Performability Modeling for Scheduling and Fault Tolerance Strategies for Scientific Workflows

Lavanya Ramakrishnan and Daniel A. Reed

Indiana University, Bloomington, IN
Microsoft Research, Redmond, WA

Proceedings of the HPDC 2008

Presenter: Sean
Outline

1. Introduction
2. Reliability Specification
3. Performability Analysis
4. Evaluation
Outline

1. Introduction
2. Reliability Specification
3. Performability Analysis
4. Evaluation
Outline

1. Introduction
2. Reliability Specification
3. Performability Analysis
4. Evaluation
Outline

1. Introduction
2. Reliability Specification
3. Performability Analysis
4. Evaluation
Lavanya Ramakrishnan

**Brief CV**  Ph.D. Student, Graduated, Now MCNC
- Ph.D. in Indiana University, 2008 (Expected), Advisor: Dennis Gannon.
- M.Sc. in Indiana University, 2002.
- B.Sc. in University of Mumbai, 2000.

**Research interests:** Distributed systems including grid computing, high performance computing and utility computing, workflow tools, resource management, monitoring and adaptation for performance and fault tolerance

**Publications:**

**Projects:**
- Linked Environments for Atmospheric Discovery(LEAD)
- Virtual Grid Application Development Software (VGrADS)
- Open Resource Control Architecture(ORCA)
Daniel A. Reed

**Brief CV**  Director of scalable computing and multicore at Microsoft Research (Since Nov. 2007)

- Ph.D. in Computer Science, Purdue University.
- M.Sc. in Computer Science, Purdue University.
- B.Sc. in Computer Science, University of Missouri, Rolla.

**Research interests:** Design of very high-speed computers, providing new computing capabilities for scholars in science, medicine, engineering and the humanities, tools and techniques for capturing and analyzing the performance of parallel systems, and collaborative virtual environments for real-time performance analysis.

Two great forces are reshaping computing: **multicore processors with unprecedented power and the explosive growth of software services hosted on megascale data centers**

**Professional Experience:**

- 2005 The North Carolina General Assembly appropriates $5.9M in state FY06 and $11.8M in FY07 and beyond to expand the Renaissance Computing Institute (RENCI)
- 2005 The President’s Information Technology Advisory Committee (PITAC) and its subcommittee on computational science, which he chaired, produced a report on the future of computational science, entitled “Computational Science: Ensuring America’s Competitiveness.”
- 2001, Reed led the effort to launch the National Science Foundation’s TeraGrid, the world’s largest, most comprehensive distributed cyberinfrastructure for open scientific research, and then served as TeraGrid chief architect through 2003.
Outline

1. Introduction
2. Reliability Specification
3. Performability Analysis
4. Evaluation
Core Concept: Performability

- **Performability**: a composite measure a system’s *performance* and its *dependability*

- **Performance**: the "quality of service (QOS), provided the system is correct"

- **Dependability**: an all-encompassing definition for reliability, availability, safety and security
Problem Statement

- Grid/Cloud computing need to be degradable
- Resource availability vary significantly: Hardware + Software
- Performance (QoS) fluctuation incurred by resource availability
- Degradable: a resource is not only in two states, "fully-operational" or "failed"
- How to be degradable
  - Resource provider: provide an assured level of service under a cost model
  - Software: provide an interface for user to express their performance and reliability requirement
  - Execution models: Characteristics of program execution need to be understood

Approach:
Using performability, present a qualitative model to capture and analyze the effect of resource reliability on application performance.
Detailed approach:

- Allow user to express the availability requirement: Based on existing Virtual Grids framework in VGrADS Project

Virtual Grid Execution System Design

- Understanding the applications’ reliability requirements: Three common programming models
Three Common Programming Models

(a) Master Workder
(b) Divide and Conquer
(c) SPMD

(a) (b) (c)
Description of vgDL: BNF grammar for Redline

Redline expression ::= Identifier‘=‘
Arithmetic_expr | Logic_expr | Predicate

Arithmetic_expr ::= A_operand [A_op A_operand]*
A_operand ::= Integer | Real
A_op ::= "+" | "-" | "*" | "/" | "^"

Logic_expr ::= L_operand [L_op L_operand]*
L_operand ::= Integer | Real | Boolean |
Description of vgDL: BNF for vgDL

\[ Vgrid ::= \text{Identifier} = \text{Rdl-expression} \ [ \text{at} \ \text{time/event} \] \]
\[ \text{Rdl-expression} ::= \text{Rdl-subexpression} \ | \ [ \text{"("} \ \text{Rdl-expression} \text{"\") \ op} \ \text{"("} \ \text{Rdl-expression} \text{"\")} \] \]
\[ \text{Rdl-subexpression} ::= \text{Associator-expression} \ | \ \text{Node-expression} \]
\[ \text{Associator-expression} ::= \text{Bag-of-expression} \ | \ \text{Cluster-of-expression} \]
\[ \text{Bag-of-expression} ::= \text{LooseBagof} \ "\langle" \ \text{Identifier} \ "\rangle" \ [\ "\text{MinNode} " : " \text{MaxNode} \] \ [ \ "\] \ [ \text{Number} \ [ \ "\text{su}" \ | \ "\text{sec}\] \ [\ ] \ ];" \ \text{Node-expression} \]
\[ \text{TightBagof} \ "\langle" \text{Identifier} \ "\rangle" \ [\ "\text{MinNode} " : " \text{MaxNode} \] \ [\ "\] \ [ \text{Number} \ [ \ "\text{su}" \ | \ "\text{sec}\] \ [\ ] \ ];" \ \text{Node-expression} \]
\[ \text{Identifier} ::= \text{String} \]
\[ \text{Min} ::= \text{Integer} \]
\[ \text{Max} ::= \text{Integer} \]
\[ \text{Node-expression} ::= \text{Identifier} \ "\=" \ \text{Node-constraint} \]
\[ \text{Node-constraint} ::= \{" \ \text{Attribute-constraint} \ | \ \text{Rdl-expression} \} \"\}" \ | \ \text{Rdl-expression} \]
\[ \text{Attribute-constraint} ::= \text{Redline expression for attribute and constraint} \ [\ \text{see Figure 3-2}\]
\[ \text{Cluster-of-expression} ::= \text{Clusterof} \ "\langle" \ \text{identifier} \ "\rangle" \ [\ "\text{MinNode} " : " \text{MaxNode} \ [ \ "\] \ [ \text{MinTime} \ "\:" \ "\text{MaxTime} \] \ ];" \ \text{Node-expression} \]
\[ \text{op} ::= \text{close} \ | \ \text{far} \ | \ \text{highBW} \ | \ \text{lowBW} \]
Example 1: mpiBLAST (vgDL)

\[
\text{mpiBLAST}1 = \text{MasterNode} = \{ \text{memory } 4\text{GB, disk } > 20\text{GB} \} \text{ highBW LooseBagOf } < \text{WorkerNode}> [4:32] ;\text{WorkerNode} = \{ \text{memory } \geq 4\text{GB} \}
\]

\[
\text{mpiBLAST}2 = \text{MasterNode} = \{ \text{memory } 4\text{GB, disk } > 20\text{GB} \} \text{ (goodReliability AND highBW) LooseBagOf } < \text{WorkerNode}> [4:32]; \text{WorkerNode} = \{ \text{memory } \geq 4\text{GB} \}
\]

\[
\text{mpiBLAST}3 = \text{HighReliabilityBag}<\text{MasterNode}> = \{ \text{memory } 4\text{GB, disk } > 20\text{GB} \} \text{ (goodReliability AND highBW) LooseBagOf } < \text{WorkerNode}> [4:32]; \text{WorkerNode} = \{ \text{memory } \geq 4\text{GB} \}; \text{MasterNode} = \{ \text{memory } 4\text{GB, disk } > 20\text{GB} \}
\]
Example 2: Weather Research and Forecast (WRF) Model

\[
\begin{align*}
\text{wrf1} &= \text{WRF Bag} = \text{Tight Bag Of} <\text{C Node}>[8:32]; \text{ C Node} = \{\text{memory} \geq 4 \text{GB}\} \\
\text{wrf2} &= \text{WRF Bag} = \text{High Reliability Bag} <\text{Many Nodes}>[1:1]; \text{ Many Nodes} = \text{Tight Bag Of} <\text{C Node}>[8:32]; \text{ C Node} = \{\text{memory} \geq 4 \text{GB}\}
\end{align*}
\]
Outline

1. Introduction
2. Reliability Specification
3. Performability Analysis
4. Evaluation
Reliability Specification for vgDL Extension

Quantitative Reliability Level: Excellent (90-100%), Good (80-89%), Satisfactory (70-79%), Fair (60-69%), Poor (0-59%)

- **Node**: HighReliabilityBag, GoodReliabilityBag, MediumReliabilityBag, LowReliabilityBag, PoorReliabilityBag

- **Link**: highReliability, goodReliability, mediumReliability, lowReliability, poorReliability
Example System

Stochastic processes are "mathematical models of systems which vary in time in a random manner". The problem is that although the model is required to represent the time-dependent behaviour of the system, time cannot be considered in the representation of the system. Markov process representation, a type of stochastic process, allows the consideration of time in a very controlled manner, thus overcoming the apparent obstacle. The Markov property is such that "given the present state, the past states have no influence on the future". A Markov process describes the structural changes in the system as faults and/or repairs occur. For the cases typically modelled, there must be bounds on the state space size. For example, a fault-tolerant computer system can only occupy so many different states...not an infinite number. So, in general Markov chains are used, which are "discrete parameter Markov processes whose state space is finite or countably infinite".

The Markov Reward Model

The most commonly used performability model today is the Markov Reward Model (MRM). To illustrate the modelling technique, applied to a cyclic case, a 3-CPU multi-processor system is used, that begins running in fully operational mode. Jobs arrive at the buffer and are stored until a processor (CPU) becomes available, then the job at the head of the buffer is sent to this CPU to be processed. In this manner jobs are shared equally between the 3 identical processors.

Model of the multi-processor system

There are some assumptions in our model to take note of. It is assumed that not more than one processor can fail at one time; there is no simultaneous failures of CPUs. This is described by the transition arrows (only one possible transition to & from a state). Another assumption is that the buffers are ultra-reliable, so buffer failure is not modelled for, although such a failure might result in a complete system breakdown. There are no limits on buffer capacity, this eliminates the possibility of job loss that would occur with a full buffer. It is also assumed that all our processors are identical, so that if for example one CPU fails, only a single state represents the three possible cases. This lumping method reduces the state space of the problem. More details on the State Space Explosion problem, and methods of dealing with it are mentioned later.

Any one processor may fail, thus causing the system to degrade. This degradable system is fault-tolerant because the work can still be shared out among the remaining operational one(s), should one (or two) CPUs fail. Thus the system has several different stages between fully functional and completely failed. The system can still do work in less than fully operational status, although at a reduced performance level. This provides the perfect opportunity for performability evaluation of the system.

The MRM is built up of two distinct models: the behaviour model and the reward structure. The behaviour model, funnily enough, describes the possible behaviour of the system. A degradable system, depending on the faults that occur, can be in different states at different times. If all the processors were running that would be one state, should one fail it would be another state, etc... The states, and the transition links between them representing by failures and repairs, make up the behaviour model.

Each different state, representing a different number of processors ‘up’, is capable of a different amount of work. In other words each has a certain performance level associated with it. The amount of performance achievable has a certain reward related to it. This reward rate quantifies the ability of the system to perform. If the system goes to a state with a higher reward, a higher performance level is reached. The set of these rewards, associated to the individual states, make up the reward structure.
Markov Reward Model

Together the behaviour model and reward model describe the MRM:

Regarding figure 2, you see that there are four states describing the system. These are:

- State 1: 3 processors up, 0 processors down
- State 2: 2 processors up, 1 processor down
- State 3: 1 processor up, 2 processors down
- State 4: 0 processors up, 3 processors down

The possible transitions between each state are described by the arrows joining them. The system will only spend a certain amount of time in each state, called the *holding time*. The holding times in each state are typically exponential distributed, therefore we can associate to each a probability of changing state, over time. These are the labels on the transition arrows, *Lambda* and *Mju*. *Lambda* refers to a state switch due to a *failure* caused by faults occurring in the system; *Mju* to one caused by *repairs* to the system. From these, and the state transition diagram, we can build up *Q*, the *Generator matrix* shown in figure 2. Each transition rate from state i to state j is denoted by the term *q*(i,j) in the matrix.

There are two possible ways of assigning rewards in the reward structure. The structure can associate reward rates with state occupancies or associate reward impulses with state transitions. These two techniques can be implemented in the model together or in mutual exclusion. In the latter case a fixed reward is associated with each transition, to be awarded at the moment of the transition's occurrence. It is, however, the former technique that is adopted in our example, whereby the reward is assigned per unit time (a rate). It can be interpreted as the rate at which reward is accumulated in each state: The longer the system remains in a state, the more reward it accumulates; the higher the reward rate, the more reward per unit time.

The rates described in figure 2 are summed up below. The highest reward rate (*R*0) is associated to state 1, and the lowest (*R*3) to state 4:

- *R*0: performance level = 3.0
- *R*1: performance level = 2.0
- *R*2: performance level = 1.0
- *R*3: performance level = 0.0

The Markov reward Model
Accumulative Reward Y(t)

The finite-state stochastic process Z(t) represents the evolution (path) of the system in time. It is the state of the system at any time t. This is shown in figure 3. In this case the path it describes goes from state 1->2->1->2->3->4->3->2->1. As can be seen from the differing lengths of the horizontal lines in Z(t), the holding time in each state is random.

X(t) defines the reward rate of the system at time t, as shown in figure 3. In this case the rate goes from 3->2->3->2->1->0->1->2->3. This follows directly from the state changes shown in the plot of Z(t).

Y(t) is the crucial variable, entirely dependent on the previous two graphs. It is the accumulated reward until time t, which is the area under the X(t) curve. The higher the reward rate in X(t), the steeper the slope of the Y(t) curve, which implies more work done per unit time. With this the performability of the system is determined. By interpreting rewards as performance levels, the distribution of accumulated reward characterizes systems that evolve through states with different performance levels (degradable systems).

J.F.Meyer, from whom the concept of performability evaluation originated, formally described the performability of a system as "the probability that the system reaches an accomplishment level y over a utilization interval (0,t)". Looking at the graph of accumulated reward Y(t) in figure 3, y can be interpreted as a value on the y-axis. We see that a certain time t' is needed before a certain reward level (height of the curve) is reached.
The Probability Distribution Function of $Y(t)$

Informally, performability can be defined as the probability that the system does a certain amount of useful work over a mission time $t$. So the fundamental question about any system, "What is the probability of completing a given amount of useful work within a specified time interval?", is answered by simply taking the complement of the above formula:

$$y'(x,t) = \text{Prob}[Y(t) > x]$$

The solution to the performability model is found by evaluating the PDF of accumulated reward $y(x,t)$.

Probability Density Functions (PDFs) Illuminated

The PDF in figure 4 is a fairly average system probability distribution. To the unindoctrinated, it will not be particularly meaningful. These further figures will clarify the idea slightly.

A PDF graph literally shows how the probabilities of accomplishing the work are distributed. The PDF is the integral of the area under the pdf (probability density function), so is a direct translation of it. The higher the density, the steeper the slope of the PDF.

It stands to reason then that a system that operates for the majority of time at fully operational status will have a higher density of probabilities nearer to the maximum accomplishment level. (R0 $\times$ t...the furthest right of the PDF, which represents the greatest possible accumulated reward).
Definition of Performability

- **Performability** is defined as “the probability that a system reaches an accomplished level $y$ over a utilization interval $(0,t)$.”

  $y(x,t) = \text{Prob}[Y(t) <= x]$  
  $y'(x,t) = \text{Prob}[Y(t) > x]$

Markov chain for the resource performance and reliability states
Resource State Reliability Model

- MTBF = MTTF + MTTR

- $\lambda = MTTF^{-1}$

- $\mu = MTTR^{-1}$

- The steady state probability of occupancy in each state: $\pi_n = \rho^n \pi_0$, $\pi_0 = 1 - \rho$, $\rho = \lambda/\mu$, failure-to-repair ratio

- Normally $\rho < 1$, otherwise, the system is towards complete failure
Performability Modeling

- $T$: the running time on high available resource

- Running time in other states: $T + n_i x$, $i = 1,2,3,4$ (Fail is not counted)

- Reward Rate: inverse of the running time $1/(T + n_i x)$

- Measure performability as the accumulated reward rate over a specificized time interval: $E[Z(t)] = \sum r_i \pi_i(t)$
### Performability Example

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Machines</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>Application running time $T$</td>
<td>30 min</td>
<td>30 min</td>
<td>25 min</td>
<td>15 min</td>
</tr>
<tr>
<td>Failure-to-repair rate $\rho$</td>
<td>0.1</td>
<td>0.4</td>
<td>0.4</td>
<td>0.6</td>
</tr>
<tr>
<td>Perform. $x=2$</td>
<td>0.033</td>
<td>0.032</td>
<td>0.038</td>
<td><strong>0.055</strong></td>
</tr>
<tr>
<td>Perform. $x=100$</td>
<td><strong>0.031</strong></td>
<td>0.224</td>
<td>0.027</td>
<td>0.029</td>
</tr>
</tbody>
</table>

$$n_1 - n_4 = 1, 2, 3, 4$$
Performability of Different Programming Models

- Master-worker application: \( E_{(M-W)} = \text{Min}(E_{\text{Master}}, E_{\text{Worker}}, E_{\text{Network}}) \)
  when \( T_{\text{Master}} >> T_{\text{Worker}} \) and \( T_{\text{Master}} >> T_{\text{Network}} \)

- Divide and Conquer: performability of the root (Tree root runs longer)

- SPMD: \( E_{SPMD} = \text{Min}(E_{\text{systemcomponents}}) \)
Workflow Panning for Performability

- Workflow scheduling can base on the projected application running time:
  \[ T_{\text{projected}} = \frac{1}{E[Z]}(\text{computation}) \]

- Follow the performability modeling procedure to achieve the network performability

- Based on the computation performability and network performability, using traditional scheduling algorithm to the workflow
Fault Tolerance Strategies

- Two common strategies: replication (good performance and reliability, but high cost) and checkpoint-restart (good reliability but low performance)
- Cost of replication: \( C_R = T_{projected} \times n \), \( n \) is the number of replica
- Cost of checkpoint-restart: \( C_{CR} = C_{checkpoint} + C_{restart-on-failure} \),
  \( C_{checkpoint} = C_{per-checkpoint} \times T_{projected} / T_{interval} \), \( T_{interval} \): optimal checkpoint interval to meet the performability level
- if \( C_R << C_{CR} \), replication is preferred
- Two cases:
  - Checkpoint-restart: \( T_{projected-FT} = T_{projected} + C_{checkpoint} + C_{restart-on-failure} \)
  - Replication: \( T_{projected-FT} = T_{projected} \)
Outline

1. Introduction
2. Reliability Specification
3. Performability Analysis
4. Evaluation
## Application Descriptions

<table>
<thead>
<tr>
<th>Name</th>
<th>Application Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>arps2wrf</td>
<td>Generates initial and lateral boundary conditions for WRF.</td>
</tr>
<tr>
<td>wrfstatic</td>
<td>Processes static data sets such as terrain, vegetation, soil texture, etc that serves as input for a meteorological model WRF.</td>
</tr>
<tr>
<td>adcirc</td>
<td>Finite element hydrodynamic model for storm surge modeling (run on 64 processors)</td>
</tr>
<tr>
<td>wrf</td>
<td>Mesoscale numerical weather prediction system (run on 128 processors)</td>
</tr>
</tbody>
</table>
### Table 2: Application Descriptions

<table>
<thead>
<tr>
<th>Name</th>
<th>Process description</th>
</tr>
</thead>
<tbody>
<tr>
<td>arps2wrf</td>
<td>Generates initial and lateral boundary conditions for WRF.</td>
</tr>
<tr>
<td>wrfstatic</td>
<td>Generates input for a meteorological model WRF.</td>
</tr>
<tr>
<td>ADCIRC</td>
<td>A storm surge modeling application that runs on 64 processors.</td>
</tr>
</tbody>
</table>

### Evaluation

#### 5.1 Application Performance Variability

- Performance data is collected on production machines such as TeraGrid.
- The cumulative failure data history is used to simulate system behavior.
- Simulation is performed to calculate failure and repair rates.
- The reliability data is collected from about 22 high-performance computing systems at Los Alamos National Laboratory.

#### 5.2 Workflow Scheduling Simulation

- The real-time monitoring testbed collects and publishes data on system status and service availability.
- Workflow scheduling simulation results validate the performability model.
- Workflow scheduling needs to account for system variability.
- The experimental data is provided as input to the model.
- The approach is orthogonal to the choice of workflow scheduling heuristics.

#### 5.2.2 Reliability data

- Reliability data for the three systems (2, 5, 9) is collected in this study.
- The data is used to calculate failure and repair rates.
- Repair times vary over the lifetime of the systems.

#### 5.2.3 Workflow Algorithms

- Two heuristics are applied: Min-min and Max-min.
- The exact algorithm can be used with other algorithms.

### Figure

- **EVALUATION**
- **Wether Forecasting DAG**

The diagram illustrates the workflow and dependencies between different steps in a weather forecasting process, including ARPS-WRF and WRF, with associated run times and transfer sizes.
Data Collection

- Application selected: WRF

- Two TeraGrid NCSA clusters observed over two weeks
  - Mercury: IBM IA-64 cluster, mixture of 1.3GHz and 1.5GHz Intel Itanium 2 CPU
  - Tungsten: collection of Dell PowerEdge 1750 servers, Intel Xeon 3.2GHz CPU

- Data collected from 132 runs on Mercury and 77 runs in Tungsten
Application (WRF) Runningtime over TeraGrid Machines

(a) Mercury

(b) Tungsten
Application (WRF) Runningtime over TeraGrid Machines

(a) Mercury
- On Mercury most take 103 minutes, 27% out of this range
- On Tungsten most take from 150 to 170 minutes, 20 minutes variation (significant)
Experiment Setup

- **Traces Selection:**
  - Not TeraGrid: Availability Prediction Service, 
  - Not INCA RealTime Monitoring Suite 
  - Still insufficient
  - Performance data: 1. application run times in different reliability states from a normal distribution with mean from real observations on TeraGrid machines, 2. bandwidth data from long-tailed Pareto distribution
  - Reliability data: LANL failure trace, select system 2 (6152 cpus), 5 (512 cpus), 9 (512 cpus)

- **Two heuristic workflow scheduling algorithms:** Min-min, Max-min

- **Two approaches**
  - **PERF:** First consider app performance time on each resource in the high state, then generate a schedule
  - **HYBRID:** Calculate the protected running time, then apply heuristics
Failure to Repair Rates

- In Exp 1, all systems using first 18,000 hours trace is used (no prior accumulated resource history)
- In Exp 2, system 2 using 20,000-30,000 hours trace (only system 2 has prior accumulated failure data history)
In Exp 1, all systems using first 18,000 hours trace is used (no prior accumulated resource history)

In Exp 2, system 2 using 20,000-30,000 hours trace (only system 2 has prior accumulated failure data history)

Failure-to-repair rates for system 2 in Exp 2 are significantly higher than the other two systems
Comparison of Workflow Tasks

Figure 6: Comparison of workflow tasks scheduled on these numbers. The HYBRID approach has the rates (referred to as HYBRID approach) and the heuristic is performability analysis (section 4.4.1) using application perlate the projected running time for the application based on a subset of machines under consideration. Next, we calculate the number of workflow tasks as expected.

As the failure-to-repair rates increase, the performability ratio compared to the other two systems significantly higher than the other two resources and we see a drop in the number of components that are scheduled on the machine. Figure 7(a) shows the ratio of makespan from system 2, the corresponding failure-to-repair rates for system 2 are the single-node and MPI jobs. This simplifying assumption is appropriate since all systems might not exhibit all fail-

Figure 5: Failure to repair rates over time for arps2wrf and wrfstatic on both clusters. Both applications were run individually.
Comparison of Workflow Tasks

Due to higher failure-to-repair rates, components running system 2 in Exp 2 is dropped.
Figure 7: Ratio of (a) makespan (b) performability

(a) Ratio of makespan
- Figure 7(a): High failure-to-repair rate, HYBRID produces longer makespan. Exp 2 makespan is higher due to high failure-to-repair ratio (best performance machines are not selected due to low reliability)

(b) Ratio of performability
- Figure 7(b): The performability of HYBRID is significantly higher than PERF
- Difference between two heuristic approaches for both makespan and performability is small
Figure 7(a): High failure-to-repair rate, HYBRID produces longer makespan. Exp 2 makespan is higher due to high failure-to-repair ratio (best performance machines are not selected due to low reliability).

Figure 7(b): The performability of HYBRID is significantly higher than PERF.

- Difference between two heuristic approaches for both makespan and performability.
- Emphasizing the importance of performability as a metric in workflow scheduling improves performability with minimal effect on makespans.
App Running Time for Meteorological and Ocean Modeling

(a) Short running

(b) Long running

EVALUATION
Different failures modes and underlying hardware characteristics have an impact on applications.
Expected Steady-State Reward Rate with Performance Degradation Factors

\[ \rho(R \text{atio of failure rate to repair rate}) \]

(a) \( x = 2 \)

(b) \( x = 70 \)
Expected Steady-State Reward Rate with Performance Degradation Factors

If performance difference between high and other states is significant, the performability decreases linearly (intuitive)
Expected Steady-State Reward Rate with Different $n_i$

![Graph showing expected steady-state reward rate with different $n_i$ values.](image)

- $n_1=1, n_2=1, n_3=1, n_4=1$
- $n_1=1, n_2=2, n_3=3, n_4=4$
- $n_1=1, n_2=3, n_3=9, n_4=27$
- $n_1=100, n_2=110, n_3=120, n_4=130$

Running time = 60 minutes, $x=30$
Expected Steady-State Reward Rate with Different $n_i$

Running time = 60 minutes, $x = 30$
Cost Analysis

\[ C_{\text{per-checkpoint}} = 1, \quad C_{\text{restart-on-failure}} = 3 \]

- \( C_R < C_{CR} \)
- Smaller application: replication
- Checkpoint frequently, cost increase
- If the app is very critical and the cost of replication and checkpoint-restart is immaterial, both might also be used simultaneously
Cost Analysis

\[ C_{\text{per-checkpoint}} = 1, C_{\text{restart-on-failure}} = 3 \]

- \( C_R < C_{CR} \)
- Smaller application: replication
- Checkpoint frequently, cost increase
- If the app is very critical and the cost of replication and checkpoint-restart is immaterial, both might also be used simultaneously
Thank You!

Thank you and any questions?

Personal Information

Zhengqiang Liang
Ph.D. Candidate of
Dept. of Computer Science
Wayne State University
sean@wayne.edu

Mobile and Internet SysTems Laboratory

Homepage URL

http://www.cs.wayne.edu/~sean