Scheduling computational workflows on failure-prone platforms

Guillaume Aupy, Anne Benoit,
Henri Casanova & Yves Robert

ENS Lyon

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

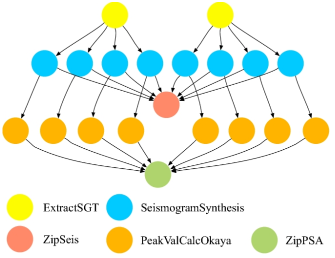
CR02 - 2016/2017

Motivation

Many HPC applications can be represented as computational workflows.

Represented by a DAG:

- Vertices are tightly coupled parallel tasks
- Edges represent data dependencies



Eg. CyberShake workflow (used to characterize earthquake hazards) as presented by Pegasus.

Outline

1. Models
   - Platform
   - Fault-tolerance
   - Application
2. Results
   - Computation of the expected makespan
   - NP-hardness, polynomial algorithms for special graphs
3. Efficient heuristic evaluation
   - Heuristics
   - Evaluation
4. Conclusion

Platform and processor assignments

Failure-prone platform:

- p processors
- Exponential failure distribution, MTBF: $\mu = \frac{1}{\lambda}$

Mixed parallelism is hard. Even without failures.

- Assignment of processors to tasks? *(throughput)*
- Traversal of the graph? *(scheduling)*
- Data redistribution? *(model redistribution cost)*

Simplified scenario

Each task uses all available processors; workflow is linearized.

Platform and processor assignments

Failure-prone platform:

- p processors
- Exponential failure distribution, MTBF: $\mu = \frac{1}{\lambda}$

Mixed parallelism is hard. Even without failures.

- Assignment of processors to tasks? *(throughput)*
- Traversal of the graph? *(scheduling)*
- Data redistribution? *(model redistribution cost)*

Simplified scenario

Each task uses all available processors; workflow is linearized.

Platform and processor assignments

Failure-prone platform:

- *p* processors
- Exponential failure distribution, MTBF: $\mu = \frac{1}{\lambda}$

~~Mixed parallelism is hard. Even without failures~~

- ~~Assignment of processors to tasks? *(throughput)*~~
- ~~Traversal of the graph? *(scheduling)*~~
- ~~Data redistribution? *(model redistribution cost)*~~

Simplified scenario

Each task uses all available processors; workflow is linearized.

Fault tolerance

We use the checkpoint technique for fault-tolerance.

Checkpointing within tasks is expensive or hard:

- Expensive: for application-agnostic checkpoint, need to checkpoint the full image
- Hard: modifying the implementation of the tasks to checkpoint only what is necessary

Checkpoint model

We only checkpoint the output data of tasks.

Application model

Given a DAG: $\mathcal{G} = (V, E)$. For all tasks T_i, we know:

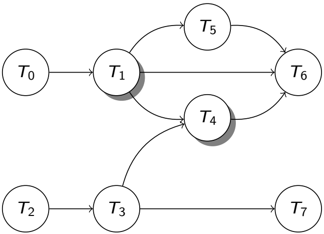
w_i: their execution time
c_i: the time to checkpoint their output
r_i: the time to recover their output

DAG-CkptSched

- In which order should the tasks be executed?
- Which tasks should be checkpointed?

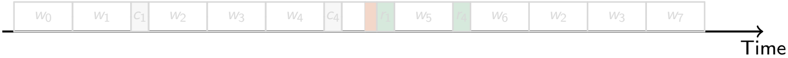
We want to minimize the expected execution time.

Motivational example

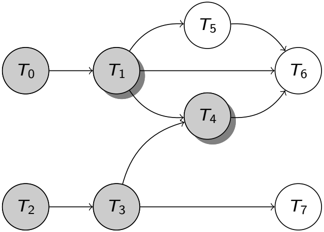


A solution (schedule):

Order: $T_0\, T_1\, T_2\, T_3\, T_4\, T_5\, T_6\, T_7$
Ckpted: T_1, T_4

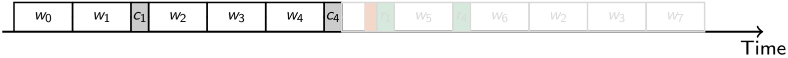


Motivational example

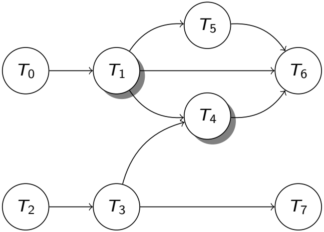


A solution (schedule):

Order: $T_0\ T_1\ T_2\ T_3\ T_4$ $\mathbf{T_5}$ $T_6\ T_7$
Ckpted: T_1, T_4

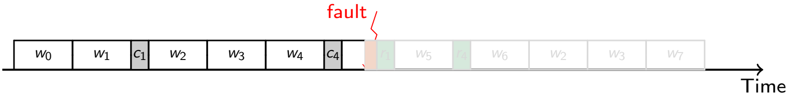


Motivational example

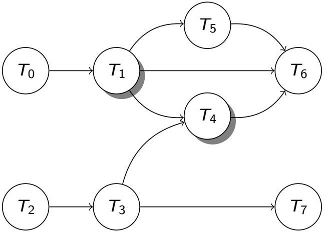


A solution (schedule):

Order: $T_0\ T_1\ T_2\ T_3\ T_4\ \mathbf{T_5}\ T_6\ T_7$
Ckpted: $T_1,\ T_4$

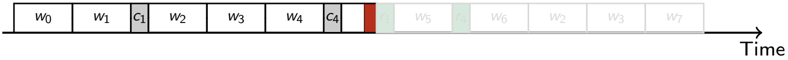


Motivational example

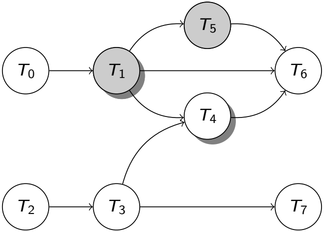


A solution (schedule):

Order: $T_0\ T_1\ T_2\ T_3\ T_4\ \mathbf{T_5}\ T_6\ T_7$
Ckpted: T_1, T_4

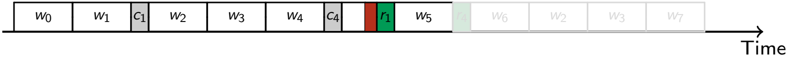


Motivational example

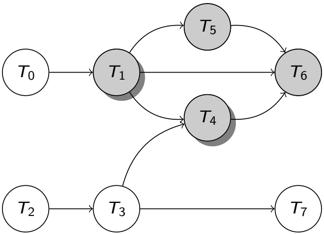


A solution (schedule):

Order: $T_0\ T_1\ T_2\ T_3\ T_4\ \mathbf{T_5}\ T_6\ T_7$
Ckpted: T_1, T_4

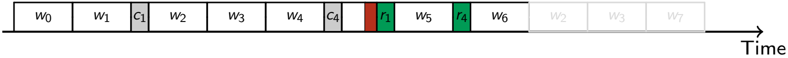


Motivational example

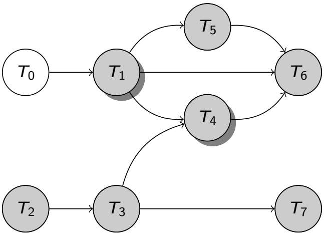


A solution (schedule):

Order: $T_0\ T_1\ T_2\ T_3\ T_4\ T_5\ \mathbf{T_6}\ T_7$
Ckpted: T_1, T_4



Motivational example



A solution (schedule):

Order: $T_0\, T_1\, T_2\, T_3\, T_4\, T_5\, T_6\, \mathbf{T_7}$
Ckpted: T_1, T_4

Outline

Previous results (Bougeret et al. 2011)

Let $\mathbb{E}[t(w; c; r)]$ the expected time to execute a single application:

- w sec. of computation in a fault-free execution
- c sec. to checkpoint the output
- r sec. to recover (if a failure occurs)

$$\mathbb{E}[t(w; c; r)] = e^{\lambda r} \left(\frac{1}{\lambda} + D \right) \left(e^{\lambda(w+c)} - 1 \right)$$

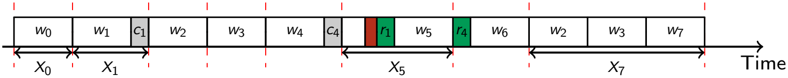
Theorem

Given a DAG, and a schedule for this DAG, it is possible to compute the expected execution time in polynomial time.

Theorem

Given a DAG, and a schedule for this DAG, it is possible to compute the expected execution time in polynomial time.

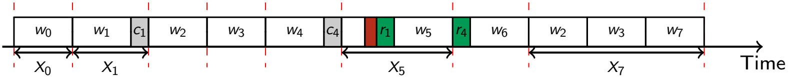
X_i: execution time between the end of the first successful execution of T_{i-1} and the end of the first successful execution of T_i (RV).



Theorem

Given a DAG, and a schedule for this DAG, it is possible to compute the expected execution time in polynomial time.

X_i: execution time between the end of the first successful execution of T_{i-1} and the end of the first successful execution of T_i (RV).



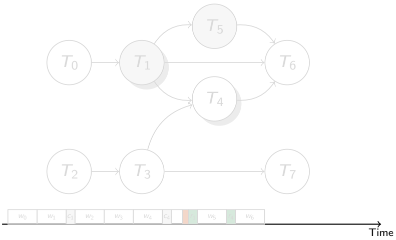
We want to compute $\mathbb{E}[\sum_i X_i] = \sum_i \mathbb{E}[X_i]$.

Sketch of Proof (1/2)

Z_k^i: "There was a fault during X_k and no fault during X_{k+1} to X_{i-1}" *(= when starting X_i, the last fault was during X_k)*.

$$\rightarrow \mathbb{E}[X_i] = \sum_{k=0}^{i-1} \mathbb{P}(Z_k^i)\mathbb{E}[X_i|Z_k^i]$$

$T_i^{\downarrow k}$: all T_j's whose output should be computed during X_i if Z_k^i.
We separate their impact on the execution time into W_k^i and R_k^i (depending upon whether T_j was checkpointed).

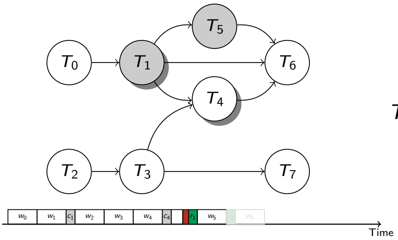


Sketch of Proof (1/2)

Z_k^i: "There was a fault during X_k and no fault during X_{k+1} to X_{i-1}" *(= when starting X_i, the last fault was during X_k).*

$$\rightarrow \mathbb{E}[X_i] = \sum_{k=0}^{i-1} \mathbb{P}(Z_k^i)\mathbb{E}[X_i | Z_k^i]$$

$T_i^{\downarrow k}$: all T_j's whose output should be computed during X_i if Z_k^i.
We separate their impact on the execution time into W_k^i and R_k^i (depending upon whether T_j was checkpointed).



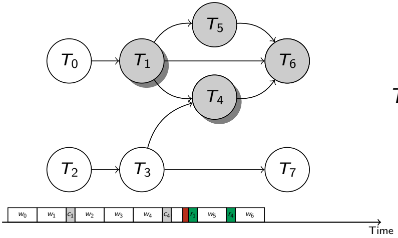
$$T_4 \in T_6^{\downarrow 5} \qquad R_5^6 = r_4$$
$$T_1, T_5, T_2, T_3 \notin T_6^{\downarrow 5}$$

Sketch of Proof (1/2)

Z_k^i: "There was a fault during X_k and no fault during X_{k+1} to X_{i-1}" (= *when starting X_i, the last fault was during X_k*).

$$\to \mathbb{E}[X_i] = \sum_{k=0}^{i-1} \mathbb{P}(Z_k^i)\mathbb{E}[X_i|Z_k^i]$$

$T_i^{\downarrow k}$: all T_j's whose output should be computed during X_i if Z_k^i.
We separate their impact on the execution time into W_k^i and R_k^i (depending upon whether T_j was checkpointed).



$T_2, T_3 \in T_7^{\downarrow 5}$ $\quad W_5^7 = w_2 + w_3$

Sketch of Proof (2/2)

- Let i, k s.t. $0 \leq k < i - 1$:

$$\mathbb{P}(Z^i_{i-1}) = 1 - \sum_{k=0}^{i-2} \mathbb{P}(Z^i_k)$$

$$\mathbb{P}(Z^i_k) = \; e^{-\lambda \sum_{j=k+1}^{i-1} \left(W^j_k + R^j_k + w_j + \delta_j c_j \right)} \; . \; \mathbb{P}(Z^{k+1}_k)$$

Sketch of Proof (2/2)

- Let i, k s.t. $0 \leq k < i-1$:

$$\mathbb{P}(Z^i_{i-1}) = 1 - \sum_{k=0}^{i-2} \mathbb{P}(Z^i_k)$$

$$\mathbb{P}(Z^i_k) = \boxed{e^{-\lambda \sum_{j=k+1}^{i-1}\left(W^j_k + R^j_k + w_j + \delta_j c_j\right)}} \cdot \mathbb{P}(Z^{k+1}_k)$$

Probability of successful execution of X_{k+1} to X_{i-1} given that there is a fault in X_k.

$$X_j = W^j_k + R^j_k + w_j + \delta_j c_j \text{ when } Z^i_k$$

Sketch of Proof (2/2)

- Let i, k s.t. $0 \leq k < i - 1$:

$$\mathbb{P}(Z^i_{i-1}) = 1 - \sum_{k=0}^{i-2} \mathbb{P}(Z^i_k)$$

$$\mathbb{P}(Z^i_k) = e^{-\lambda \sum_{j=k+1}^{i-1}\left(W^j_k + R^j_k + w_j + \delta_j c_j\right)} \cdot \mathbb{P}(Z^{k+1}_k)$$

Probability that there is a fault in X_k.

Sketch of Proof (2/2)

- Let i, k s.t. $0 \leq k < i-1$:

$$\mathbb{P}(Z_{i-1}^i) = 1 - \sum_{k=0}^{i-2} \mathbb{P}(Z_k^i)$$

$$\mathbb{P}(Z_k^i) = \; e^{-\lambda \sum_{j=k+1}^{i-1}\left(W_k^j + R_k^j + w_j + \delta_j c_j\right)} \; . \; \mathbb{P}(Z_k^{k+1})$$

- $\mathbb{E}[X_i | Z_k^i] =$
$\mathbb{E}[t\left(\; W_k^i + R_k^i + w_i \; ; \; \delta_i c_i \; ; \; W_i^i + R_i^i - \left(W_k^i + R_k^i\right) \; \right)]$

Sketch of Proof (2/2)

- Let i, k s.t. $0 \leq k < i-1$:

$$\mathbb{P}(Z^i_{i-1}) = 1 - \sum_{k=0}^{i-2} \mathbb{P}(Z^i_k)$$

$$\mathbb{P}(Z^i_k) = e^{-\lambda \sum_{j=k+1}^{i-1} \left(W^j_k + R^j_k + w_j + \delta_j c_j\right)} \cdot \mathbb{P}(Z^{k+1}_k)$$

- $\mathbb{E}[X_i | Z^i_k] =$
$\mathbb{E}[t\left(W^i_k + R^i_k + w_i \,;\, \delta_i c_i \,;\, W^i_i + R^i_i - \left(W^i_k + R^i_k\right) \right)]$

By definition of W^i_k and R^i_k, this is the work to be done after Z^i_k.

Sketch of Proof (2/2)

- Let i, k s.t. $0 \leq k < i - 1$:

$$\mathbb{P}(Z^i_{i-1}) = 1 - \sum_{k=0}^{i-2} \mathbb{P}(Z^i_k)$$

$$\mathbb{P}(Z^i_k) = e^{-\lambda \sum_{j=k+1}^{i-1} \left(W^j_k + R^j_k + w_j + \delta_j c_j \right)} \cdot \mathbb{P}(Z^{k+1}_k)$$

- $\mathbb{E}[X_i | Z^i_k] =$
$\mathbb{E}[t \left(W^i_k + R^i_k + w_i \; ; \; \delta_i c_i \; ; \; W^i_i + R^i_i - \left(W^i_k + R^i_k \right) \right)]$

$\delta_i = 0$ if T_i is not checkpointed, 1 otherwise

Sketch of Proof (2/2)

- Let i, k s.t. $0 \leq k < i-1$:

$$\mathbb{P}(Z^i_{i-1}) = 1 - \sum_{k=0}^{i-2} \mathbb{P}(Z^i_k)$$

$$\mathbb{P}(Z^i_k) = e^{-\lambda \sum_{j=k+1}^{i-1} \left(W^j_k + R^j_k + w_j + \delta_j c_j\right)} \cdot \mathbb{P}(Z^{k+1}_k)$$

- $\mathbb{E}[X_i | Z^i_k] =$
$\mathbb{E}[t\left(W^i_k + R^i_k + w_i \; ; \; \delta_i c_i \; ; \; W^i_i + R^i_i - (W^i_k + R^i_k) \right)]$

If there is a failure during X_i, then the work to be done becomes $W^i_i + R^i_i + w_i$.

Sketch of Proof (2/2)

- Let i, k s.t. $0 \leq k < i-1$:

$$\mathbb{P}(Z_{i-1}^i) = 1 - \sum_{k=0}^{i-2} \mathbb{P}(Z_k^i)$$

$$\mathbb{P}(Z_k^i) = \; e^{-\lambda \sum_{j=k+1}^{i-1} \left(W_k^j + R_k^j + w_j + \delta_j c_j \right)} \; . \; \mathbb{P}(Z_k^{k+1})$$

- $\mathbb{E}[X_i | Z_k^i] =$
$\mathbb{E}[t \left(\; W_k^i + R_k^i + w_i \; ; \; \delta_i c_i \; ; \; W_i^i + R_i^i - \left(W_k^i + R_k^i \right) \; \right)]$
- LEMMA: We can compute W_k^i and R_k^i in polynomial time. $\square$

Other results

Theorem (Complexity)

DAG-CkptSched *for fork DAGs can be solved in linear time.*
DAG-CkptSched *for join DAGs is NP-complete.*

Theorem

DAG-CkptSched *for a join DAG where $c_i = c$ and $r_i = r$ for all i can be solved in quadratic time.*

Open Problem

Complexity of DAG-CkptSched for a general DAG where $c_i = c$ and $r_i = r$ for all i?

Other results

Theorem (Complexity)

DAG-CkptSched *for fork DAGs can be solved in linear time.*
DAG-CkptSched *for join DAGs is NP-complete.*

Theorem

DAG-CkptSched *for a join DAG where $c_i = c$ and $r_i = r$ for all i can be solved in quadratic time.*

Open Problem

Complexity of DAG-CkptSched for a general DAG where $c_i = c$ and $r_i = r$ for all i?

Outline

1. Models
   - Platform
   - Fault-tolerance
   - Application
2. Results
   - Computation of the expected makespan
   - NP-hardness, polynomial algorithms for special graphs
3. Efficient heuristic evaluation
   - Heuristics
   - Evaluation
4. Conclusion

Efficient heuristic evaluation

Designing efficient heuristics used to take:

- Numerous, time-consuming and expensive stochastic experiments on an actual platform
- Numerous, time-consuming simulations with a fault-generator

Now we can simply compute the expected makespan!

2-step heuristics

Linearization strategies

- DF Depth First (prio tasks by decreasing outweight)
- BF Breadth First (prio tasks by decreasing outweight)
- RF Random First

Checkpoint strategies

- CkNvr Never checkpoint (default)
- CkAlws Always checkpoint (default)

Below: extensive search for |checkpoint| from 1 to $n-1$

- CkPer "Periodic" checkpoint
- CkW Prioritize large w_i
- CkC Prioritize small c_i

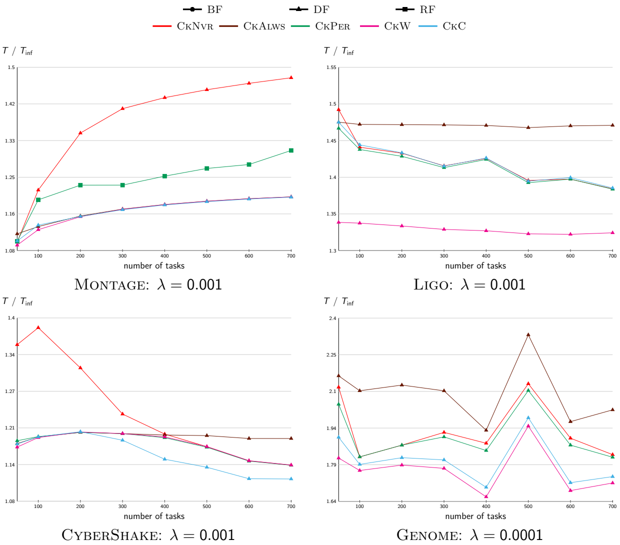
Methodology

We use the Pegasus Workflow Generator to generate realistic synthetic workflows:

MONTAGE:	mosaics of the sky	Average $w_i \approx 10$s.
LIGO:	gravitational waveforms	Average $w_i \approx 220$s.
CYBERSHAKE:	earthquake hazards	Average $w_i \approx 25$s.
GENOME:	genome sequence processing	Average $w_i > 1000$s.

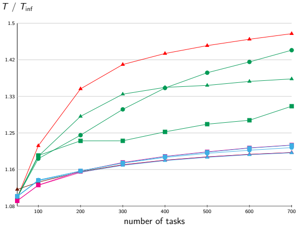
- We plot the ratio of the expected execution time (T) over the execution time of a failure-free, checkpoint-free execution (T_{inf})
- No downtime
- $c_i = r_i = 0.1 w_i$ (similar for other values)

Results

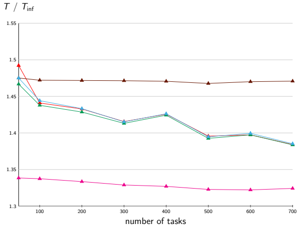

BF
DF
RF
CkNvr
CkAlws
CkPer
CkW
CkC
T / T_inf
number of tasks
Montage: λ = 0.001
Ligo: λ = 0.001
CyberShake: λ = 0.001
Genome: λ = 0.0001

Results

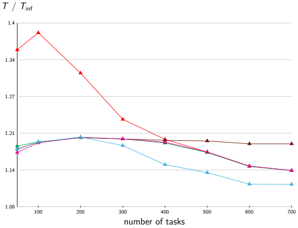
BF DF RF
CkNvr CkAlws CkPer CkW CkC



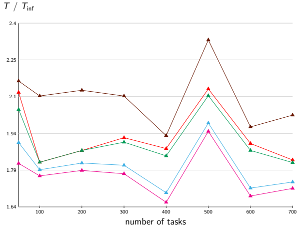
Montage: $\lambda = 0.001$



Ligo: $\lambda = 0.001$

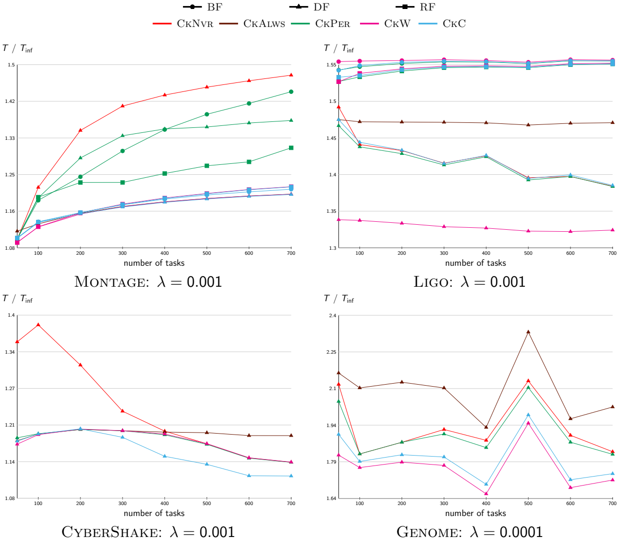


CyberShake: $\lambda = 0.001$



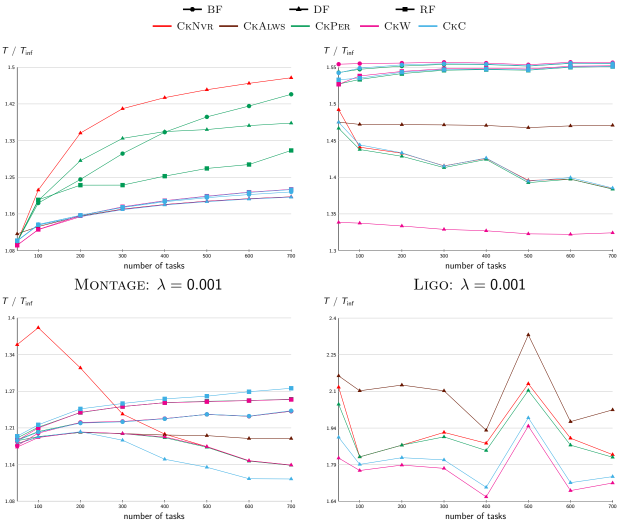
Genome: $\lambda = 0.0001$

Results



BF DF RF
CkNvr CkAlws CkPer CkW CkC
T / T_{inf}
number of tasks
Montage: $\lambda = 0.001$
Ligo: $\lambda = 0.001$
CyberShake: $\lambda = 0.001$
Genome: $\lambda = 0.0001$

Results



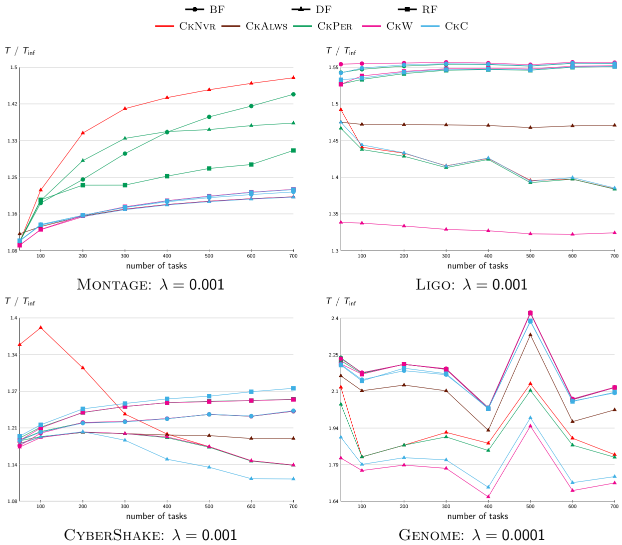
Montage: $\lambda = 0.001$

Ligo: $\lambda = 0.001$

CyberShake: $\lambda = 0.001$

Genome: $\lambda = 0.0001$

Results


BF
DF
RF
CkNvr
CkAlws
CkPer
CkW
CkC
T / T_inf
number of tasks
Montage: λ = 0.001
Ligo: λ = 0.001
CyberShake: λ = 0.001
Genome: λ = 0.0001

- BF is not a good heuristic for linearization
- CkPer is not a good heuristic for checkpointing DAGs

- DF seems to be a good heuristic for linearization
- CkW, CkC seem to be good heuristics for checkpointing (especially CkW)

Outline

Conclusion

- Framework: Applications are scheduled on the whole platform, subject to IID exponentially distributed failures.
- A polynomial time algorithm to compute the expected makespan for general DAGs.
- Polynomial-time algorithm for fork DAGs, some join DAGs, intractability in the general case.
- Evaluation of several heuristics on representative workflow configurations.
  $\rightarrow$ Periodic checkpoint is not good for general DAGs.

Future directions

- Our key result has opened the road to designing efficient heuristics.

- On a theoretical point of view:
  (i) Non-blocking checkpoint
  (ii) Remove linearization assumption