LA-UR- 11-02874

Approved for public release;
distribution is unlimited.

Title:	Modeling High-Performance Computing Application-Failure and Hardware-Failure Data
Author(s):	Todd Graves Sarah Michalak Lori Pritchett-Sheats
Intended for:	Hierarchical Models and Markov Chain Monte Carlo Conference in Honor of AFM Smith

Form 836 (7/06)

Modeling High-Performance Computing Application-Failure and Hardware-Failure Data

Todd Graves, Sarah Michalak, and Lori Pritchett-Sheats

Los Alamos National Laboratory

Application-failure data and hardware-failure data can provide contrasting information about high-performance computing (HPC) platform health. For example, hardware-failure data may indicate that the system is in a typical state, while application users may be experiencing an unusually high number of failures. Using application-failure and hardware-failure data jointly to understand HPC platform health should provide a more accurate description of system health. This poster presents preliminary (non-joint) models for application-failure data and for hardware-failure data and considers how the two might be combined.

Modeling High-Performance Computing Application-Failure and Hardware-Failure Data

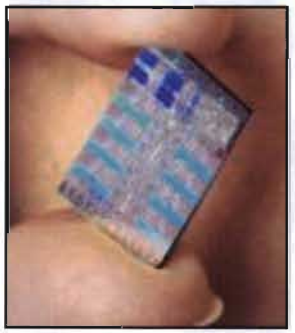
Todd Graves

Sarah Michalak

Lori Pritchett-Sheats

High-Performance Computing Platforms

- **HPC platforms are used for large scientific calculations:**
  - Capability: large calculations (4k-20k processes) that run for weeks to months
  - Capacity: small/medium calculations (10-40k processes) that can run for days
- **Some HPC platforms are at the cutting edge in terms of scale/technology**



The Problem

- **Two perceptions of HPC platform health:**
  - Users run applications that may not complete correctly
    - Job fails to launch, I/O or FS issue, node failure, …
  - HPC Division collects hardware failure data
    - Intra-node (DIMM, CPU, …), switches, …
- **User perception is not always consistent with hardware failure data**
- **Can both data sources be used to:**
  - Better describe platform health
  - Maximize application throughput
    - Which job mixes lead to the best throughput

Pilot Study: Collecting Application Data

- **Application monitoring test (July; Sept/Oct 2009)**
  - 39 calculations requiring 1142 individual jobs
  - Job completes successfully if it does not suffer an unscheduled interrupt
- **Run on Linux cluster with 255 compute nodes in each of 13 segments**
- **RAGE: Eulerian adaptive mesh code**
- **AppTK library to monitor job progress, detect hangs**
  - Application monitoring software captures exit status for most jobs
- **"Hello World" MPI program before launching binary**
- **Run two calculations using RAGE:**
  - Problem 1: 8, 16, 32, and 64 nodes
  - Problem 2: 20 and 40 nodes
- **Jobs submitted during off-peak hours: nights and weekends**

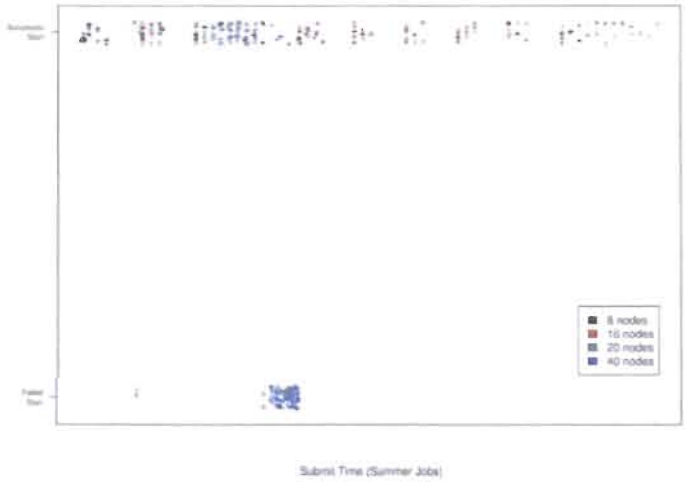
Pilot Study: High-level Results

- **Many job failures caused by full file system – unable to write to it**
  - Continue to submit jobs when file system full – many errors
  - Full file system not in hardware failure data collected then
- **Hung jobs failed on MPI "Hello World" program**
  - Typically could not be traced to error messages
  - Reported hardware problems did not coincide with the hung jobs
- **Certain other FS errors and some node re-boots not reflected in hardware failure data**

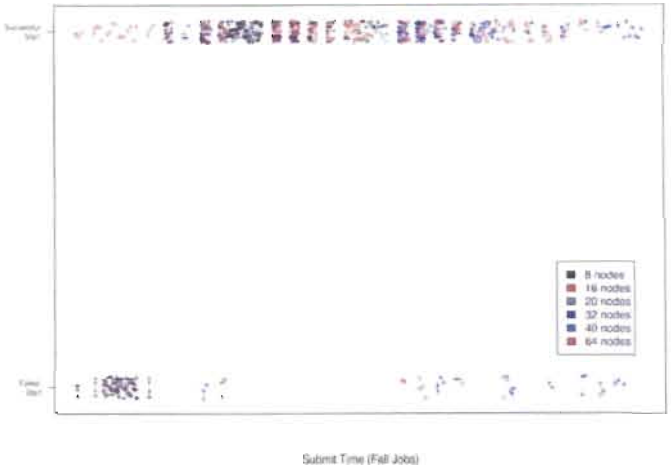
Application and hardware failure data can be complementary

Pilot Study: Data Analysis Results I

- **Pr(successful job start)**
  - Sine and cosine of start time important predictors (4 AM good start time)
  - Tuesdays, Wednesdays, Fridays, Saturdays better days to start jobs
  - Sunday July 19, 2009 particularly bad (85/109 fail b/c file system full)
  - System tends to experience bad times when many jobs fail to start



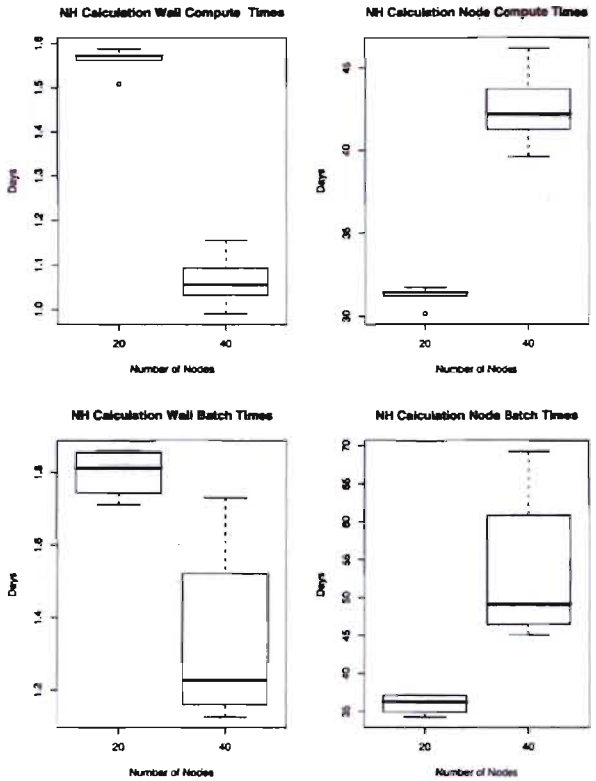
Submit Time (Summer Jobs)



Submit Time (Fall Jobs)

Pilot Study: Data Analysis Results II

- **Time to completion (Problem 2)**



To decrease wall time to completion (complete a job quickly), use 40 nodes.

To decrease node time to completion (minimize resource use), use 20 nodes.

Implications for jobs scheduling depending on the objective.

Hardware Failure Data

- **Roughly one year of data from the same cluster**
- **Focus on single-node failures**
  - Start time, resolution time, node affected, reason category, type of hardware problem (if applicable), scheduled indicator
  - Further parsing: swapped nodes, user interrupt
- **Analyses need to reflect:**
  - Nodes that never failed
  - Scheduled outages
- **List of uptimes for each node**
  - End in failure, scheduled interrupt, censored (final uptime period)
  - Downtimes implicit in this list

Hardware Failure Analysis Results I

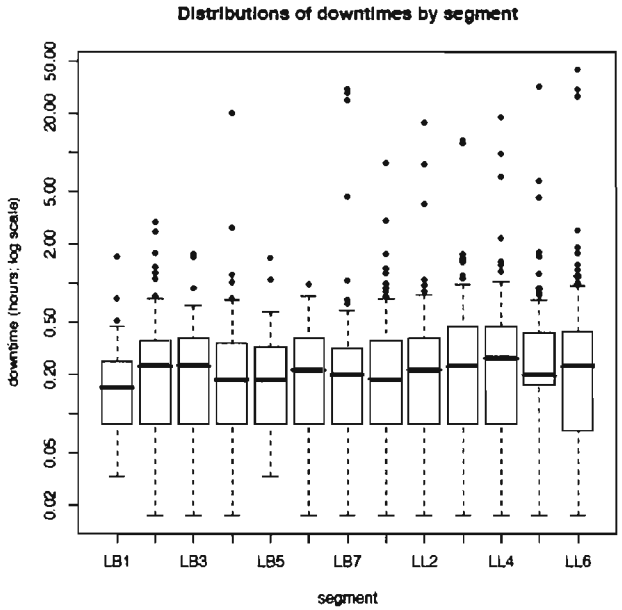
- **Exponential-Gamma hierarchical model for single-node failures within each segment**

Table 10: Results for single node failure time analyses.

Segment	Failures	MTTF	.01	.05	.10	$P(>10\text{yr})$	.01	.05	.1	$P(>10\text{yr})$
LB1	45	69	8.6	18	26	.56	6.0	15	26	.67
LB2	92	33	2.7	6.2	11	.63	2.0	6.3	13	.69
LB3	81	38	3.8	7.8	13	.56	2.7	7.5	15	.65
LB4	82	38	3.9	8.1	13	.56	2.7	7.5	14	.64
LB5	64	48	7.3	13	19	.43	4.8	11	18	.56
LB6	59	53	6.1	12	18	.55	4.3	11	19	.64
LB7	78	41	15	22	27	.05	7.1	13	17	.34
LL1	90	34	11	16	20	.07	5.3	9.8	14	.32
LL2	78	39	15	22	25	.04	6.8	12	16	.32
LL3	112	28	6.6	10	13	.14	3.7	7.2	11	.33
LL4	111	28	8.8	13	16	.05	4.4	8	11	.27
LL5	115	27	5.1	8.3	11	.24	3.0	6.6	10	.40
LL6	199	15	2.5	4.3	5.9	.23	1.6	3.5	5.8	.36

Hardware Failure Analysis Results II

- **Segments differed in terms of the % of node failures that could cause a user interrupt**
  - Min = 68%; max = 87%
- **Single-node downtimes similar in 13 segments**



Conclusions

- **Both application and hardware data are needed to understand the application-user's experience**
- **Could use predictive distributions for queue times, pr(successful start), times to hardware failures, node(s) affected, and downtimes to describe/predict user experience and optimize application throughput**
  - Parameters in these models could be updated as new data become available