Tools and Algorithms for Coping with Uncertainty in Application Scheduling on Distributed Platforms

PhD Defense

Louis-Claude CANON

Supervisor: Emmanuel JEANNOT

October 18, 2010
Distributed Computing

Distributed Platform

- Network of entities.
- In a computing platform, each entity has one processor.
- Use case examples: data sharing, parallel computation.
Distributed Computing

Distributed Platform

- Network of entities.
- In a computing platform, each entity has one processor.
- Use case examples: data sharing, parallel computation.

Computation

- Web services: Amazon.com.
- Scientific simulations: weather prediction (simulation of typhoon Mawar in 2005).
- Research projects: SETI@home.
A parallel application consists of a set of tasks.

\[ t_1, t_2, t_3, t_4 \]
A parallel application consists of a set of tasks.

Each task processes data and produces a result.
A parallel application consists of a set of tasks.

Each task processes data and produces a result.

Some tasks require the results of other tasks (the precedences are specified by a task graph).
A parallel computing platform consists of a set of interconnected processors.

Each task can be computed by one machine.

The execution durations may be a function of the processor speeds and task costs.

Communication durations are determined by the network capacity.
Schedule Structure

A schedule may be defined by (not inclusively):

- **a mapping** each task is assigned to a processor
- **dates** start and end times of each execution
- **an order** the order in which each task must be executed
Schedule Structure

A schedule may be defined by (not inclusively):

- **a mapping** each task is assigned to a processor
- **dates** start and end times of each execution
- **an order** the order in which each task must be executed

Scheduling Strategies

- **offline** scheduling decisions are taken before any computation
- **online** decisions are taken while the tasks are executed
Scheduling

Schedule Structure
A schedule may be defined by (not inclusively):
- a mapping each task is assigned to a processor
- dates start and end times of each execution
- an order the order in which each task must be executed

Scheduling Strategies
- offline scheduling decisions are taken before any computation
- online decisions are taken while the tasks are executed

Example of Criteria
- efficiency total duration of a schedule
- fairness for multiple users/organization
Uncertainty

Object
- computation duration
- computation success
- result correctness
Uncertainty

Object

- computation duration
- computation success
- result correctness

Consequence

- unpredictability of the performances
- failure of the schedule
- invalidity of the solution
## Uncertainty Characteristics

### Nature [Haimes, 2009]

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>methodological</td>
<td>limitation(s) of the method (e.g., model simplification)</td>
</tr>
<tr>
<td>epistemic</td>
<td>inaccessible knowledge (e.g., online task submission)</td>
</tr>
<tr>
<td>aleatory</td>
<td>stochastic variability (e.g., hardware fault)</td>
</tr>
</tbody>
</table>
Uncertainty Characteristics

Nature [Haimes, 2009]

- **methodological** limitation(s) of the method (e.g., model simplification)
- **epistemic** inaccessible knowledge (e.g., online task submission)
- **aleatory** stochastic variability (e.g., hardware fault)

Origin

- hardware
- software
- human
## Uncertainty and Scheduling

<table>
<thead>
<tr>
<th>Uncertainty</th>
<th>Scheduling</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Object</strong></td>
<td><strong>Nature</strong></td>
</tr>
<tr>
<td>computation duration</td>
<td>methodological</td>
</tr>
<tr>
<td>computation success</td>
<td>aleatory</td>
</tr>
<tr>
<td>result correctness</td>
<td>epistemic</td>
</tr>
</tbody>
</table>

---

Outline

1. Context
2. Robustness
3. Reliability
4. Precision
5. Conclusion
1. Context

2. Robustness
   - Louis-Claude Canon and Emmanuel Jeannot, A Comparison of Robustness Metrics for Scheduling DAGs on Heterogeneous Systems, in heteroPar’07, Austin, Texas, September 2007.

3. Reliability

4. Precision

5. Conclusion
Application

- The parallel application is specified by a task graph.
- For each precedence constraint, data need to be transferred.
Application

- The parallel application is specified by a task graph.
- For each precedence constraint, data need to be transferred.

Platform

- The parallel computing platform consists of a set of processors.
- Processors are unrelated: the duration of each task is specific to the executing processor.
- Each pair of machines is interconnected by a dedicated link.
Uncertainty on Computation Durations

- Evaluating analytically the duration of a computation is difficult because the application and the platform are complex (methodological uncertainty).
- Durations are *random variables*.
Uncertainty on Computation Durations

- Evaluating analytically the duration of a computation is difficult because the application and the platform are complex (methodological uncertainty).
- Durations are random variables.

Random Variables

- Each duration is represented by a set of values and probabilities.
- Each probability gives the likelihood that the corresponding duration occurs during a given execution.
Efficiency

Efficiency is defined by the duration of the schedule execution ($C_{\text{max}}$). Evaluating this duration consists in evaluating a *stochastic DAG*.
Efficiency

Efficiency is defined by the duration of the schedule execution ($C_{\text{max}}$). Evaluating this duration consists in evaluating a stochastic DAG.

Stochastic DAG Evaluation

We prove the problem to be #P'-Complete. #P' is a generalization of counting problems (#P) for reliability evaluation problems [Bodlaender et al., 2004].
Efficiency
Efficiency is defined by the duration of the schedule execution ($C_{\text{max}}$). Evaluating this duration consists in evaluating a stochastic DAG.

Stochastic DAG Evaluation
We prove the problem to be #P'-Complete. #P' is a generalization of counting problems (#P) for reliability evaluation problems [Bodlaender et al., 2004].

Remark on the Complexity Class
Any #P' problem based on a NP-Complete problem is #P'-Complete. However, this evaluation problem corresponds to a P problem.
Robustness

Capacity of a system to maintain its performances despite variations (criterion depending on an efficiency measure).
Robustness

Capacity of a system to maintain its performances despite variations (criterion depending on an efficiency measure).

Robust Schedule

- Random variables model the variations in the inputs.
- A schedule is robust if its total duration remains stable despite the task durations variations.
## Robustness Measures

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Wu et al., 1994]</td>
<td>Same methodological approach as ours but with unavailabilities.</td>
</tr>
<tr>
<td>[Shestak et al., 2006]</td>
<td>Stochastic robustness metric and deterministic robustness metric.</td>
</tr>
<tr>
<td>[Bölöni et al., 2002]</td>
<td>Total slack and differential entropy.</td>
</tr>
</tbody>
</table>
### Related Work

#### Robustness Measures

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Wu et al., 1994]</td>
<td>Same methodological approach as ours but with unavailabilities.</td>
</tr>
<tr>
<td>[Shestak et al., 2006]</td>
<td>Stochastic robustness metric and deterministic robustness metric.</td>
</tr>
<tr>
<td>[Bölöni et al., 2002]</td>
<td>Total slack and differential entropy.</td>
</tr>
</tbody>
</table>

#### Optimization

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Davenport et al., 2001]</td>
<td>Insertion of temporal slack.</td>
</tr>
<tr>
<td>[Fargier et al., 2003]</td>
<td>Scheduling techniques using fuzzy logic.</td>
</tr>
</tbody>
</table>
Many measures exist in the literature. For a robust schedule, we expect:

- a small **standard deviation** of the total duration
- a small **differential entropy** of the total duration
- a large **expected slack** (large temporal protection)
- values for the **stochastic metrics** close to 1
- a **lateness probability** close to 0
- a small **99th percentile** of the total duration (almost equivalent to the expected value of the total duration)
### Application

layered task graph with 1000 tasks, 10% uncertainty, beta distribution

### Platform

50 processors (2.5 GFLPOS each)

### Schedules

5000 randomly generated
Robustness Measure

The standard deviation of the total duration is equivalent to the entropy of the total duration and one of the stochastic metrics.
Robustness Measure

The standard deviation of the total duration is equivalent to the entropy of the total duration and one of the stochastic metrics.

Expected Value of the Slack

Invalid measure of robustness (no correlation with other robustness measures).
Robustness Measure
The standard deviation of the total duration is equivalent to the entropy of the total duration and one of the stochastic metrics.

Expected Value of the Slack
Invalid measure of robustness (no correlation with other robustness measures).

Multi-criteria Problem
Correlation between expected value and standard deviation of the total duration.
However, the efficiency and the robustness are not equivalent.
Pareto Region Study

Methods

- Greedy construction: aggregate both criteria and schedule each task by making the best local choice.
- Multi-criteria evolutionary algorithm: we prove its convergence.
Pareto Region Study

Methods

- Greedy construction: aggregate both criteria and schedule each task by making the best local choice.
- Multi-criteria evolutionary algorithm: we prove its convergence.

Conclusion

Several methods that estimate the Pareto-front.
The parallel application is specified by a task graph.
Application

- The parallel application is specified by a task graph.

Platform

- The parallel computing platform consists of a set of processors.
- Processors are unrelated: the duration of each task is specific to the executing processor.
- Immediate synchronizations occur on the network.
Reliability

Fault Model

Uncertainty on Computation Successes

Each machine may fail during the execution of a task with a non-zero probability (aleatory uncertainty).
Fault Model

Uncertainty on Computation Successes
Each machine may fail during the execution of a task with a non-zero probability (aleatory uncertainty).

Transient failures
An execution fails but the processor recovers immediately. Example: arithmetic/software errors or recoverable hardware faults (power loss).
Fault Model

Uncertainty on Computation Successes
Each machine may fail during the execution of a task with a non-zero probability (aleatory uncertainty).

Transient failures
An execution fails but the processor recovers immediately. Example: arithmetic/software errors or recoverable hardware faults (power loss).

Fail-stop failures
A processor dies until the end of the schedule (all remaining tasks fail). Example: hardware resource crashes, recovery of a loaned machine by a user during a cycle-stealing episode.
Scheduling with Replication

General policy

Each task can be scheduled after at least one replica of each of its predecessors is finished (if there is no failure).

\[ \begin{align*}
p_1 & \rightarrow t_2 \\
p_2 & \rightarrow t_1 \\
p_3 & \rightarrow t_2 \\
p_4 & \rightarrow t_1 \\
\end{align*} \]
Scheduling with Replication

General policy

Each task can be scheduled after at least one replica of each of its predecessors is finished (if there is no failure).

Strict policy

A task must be scheduled after all the end times of the replicas of its predecessors. Also called *replication for reliability scheme*. 
The reliability of a static schedule is the probability that it terminates successfully.

A schedule is successful if all tasks have at least one successful replica.

The execution of any replica is successful if at least one replica for each of its predecessors are successfully executed and if the processor does not fail during the execution (or has not yet been subjected to a fail-stop failure).
Related Work

Bi-criteria Scheduling

[Dongarra et al., 2007] Scheduling without replication.
[Jeannot et al., 2008] Scheduling without replication.
[Girault et al., 2009] Strict scheduling with transient faults.
Related Work

Bi-criteria Scheduling

[Dongarra et al., 2007] Scheduling without replication.

[Jeannot et al., 2008] Scheduling without replication.

[Girault et al., 2009] Strict scheduling with transient faults.

Reliability Block Diagram

[Bream, 1995] Introduce a diagram-based technique for reliability evaluation.

This problem is considered difficult (NP-Hard).
Main Contribution

We prove the general problem to be \#P'-Complete.
Main Contribution

We prove the general problem to be \#P'-Complete.

Intuition of the Exponential Evaluation

The problem can be solved by an exponential algorithm by considering each equiprobable scenarios. Then, the solution can be found by counting the number of scenarios that lead to a successful schedule.
Main Contribution

We prove the general problem to be $\#P'$-Complete.

Intuition of the Exponential Evaluation

The problem can be solved by an exponential algorithm by considering each equiprobable scenarios. Then, the solution can be found by counting the number of scenarios that lead to a successful schedule.
Main Contribution

We prove the general problem to be \#P’-Complete.

Intuition of the Exponential Evaluation

The problem can be solved by an exponential algorithm by considering each equiprobable scenarios. Then, the solution can be found by counting the number of scenarios that lead to a successful schedule.
Main Contribution

We prove the general problem to be \( \#P' \)-Complete.

Intuition of the Exponential Evaluation

The problem can be solved by an exponential algorithm by considering each equiprobable scenarios. Then, the solution can be found by counting the number of scenarios that lead to a successful schedule.
Monotonic Chain Case

Monotonicity Property

A schedule of chains is monotonic if the success of any task on processor $p_j$ depends only on the successes of the first $j$ processors.
Monotonic Chain Case

Monotonicity Property

A schedule of *chains* is *monotonic* if the success of any task on processor $p_j$ depends only on the successes of the first $j$ processors.

Evaluation Algorithm

The reliability of a monotonic schedule $\pi$ on a platform with *fail-stop* failures is $\text{rel}(\pi) = \Pr[J_{nm}]$.

\[
\begin{align*}
\Pr[J_{i1}] &= \Pr[E_{i1}] \\
\Pr[J_{1j}] &= \Pr[J_{1,j-1}] + (1 - \Pr[J_{1,j-1}]) \Pr[E_{1j}] \\
\Pr[J_{ij}] &= \Pr[J_{i,j-1}] + (\Pr[J_{\rho(i,j),j-1,i-1}] - \Pr[J_{i,j-1}]) \Pr[E_{ij}]
\end{align*}
\]
Uncertainty in Scheduling
1. Context

2. Robustness

3. Reliability

4. Precision
   - Louis-Claude Canon, Emmanuel Jeannot and Jon Weissman, A Dynamic Approach for Characterizing Collusion in Desktop Grids, in IEEE IPDPS, Atlanta, Georgia, April 2010.

5. Conclusion
The parallel application is specified by a set of independent tasks.
Application and Platform Models

**Application**
- The parallel application is specified by a set of independent tasks.

**Platform**
- The parallel computing platform consists of a set of processors.
- Processors are unrelated: the duration of each task is specific to the executing processor.
- Machines are connected to a central server via Internet.
Uncertainty on Result Correctness

- The results returned by each machine may be incorrects (epistemic uncertainty).
- 35% of SETI@home participants have given at least one incorrect result [Kondo et al., 2007].
- These Byzantine faults are due to unreliability or malicious behaviors (for credit increase).
Uncertainty on Result Correctness

- The results returned by each machine may be incorrects (epistemic uncertainty).
- 35% of SETI@home participants have given at least one incorrect result [Kondo et al., 2007].
- These Byzantine faults are due to unreliability or malicious behaviors (for credit increase).

Incorrectness Induces Redundancy

- Each task can be assigned to several machines.
- One of the received results must be selected as the final one by a certification mechanism.
**Collusion**

Some machines produce the same incorrect result for a given task.

[Diagram showing non-colluders and colluders for Task 1 and Task 2 with different outcomes indicating collusion and no collusion.]
Collusion

Some machines produce the same incorrect result for a given task.

Colluding groups

- Machines can be partitioned into several groups: machines in the same group always return the same result for a given task.
- Collusion occurs with a given probability.
- There may be cooperation between distinct colluding groups.
### Objective

Study the machine behaviors:

- estimate the probability that any pair of machines gives the same incorrect result for the same task.
- estimate the colluding groups composition.
Objective

Study the machine behaviors:
- estimate the probability that any pair of machines gives the same incorrect result for the same task.
- estimate the colluding groups composition.

Inputs

Chronological succession of events:
- \( <d, p, t, r> \) at time \( d \), machine \( p \) finishes task \( t \) and returns result \( r \).
- \( <d, t> \) at time \( d \), task \( t \) finishes.
Related Work

Scheduling

- [Zhao et al., 2005] quiz
- [Domingues et al., 2007] intermediate verifications
- [Silaghi et al., 2009] use a reputation system to detect colluders
- [Krings et al., 2005] a posteriori verification of results
- [Wong, 2005] no unreliability and verification of results
Related Work

Scheduling

[Zhao et al., 2005] quiz
[Domingues et al., 2007] intermediate verifications
[Silaghi et al., 2009] use a reputation system to detect colluders
[Krings et al., 2005] a posteriori verification of results
[Wong, 2005] no unreliability and verification of results

Reputation System

[Kamvar et al., 2003] EigenTrust algorithm
[Jøsang, 1999] subjective logic that allows reaching consensus
Interaction Model

**Two Interaction Representations**

Interaction between machines $i$ and $j$:

- **collusion** machines $i$ and $j$ collude together (collusion estimation $c_{ij}$)
- **agreement** machines $i$ and $j$ agree together (agreement estimation $a_{ij}$)
Two Interaction Representations

Interaction between machines $i$ and $j$:

- **collusion** machines $i$ and $j$ collude together (collusion estimation $c_{ij}$)
- **agreement** machines $i$ and $j$ agree together (agreement estimation $a_{ij}$)

Relations

- $a_{ij} \leq 1 + 2 \times c_{ij} - c_{ii} - c_{jj}$
- $c_{ij} \leq \frac{1 + a_{ij} - a_{1i} - a_{1j}}{2}$ (given that the index of the largest groups is 1)
Online Algorithm

Data structure

- Nodes correspond to sets of machines.
- Edges correspond to interaction characteristics (*agreement* here).
Online Algorithm

Data structure
- Nodes correspond to sets of machines.
- Edges correspond to interaction characteristics (*agreement* here).

Algorithm
- Initially, each machine is in a singleton.
- Estimated groups are successively merged and split.
Online Algorithm

Data structure

- Nodes correspond to sets of machines.
- Edges correspond to interaction characteristics (agreement here).

Algorithm

- Initially, each machine is in a singleton.
- Estimated groups are successively merged and split.
Convergence time

time needed in order to achieve a desirable accuracy

Stabilized accuracy
accuracy achieved after a large number of events
Outline

1 Context
2 Robustness
3 Reliability
4 Precision
5 Conclusion
## Uncertainty in Scheduling

<table>
<thead>
<tr>
<th>Uncertainty</th>
<th>Object</th>
<th>Nature</th>
<th>Origin</th>
<th>Type</th>
<th>Criterion</th>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>computation duration</td>
<td>computation duration</td>
<td>methodological</td>
<td>hardware, software</td>
<td>optimization</td>
<td>robustness</td>
<td>$R</td>
</tr>
<tr>
<td>computation success</td>
<td>computation success</td>
<td>aleatory</td>
<td>hardware</td>
<td>evaluation</td>
<td>reliability</td>
<td>$R</td>
</tr>
<tr>
<td>result correctness</td>
<td>result correctness</td>
<td>epistemic</td>
<td>software, human</td>
<td>characterization</td>
<td>precision</td>
<td>$R</td>
</tr>
</tbody>
</table>

*Note: The table above outlines the various types of uncertainty in scheduling, including the object, nature, origin, type, criterion, and problem.
Results

- Robustness
  - comparison of robustness measures (selection of the standard deviation)
  - multi-criteria methods that tackle robustness
Results

- Robustness
  - comparison of robustness measures (selection of the standard deviation)
  - multi-criteria methods that tackle robustness
- Reliability
  - taxonomy of reliability problems
  - complexity classes of corresponding evaluation problems
Results

- Robustness
  - comparison of robustness measures (selection of the standard deviation)
  - multi-criteria methods that tackle robustness

- Reliability
  - taxonomy of reliability problems
  - complexity classes of corresponding evaluation problems

- Precision
  - strong model of adversity
  - methods for estimating machine behaviors
Results

- Robustness
  - comparison of robustness measures (selection of the standard deviation)
  - multi-criteria methods that tackle robustness

- Reliability
  - taxonomy of reliability problems
  - complexity classes of corresponding evaluation problems

- Precision
  - strong model of adversity
  - methods for estimating machine behaviors

- Generic conclusion
  - probabilistic dimension and multi-criteria formulation
Defeat collusion using our proposed characterization system
Future Directions

- Defeat collusion using our proposed characterization system
- Develop online scheduling algorithm (relevant in case of high uncertainty)
Future Directions

- Defeat collusion using our proposed characterization system
- Develop online scheduling algorithm (relevant in case of high uncertainty)
- Explore other multi-criteria techniques (e.g., $\epsilon$-constraint method)
Future Directions

- Defeat collusion using our proposed characterization system
- Develop online scheduling algorithm (relevant in case of high uncertainty)
- Explore other multi-criteria techniques (e.g., $\epsilon$-constraint method)
- Propose tractable uncertainty models
Future Directions

- Defeat collusion using our proposed characterization system
- Develop online scheduling algorithm (relevant in case of high uncertainty)
- Explore other multi-criteria techniques (e.g., $\epsilon$-constraint method)
- Propose tractable uncertainty models
- Design guaranteed algorithm
Thank you for your attention.
Questions, comments, remarks, . . .
Defeating the Collusion

Uncertainty-Related Criterion

- **precision**: proportion of correctly certified results
- **overhead**: average duplication ratio

---

Effect of the trace

- **BOINC precision**
- **CAA precision**
- **BOINC overhead**
- **CAA overhead**

---

Conclusion
Literature on Collusion

Wong, An authentication protocol in Web-computing, IPDPS 2006

Propose a ring-based scheduling approach (duplication ratio of 2). Estimate the fraction and probability of collusion (given that non-colluding machines do not fail). Certify the results based on a probabilistic analysis of the results (using only the results of the active tasks). Extension with an audit mechanism that take into account the estimation (better than random sampling).
Literature on Collusion

Wong, An authentication protocol in Web-computing, IPDPS 2006

Propose a ring-based scheduling approach (duplication ratio of 2). Estimate the fraction and probability of collusion (given that non-colluding machines do not fail). Certify the results based on a probabilistic analysis of the results (using only the results of the active tasks). Extension with an audit mechanism that take into account the estimation (better than random sampling).

Taufer, Anderson et al., Homogeneous Redundancy, IPDPS 2005

Characterize a *divergent* application: large numerical differences in the results generated by different machines. Fuzzy comparison is not sufficient. Homogeneous redundancy: assign tasks to numerically equivalent machines (same software and platform characteristics).
Schedule Size

Inputs

Number of tasks: $n$
Number of processors: $m$
Duration of every possible executions (each tasks on each processors): $w_{ij}$
Longest duration: $W = \max_{ij} w_{ij}$
The size of the input is $O(nm \log(W))$. 
Inputs

Number of tasks: \( n \)
Number of processors: \( m \)
Duration of every possible executions (each tasks on each processors): \( w_{ij} \)
Longest duration: \( W = \max_{ij} w_{ij} \)
The size of the input is \( O(nm \log(W)) \).

Schedule providing Dates

Largest date: \( O(nW) \) (each task are scheduled on the slowest processor)
With duplication, each task may be scheduled on each processor: there is \( O(nm) \) dates.
A schedule requires \( O(nm \log(nW)) \) space for encoding the dates.
### Stochastic DAG evaluation

<table>
<thead>
<tr>
<th>Random variables type</th>
<th>Chain</th>
<th>Join</th>
<th>Series-parallel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discrete</td>
<td>regular domain</td>
<td>all</td>
<td>regular domain</td>
</tr>
<tr>
<td>Non-discrete</td>
<td>normal, gamma, Erlang</td>
<td>exponential, Weibull</td>
<td>none</td>
</tr>
</tbody>
</table>