{100}$ and $r = 0.8$
- Work $W = 25000$ seconds, distributed between up to $n = 50$ tasks:
  - *Uniform*: all tasks share the same cost $\frac{W}{n}$ (matrix multiplication, iterative stencil kernels)
  - *Decrease*: task T_i has cost $\alpha(n+1-i)^2$, where $\alpha \approx \frac{3W}{n^3}$ (dense matrix solvers)
  - *HighLow*: set of identical tasks with large costs followed by tasks with small costs
- Platforms used to evaluate Scalable Checkpoint/Restart (SCR) library (Moody et al.):

platform	#nodes	λ_f	λ_s	C_D	C_M
Hera	256	9.46e-7	3.38e-6	$300s$	$15.4s$
Atlas	512	5.19e-7	7.78e-6	$439s$	$9.1s$
Coastal	1024	4.02e-7	2.01e-6	$1051s$	$4.5s$
Coastal SSD	1024	4.02e-7	2.01e-6	$2500s$	$180.0s$

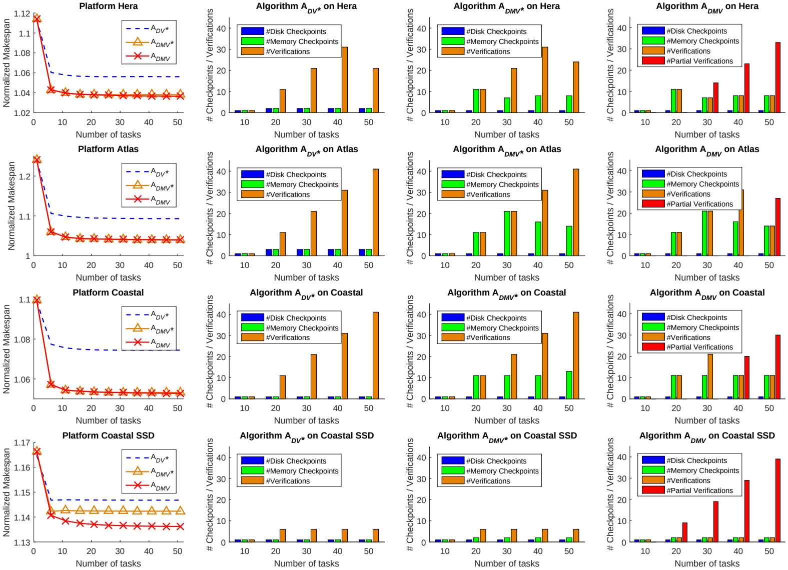


Figure: Performance of the three algorithms with *uniform* distribution

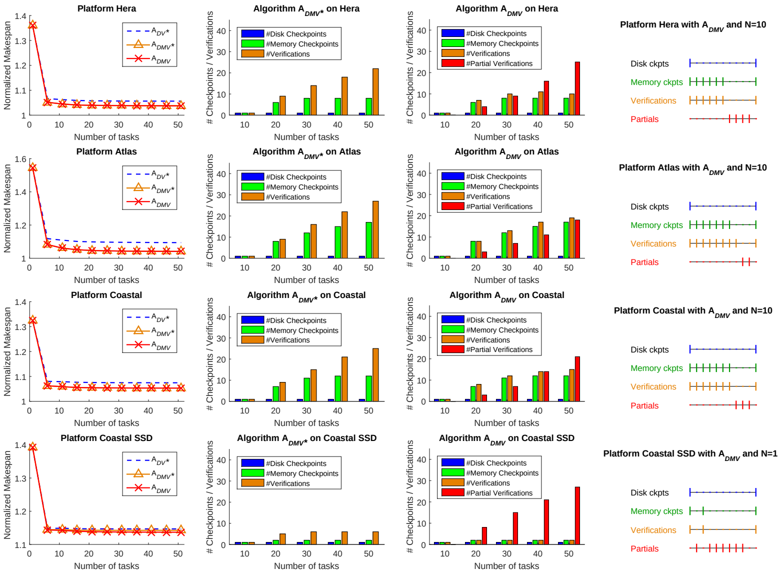


Figure: Performance of the three algorithms with *decrease* distribution

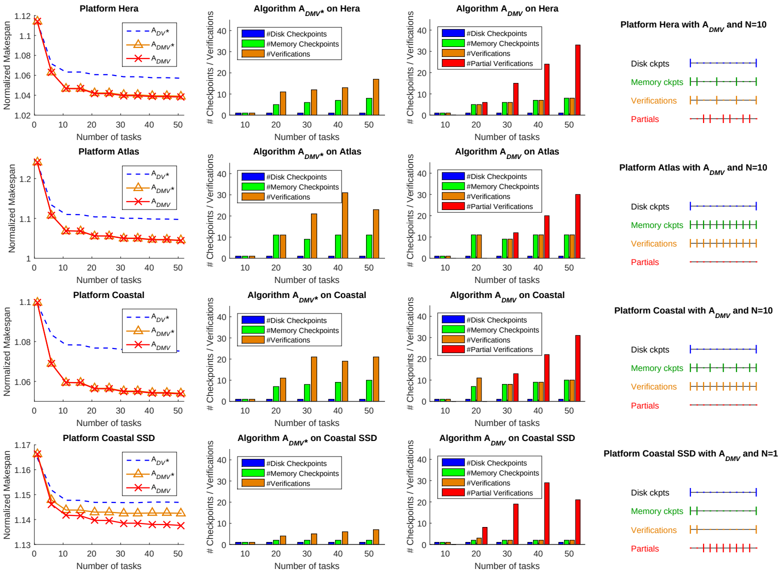


Figure: Performance of the three algorithms with *highlow* distribution

Summary of simulations

- More tasks → better performance
- Single-level algorithm: Guaranteed verifications everywhere, except with too many tasks ($n = 50$ on Hera) or cost of verification too high (Coastal SSD)
- Two-level algorithms: Use of memory checkpoints drastically reduces makespan
- With partial verifications: Need to use a lot of them (smaller recall): useful only when enough tasks; limited impact, except for Coastal SSD with higher checkpointing and verification costs

Outline

1. Problem statement
2. Theoretical analysis
3. Performance evaluation
4. Conclusion

Conclusion

- Two-level checkpointing scheme to cope with fail-stop and silent errors
- Combines disk/memory checkpoints with guaranteed/partial verifications
- Theoretically: multi-level polynomial-time dynamic programming algorithm for linear chains ($O(n^6)$)
- Practically: benefit of combined approach with realistic parameters, fast in practice

Future directions

- Usefulness of the approach on general application workflows
- Need of efficient polynomial-time heuristics

Research report RR-8794 available at `graal.ens-lyon.fr/~abenoit`