Checkpointing workflows for fail-stop errors

Li Han[1,2], Louis-Claude Canon[1,3], Henri Casanova[4],
Yves Robert[1,5] and Frédéric Vivien[1]

1. Laboratoire LIP, ENS Lyon & Inria, France
2. East China Normal University, China
3. FEMTO-ST, Université de Bourgogne Franche-Comté, France
4. University of Hawai'i at Manoa, USA
5. University of Tennessee Knoxville, USA

ICL Friday Lunch – February 2, 2018

IEEE TRANSACTIONS ON

COMPUTERS

A publication of the IEEE Computer Society

[illegible]

Introducing the New Editor-in-Chief of the IEEE Transactions on Computers [illegible]

PAPERS

Compiler Techniques

[illegible]

Digital Devices, Computer Components, and Interconnection Networks

[illegible]

Operating Systems, Software Systems, and Communication Protocols

[illegible]

Real-Time Systems and Embedded Systems

[illegible]

Security

[illegible]

COMPUTER SOCIETY

IEEE

Revised version just made it!

Framework

- Scheduling workflows for fail-stop errors
- Decide
  (i) allocation of tasks to processors
  (ii) which task to checkpoint
- Go beyond linear chains (ok, with parallel tasks)
  ⇒ analysis with unique (powerful, error-prone) super-processor
- Go beyond linear algebra kernels
  ⇒ extensive re-use of input/output files

Outline

1. CkptNone
2. CkptAll
3. CkptSome
4. Experiments
5. Conclusion

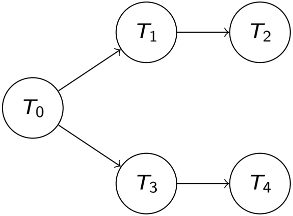
Outline

1. CkptNone
2. CkptAll
3. CkptSome
4. Experiments
5. Conclusion

CkptNone

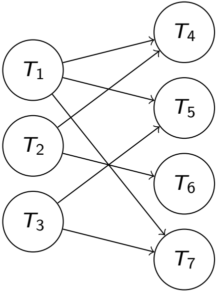
- No task is checkpointed

CkptNone



With one processor

CkptNone



With one processor

CkptNone

?

With several processors

CkptNone

ALLOCATION & ORDERING GIVEN

$\Longrightarrow$ compute makespan

?

With several processors

CkptNone

ONE PROCESSOR PER TASK

$\Longrightarrow$ compute makespan

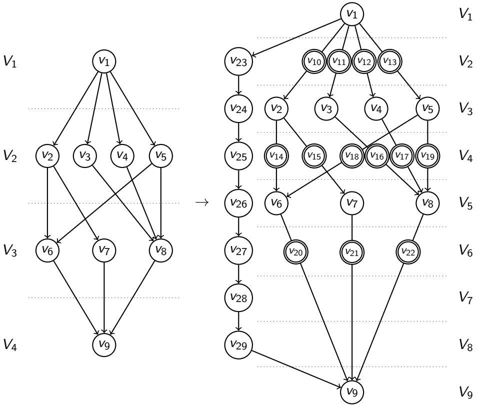
?

CkptNone

ONE PROCESSOR PER TASK,
UNIT-SIZE TASKS, NO COMMUNICATION COST
$\Longrightarrow$ compute makespan

?

CkptNone



#P-COMPLETE

ONE PROCESSOR PER TASK,
UNIT-SIZE TASKS, NO COMMUNICATION COST

$\Longrightarrow$ compute makespan

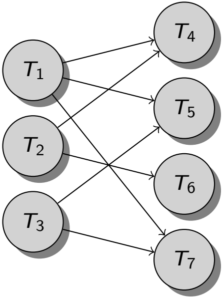
Outline

1. CkptNone
2. CkptAll
3. CkptSome
4. Experiments
5. Conclusion

CkptAll

- Each task is checkpointed
- De-facto standard for Workflow Management Systems
- Problem 1: find an allocation and ordering (solution)
- Problem 2: given a solution, compute the makespan?

Problem 2: Makespan of a solution



- Each task is checkpointed:
  expected makespan ⇒ longest path of probabilistic DAGs
- Equivalent to having one processor per task

Longest path of probabilistic DAGs

- Also known as PERT problems
- Task weights = random variables
- Unlimited resources (one processor per task)
- Expected length of longest path?
  #P-complete problem (reduction from reliability)
  #P-complete if each task has two possible values

Longest path of probabilistic DAGs

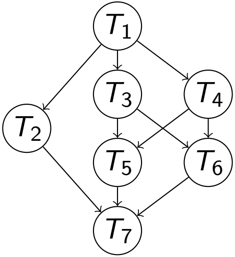
Evaluation methods

- MonteCarlo
- Dodin
- Normal

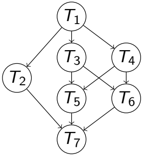
Outline

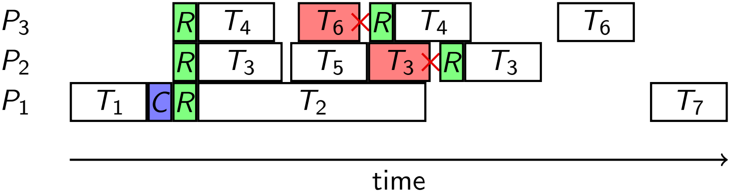
1. CkptNone
2. CkptAll
3. CkptSome
4. Experiments
5. Conclusion

CkptSome

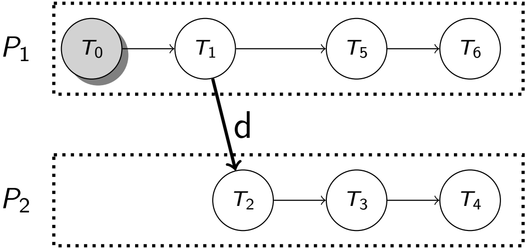


CkptSome

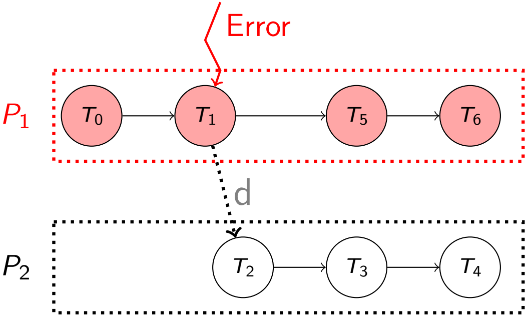

T_1
T_3
T_4
T_2
T_5
T_6
T_7


P_3
R
T_4
T_6
R
T_4
T_6
P_2
R
T_3
T_5
T_3
R
T_3
P_1
T_1
C
R
T_2
T_7
time

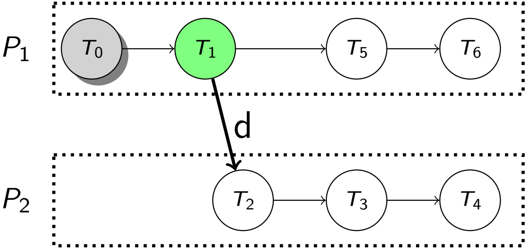
Crossover dependency



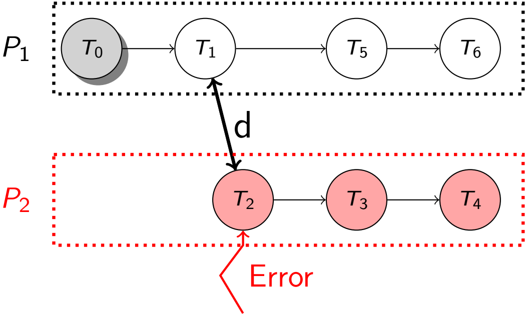
Crossover dependency



Crossover dependency



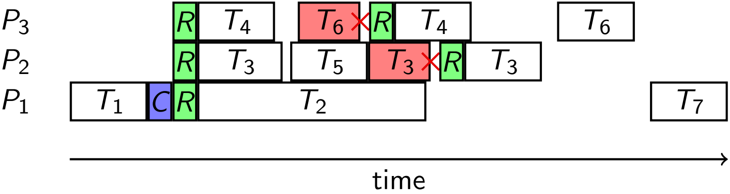
Crossover dependency



Lesson learnt

- Each processor is scheduled to execute many tasks
- Due to crossover dependencies, a few crashes can lead to many task re-executions and data re-transfers, during which other crashes can occur
- Avoid crossover dependencies!

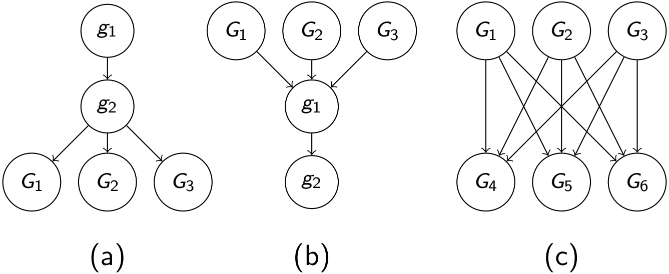
Back to the example



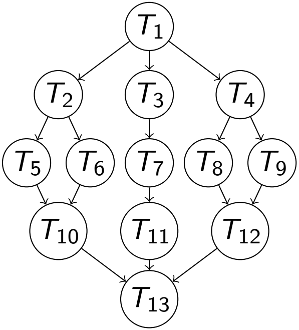
Checkpoint every task except T_2 and T_7 to avoid cross-dependencies

M-SPGs

- Series-parallel graphs without merging sources/sinks
- Examples:

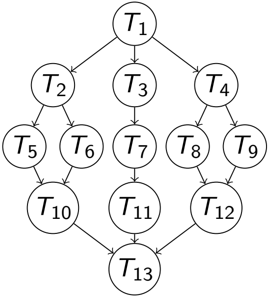


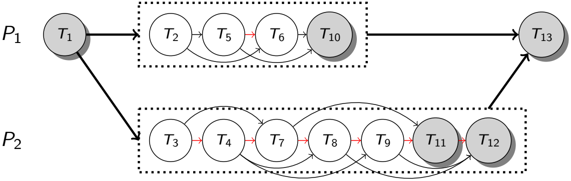
Scheduling M-SPGs



- Proportional mapping applied recursively
- Sets of tasks assigned to a single processor are linearized into superchains
- Checkpoint all exit tasks in each superchain

Scheduling M-SPGs

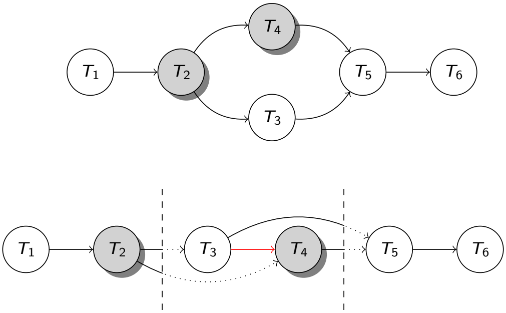




First solution

- Coalesce all tasks of a superchain into a single big task
- Checkpoint that big task
- Then use Monte-Carlo to evaluate makespan
- Works but may not use enough checkpoints

Checkpointing superchains



Checkpoint = saving to stable storage all output data of previously executed but un-checkpointed tasks

Optimal dynamic programming algorithm

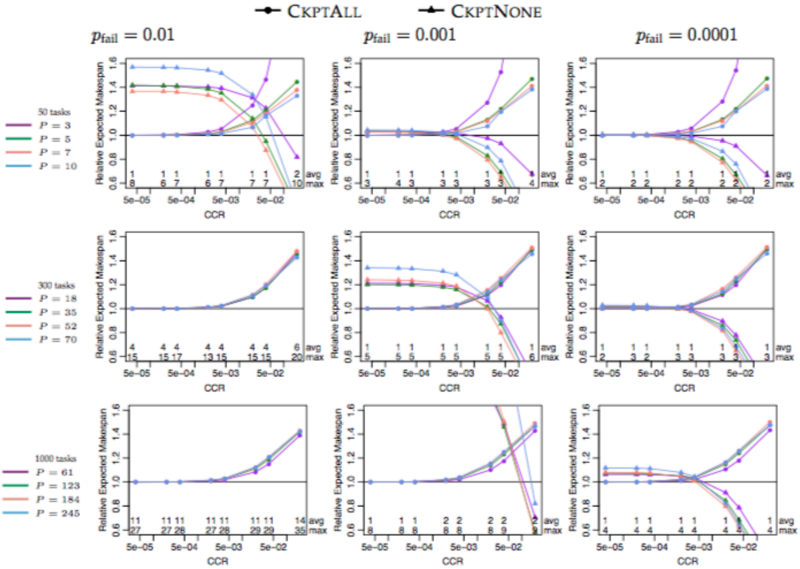
Outline

1. CkptNone
2. CkptAll
3. CkptSome
4. Experiments
5. Conclusion

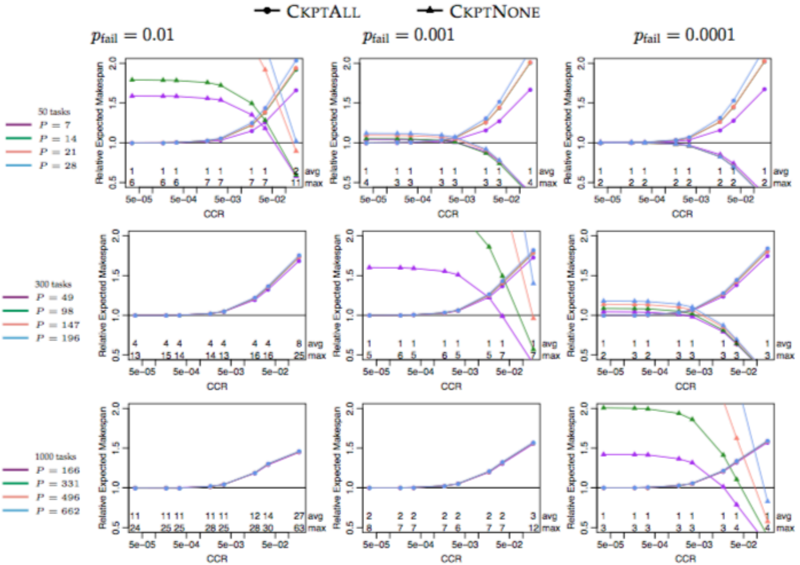
Framework

- Pegasus workflows: Montage, Ligo, Genome
- Task weights in seconds, file sizes in bytes
  ⇒ vary Communication-to-Computation Ratio CCR
- Each task fails with probability $p_{\text{fail}} \in \{1\%, 0.1\%, 0.01\%\}$

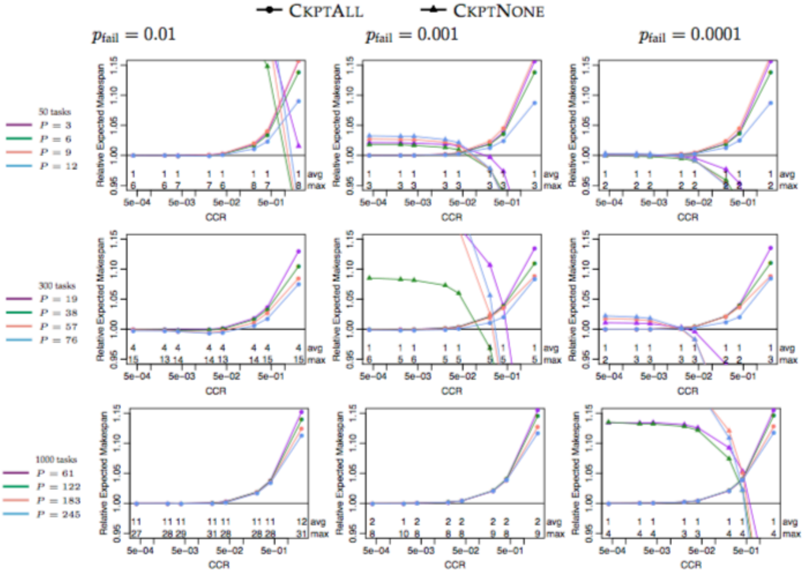
Results for GENOME



Results for Montage



Results for LIGO



Outline

1. CkptNone
2. CkptAll
3. CkptSome
4. Experiments
5. Conclusion

Lame Conclusion

Summary

- M-SPGs broadly relevant to production workflows
- M-SPGs structure key to design CkptSome
- Attractive compromise between CkptAll and CkptNone:
  – CkptSome always outperforms CkptAll
  – CkptSome outperformed by CkptNone only when checkpoints are expensive and/or failures are rare

Future work

- Extension to parallel (moldable) tasks
- Extension to General Series Parallel Graphs
  (transitive reduction is an M-SPG)
- Refine linearization algorithm
  (related to sum-cut problem)

Lame Conclusion

Summary

- M-SPGs broadly relevant to production workflows
- M-SPGs structure key to design CkptSome
- Attractive compromise between CkptAll and CkptNone:
  – CkptSome always outperforms CkptAll
  – CkptSome outperformed by CkptNone only when checkpoints are expensive and/or failures are rare

Future work

Candid conclusion

Disappointing to be stuck with specific graphs

Problem still open ☹

(related to sum-cut problem)

This is my 10th trip to Hawai‘i

MAHALO!

This is my 100th trip to Knoxville

THANK YE ALL!