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 - 2015/2016
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.
Given a DAG: $G = (V, E)$. 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: $T_0 \, T_1 \, T_2 \, T_3 \, T_4 \, T_5 \, T_6 \, T_7$

Ckpted: $T_1, \, T_4$
Motivational example

A solution (schedule):

Order: $T_0 T_1 T_2 T_3 T_4 T_5 T_6 T_7$

Ckpted: $T_1, T_4$
Motivational example

A solution (schedule):

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

Ckpted: $T_1, \ T_4$
Motivational example

A solution (schedule):

Order: $T_0 T_1 T_2 T_3 T_4 T_5 T_6 T_7$

Ckpted: $T_1, T_4$
Motivational example

A solution (schedule):

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

Ckpted: $T_1, T_4$
Motivational example

A solution (schedule):

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

Ckpted: $T_1, T_4$
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
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.

\[ X_i: \text{ execution time between the end of the first successful execution of } T_{i-1} \text{ and the end of the first successful execution of } T_i \text{ (RV).} \]
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 \( \mathbb{E}[\sum_i X_i] = \sum_i \mathbb{E}[X_i] \).
\( Z_i^k \): “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 \( X_k \)).

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

\( T_i^{\downarrow k} \): all \( T_j \)'s whose output should be computed during \( X_i \) if \( Z_i^k \).

We separate their impact on the execution time into \( W_i^k \) and \( R_i^k \) (depending upon whether \( T_j \) was checkpointed).
Sketch of Proof (1/2)

\( Z^i_k \): “There was a fault during \( X_k \) and no fault during \( X_{k+1} \) to \( X_{i-1} \)”

\( = \text{when starting } X_i, \text{ the last fault was during } X_k \).

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

\( T^\downarrow_k \): all \( T_j \)’s whose output should be computed during \( X_i \) if \( Z^i_k \).

We separate their impact on the execution time into \( W^i_k \) and \( R^i_k \)

(depending upon whether \( T_j \) was checkpointed).
Sketch of Proof (1/2)

$Z^i_k$: “There was a fault during $X_k$ and no fault during $X_{k+1}$ to $X_{i-1}$”  
($= \text{when starting } X_i, \text{ the last fault was during } X_k$).

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

$T^↓_i$: all $T_j$’s whose output should be computed during $X_i$ if $Z^i_k$.
We separate their impact on the execution time into $W^i_k$ and $R^i_k$  
(depending upon whether $T_j$ was checkpointed).

$T_2, T_3 \in T^↓_7$ \hspace{1cm} $W^7_5 = w_2 + w_3$
Let $i, k$ s.t. $0 \leq k < i - 1$:

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

$$
\mathbb{P}(Z_k^i) = e^{-\lambda \sum_{j=k+1}^{i-1} (W_j^i + R_j^i + w_j + \delta_j c_j)} \cdot \mathbb{P}(Z_{k+1}^{k+1})
$$
Sketch of Proof (2/2)

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

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

\[
P(Z_k^i) = e^{-\lambda \sum_{j=k+1}^{i-1} (W_k^j + R_k^j + w_j + \delta_j c_j)} \cdot P(Z_{k+1}^i)\]

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

$X_j = W_k^j + R_k^j + w_j + \delta_j c_j$ when $Z_k^i$
Sketch of Proof (2/2)

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

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

$$P(Z_k^i) = e^{-\lambda \sum_{j=k+1}^{i-1} (W_j^i + R_j^i + w_j + \delta_j c_j)} \cdot P(Z_{k+1}^i)$$

Probability that there is a fault in $X_k$. 

Anne.Benoit@ens-lyon.fr CR02 DAG scheduling with failures 12/22
Sketch of Proof (2/2)

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

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

$$P(Z_k^i) = e^{-\lambda \sum_{j=k+1}^{i-1} (W_k^j + R_k^j + \delta_j c_j)} \cdot P(Z_{k+1}^{k+1})$$

- $E[X_i|Z_k^i] = E[t \left( W_k^i + R_k^i + \delta_i c_i \ ; \ W_i^i + R_i^i - (W_k^i + R_k^i) \right)]$
Sketch of Proof (2/2)

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

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

\[ P(Z_k^i) = e^{-\lambda \sum_{j=k+1}^{i-1} (W_j^i + R_j^i + w_j + \delta_j c_j)} \cdot P(Z_{k+1}^{i}) \]

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

By definition of $W_k^i$ and $R_k^i$, this is the work to be done after $Z_k^i$. 

Anne.Benoit@ens-lyon.fr

CR02

DAG scheduling with failures
Sketch of Proof (2/2)

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

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

$$
P(Z_k^i) = e^{-\lambda \sum_{j=k+1}^{i-1} (W_j^i + R_j^i + w_j + \delta_j c_j)} \cdot P(Z_{k+1}^i)
$$

- $E[X_i | Z_k^i] =

$$
E[t \left( W_k^i + R_k^i + w_i ; \delta_i c_i ; W_i^i + R_i^i - (W_k^i + R_k^i) \right)]
$$

$\delta_i = 0$ if $T_i$ is not checkpointed, 1 otherwise
Sketch of Proof (2/2)

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

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

\[
P(Z_k^i) = e^{-\lambda \sum_{j=k+1}^{i-1} (W_j^i + R_j^i + w_j + \delta_j c_j)} \cdot P(Z_{k+1}^k)
\]

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

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. $0 \leq k < i - 1$:

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

$$\mathbb{P}(Z_k^i) = e^{-\lambda \sum_{j=k+1}^{i-1} (W_j^i + R_j^i + w_j + \delta_j c_j)} \cdot \mathbb{P}(Z_{k+1}^{i-1})$$

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

- LEMMA: We can compute $W_k^i$ and $R_k^i$ in polynomial time. 

\[\square\]
Other results

Theorem (Complexity)

\textbf{DAG-CkptSched} for fork DAGs can be solved in linear time.
\textbf{DAG-CkptSched} for join DAGs is NP-complete.

Theorem

\textbf{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 \textbf{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 $c_i$
We use the Pegasus Workflow Generator to generate realistic synthetic workflows:

**Montage:** mosaics of the sky  \( \text{Average } w_i \approx 10\text{s.} \)

**Ligo:** gravitational waveforms  \( \text{Average } w_i \approx 220\text{s.} \)

**CyberShake:** earthquake hazards  \( \text{Average } w_i \approx 25\text{s.} \)

**Genome:** genome sequence processing  \( \text{Average } w_i > 1000\text{s.} \)

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

Montage: $\lambda = 0.001$

CyberShake: $\lambda = 0.001$

Ligo: $\lambda = 0.001$

Genome: $\lambda = 0.0001$
Results

Montage: $\lambda = 0.001$

Ligo: $\lambda = 0.001$

CyberShake: $\lambda = 0.001$

Genome: $\lambda = 0.0001$
Results

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

Montage: $\lambda = 0.001$

Ligo: $\lambda = 0.001$

CyberShake: $\lambda = 0.001$

Genome: $\lambda = 0.0001$
- 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

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
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.
  → 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