# Adaptive Fault Management for High Performance Computing

Zhiling Lan

Illinois Institute of Technology (Dept. of CS)



# Reliability Concerns

- Systems are getting bigger
  - 1024-4096 processors is today's "medium" size (73.4% on the recent TOP500 List)
  - O(10,000)~ O(100,000) processor systems are being designed/deployed
- Even highly reliable HW can become an issue at scale
  - 1 node fails every 10,000 hours
  - 6,000 nodes fail every 1.6 hours
  - 64,000 nodes fail every 5 minutes


☞ Needs for fault management!

Losing the entire job due to one node's failure is costly in time and CPU cycles!

# The Big Picture

- Checkpoint/restart is widely used for fault tolerance
  - Simple
  - IO intensive, may trigger a cycle of deterioration
  - Reactively handle failures through rollbacks
- Newly emerging proactive methods
  - Good at preventing failures and avoiding rollbacks
  - But, relies on accurate prediction of failure

**FENCE: Fault awareness ENabled Computing Environment**

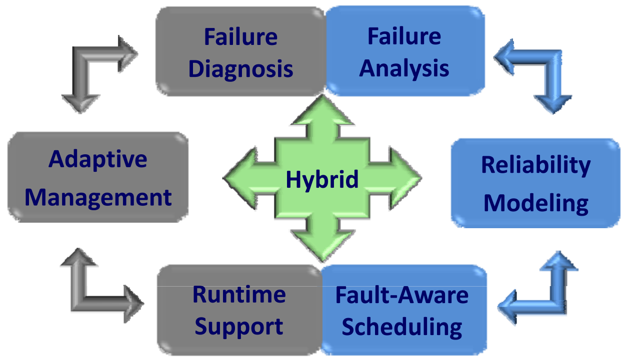
- **A "fence" to protect system and appl. from severe failure impact**
- **Exploit the synergy between various methods to advance fault management**
- **In collaboration with Xian-He Sun at IIT**



# FENCE Overview

- Adopt a hybrid approach:
  - *Long-term* reliability modeling and scheduling enables intelligent mapping of applications to resources
  - *Short-term* fault resilience support allows applications to avoid imminent failures
- Explore runtime adaptation:
  - *Proactive actions* prevent applications from anticipated failures
  - *Reactive actions* minimize the impact of unforeseeable failures
- Address fundamental issues
  - Failure analysis & diagnosis
  - Adaptive management
  - Runtime support
  - Reliability modeling & scheduling

# Key Components

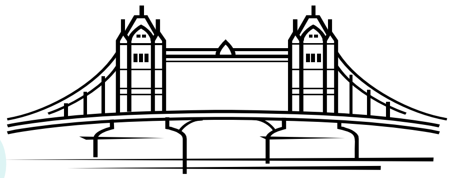





# Outline

- Overview of FENCE Project
- This talk will focus on the short-term support
  - Failure analysis and diagnosis
  - Adaptive management
  - Runtime support

## Failure Analysis and Diagnosis



**Health/perf monitoring:**

- **Hardware sensors**
- **System monitoring tools**
- **Error checking services,**

**e.g. Blue Gene series and Cray XT series**

**Fault tolerance technologies**

- **Checkpointing**
- **Process/object migration**
- **Other resilience supports**



# Failure Analysis & Diagnosis

- Goal:
  - To DISCOVER *failure patterns and trends* from data
  - To PROVIDE timely *alerts* regarding *"when and where"* failures are likely to occur
- Challenge:
  - Potentially overwhelming amount of information collected by error checking and monitoring tools
  - Fault patterns and root causes are often buried like needles in a haystack!

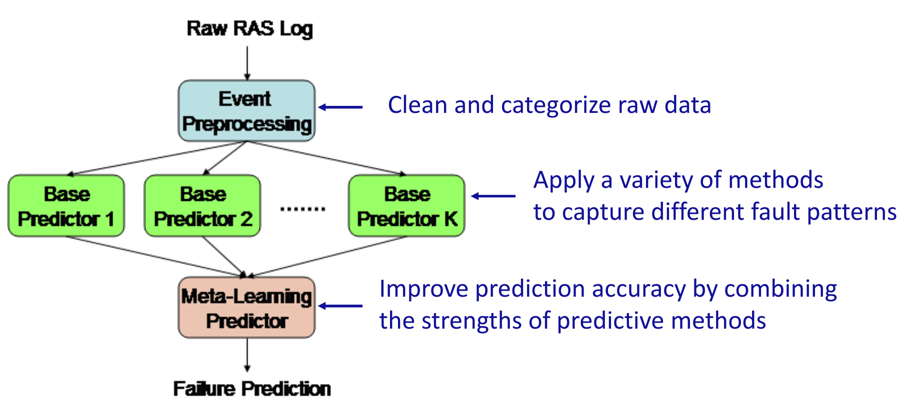
- How to capture a variety of fault patterns?
- How to achieve better diagnosis ?

# Failure Analysis & Diagnosis

- Our approach:
  - Integrate multiple data sources: RAS log, perf data, sensor readings, …
  - Coordinate predictive methods: statistical learning, data mining, pattern recognition, ensemble learning (meta-learning)
- The "when" question
  - Ensemble learning based prediction
- The "where" question
  - PCA (Principal component analysis) based localization



# Ensemble Learning Based Prediction



# Case Study: Blue Gene/L

- Blue Gene/L systems at ANL and SDSC
  - Each had 1,024 dual-core PowerPC440-based compute nodes and 32-128 IO nodes
- RAS (Reliability, Availability, and Serviceability) logs collected by the CMCS service

| | SDSC BGL | ANL BGL |
|---|---|---|
| Start Date | 12/6/04 | 1/21/05 |
| End Date | 2/21/06 | 4/28/06 |
| No. of Records | 428,953 | 4,172,359 |
| Log Size | 540MB | 5GB |



# BGL RAS Log

- The error checking service called CMCS (Cluster Monitoring and Control System)
  - Each record consists of a number of attributes
    - The SEVERITY attribute: INFO, WARNING, SEVERE, ERROR, FATAL, FAILURE
  - Our primary focus is to predict *fatal events (FATAL and FAILURE events)*
    - Removing "fake" fatal events is one of our on-going research

```
28543 RAS KERNEL FATAL 2005-02-01-11.12.18.562314 442 mikesyourdaddy   rts panic! - stopping execution
20300 RAS KERNEL FATAL 2004-12-28-12.12.49.832518  51 SDSC_FULL_128    machine check interrupt (bit=0x1d): L2 dcache unit data parity error
20301 RAS KERNEL FATAL 2004-12-28-12.12.49.909564  51 SDSC_FULL_128    instruction address: 0x00009b10
20302 RAS KERNEL FATAL 2004-12-28-12.12.49.979951  51 SDSC_FULL_128    machine check status register: 0x91800000
20303 RAS KERNEL FATAL 2004-12-28-12.12.50.053836  51 SDSC_FULL_128      summary...........................1
20304 RAS KERNEL FATAL 2004-12-28-12.12.50.117789  51 SDSC_FULL_128      instruction plb error.............0
20305 RAS KERNEL FATAL 2004-12-28-12.12.50.179673  51 SDSC_FULL_128      data read plb error...............0
20306 RAS KERNEL FATAL 2004-12-28-12.12.50.244230  51 SDSC_FULL_128      data write plb error..............1
20307 RAS KERNEL FATAL 2004-12-28-12.12.50.309678  51 SDSC_FULL_128      tlb error.........................0
20308 RAS KERNEL FATAL 2004-12-28-12.12.50.372156  51 SDSC_FULL_128      i-cache parity error..............0
20309 RAS KERNEL FATAL 2004-12-28-12.12.50.438323  51 SDSC_FULL_128      d-cache search parity error.......0
20310 RAS KERNEL FATAL 2004-12-28-12.12.50.502715  51 SDSC_FULL_128      d-cache flush parity error........1
```

# Event Preprocessing

- Step 1: Hierarchical event categorization
  - Based on LOCATION, FACILITY and ENTRY DATA

| Main Category | Subcategories | Examples |
|---|---|---|
| Application | 12 | loadProgramFailure, loginFailure, nodemapCreateFailure,… |
| Iostream | 8 | socketReadFailure, streamReadFailure,… |
| Kernel | 20 | alignmentFailure, dataAddressFailure, instructionAddressFailure, … |
| Memory | 22 | cachePrefetchFailure, dataReadFailure, dataStoreFailure, parityFailure,… |
| Midplane | 6 | linkcardFailure, ciodSignalFailure, midplaneServiceWarning,… |
| Network | 11 | ethernetFailure, rtsFailure, torusFailure, torusConnectionErrorInfo,… |
| NodeCard | 10 | nodecardDiscoveryError, nodecardAssemblyWarning,… |
| Other | 12 | BGLMasterRestartInfo, CMCScontrolInfo, linkcardServiceWarning,… |



# Event Preprocessing

- Step 2: temporal compression at a single location
  - To coalesce events from the same location with the same JOB_ID and LOCATION, if reported within time duration of 300 seconds

- Step 3: spatial compression across multiple locations
  - To remove entries close to each other within time duration of 300 seconds, with the same ENTRY_DATA and JOB_ID

Y. Liang, Y. Zhang, A. Sivasubramanium, R. Sahoo, J Moreira, M. Gupta, "Filtering Failure Logs for a BlueGene/L Prototype", *Proc. of DSN,* 2005.

## Event Preprocessing

| Main Category | ANL | SDSC |
|---|---|---|
| Application | 762 | 587 |
| Iostream | 1173 | 905 |
| Kernel | 224 | 182 |
| Memory | 52 | 25 |
| Midplane | 102 | 97 |
| Network | 482 | 366 |
| Node Card | 20 | 17 |
| Other | 8 | 3 |
| TOTAL | 2823 | 2182 |

**Distribution of Compressed Fatal Events**



## Base Prediction

- To capture a variety of fault patterns
- Apply two base predictive methods
  - statistical method and rule-based method
- Evaluation method:
  - Ten-fold cross validation
  - *Precision:* $\frac{T_p}{T_p + F_p}$
  - *Recall:* $\frac{T_p}{T_p + F_n}$

# Statistical Method

- Discover *probabilistic characteristics among fatal events* for failure prediction
- How often and with what probability will the occurrence of one failure influence subsequent failures
  - Step 1: on the learning set, obtain and verify the statistical pattern of failures from the training data
  - Step 2: on the testing set, produce a warning if the pattern is observed in a fixed window (i.e. 5 min to 1 hour)

| Log Name | Precision | Recall |
|---|---|---|
| ANL | 0.5157 | 0.4872 |
| SDSC | 0.2837 | 0.3117 |



# Statistical Method

- Issues with statistical method
  - Precision is too low.
    - Reason: the number of failures which do not have subsequent failures is substantial
  - Temporal correlation mainly exists for IO and network failures

# Rule Base Method

- Examine *causal correlations between non-fatal and fatal events* for failure prediction
- Rules in the form (*X =>Y)*, where *X* and *Y* are subsets of events (i.e. association rules)
  - If X occurs, then it is *likely* that Y will occur
- Two measures of rule interestingness:
  - *Support:* percentage of all the transactions under analysis that contain both *X* and *Y*
  - *Confidence*: percentage of all the transactions containing *X* also contains *Y*
  - We set the minimal value for *support* as of *0.04* and *confidence* of *0.2*



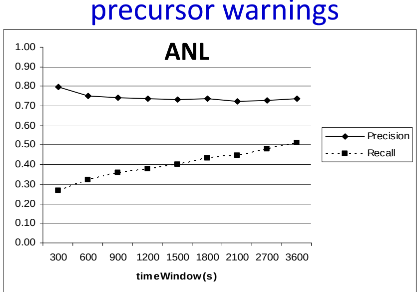
# Association Rules

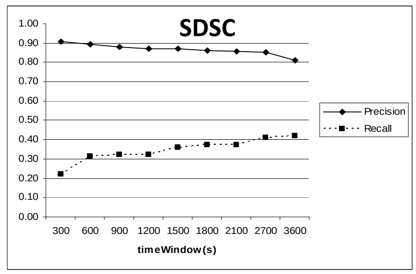
*Step 1:* on the learning set, for each fatal event, identify the set of non-fatal events frequently preceding it within a fixed time window (i.e. rule generation window);
*Step 2:* apply the standard association rule algorithm to build rule models that are above the minimum user-defined support;
*Step 3*: Combine rules as we focus on predicting whether there is an imminent failure;
*Step 4*: sort the generated rules in descending order of their confidence values;
*Step 5*: evaluate rules generated with different rule generation windows, and select the window size that can best capture a variety of patterns;
*Step 6*: on the testing set, use the rules generated to produce a warning if an association rule is observed within a prediction window.

```
nodeMapFileError ==> nodeMapCreateFailure: 1
nodeMapError ==> nodeMapCreateFailure: 0.947368
controlNetworkMMCSError ==> nodeConnectionFailure: 0.708333
ddrErrorCorrectionInfo maskInfo ==> socketReadFailure: 0.697674
ciodRestartInfo midplaneStartInfo controlNetworkInfo ==> rtsLinkFailure: 0.696629
ciodRestartInfo midplaneStartInfo ==> rtsLinkFailure: 0.688889
nodecardVPDMismatch nodecardAssemblySevereDiscovery nodecardFunctioanlityWarning ==> linkcardFailure: 0.636364
nodecardVPDMismatch nodecardFunctioanlityWarning midplaneLinkcardRestartWarning ==> linkcardFailure: 0.6
coredumpCreated ==> loadProgramFailure: 0.583333
midplaneStartInfo controlNetworkInfo BGLMasterRestartInfo ==> cacheFailure: 0.555556
nodecardDiscoveryError nodecardFunctioanlityWarning endServiceWarning midplaneLinkcardRestartWarning ==> linkcardFailure: 0.545455
```

# Rule Based Method

- Issues with rule based method:
  - The recall value is lower than 0.55, which is caused by the fact that a number of failures (31%-66% in ANL log and 47%-75% in SDSC log) do not have any precursor non-fatal events
  - Limited by the proportion of fatal events without any precursor warnings



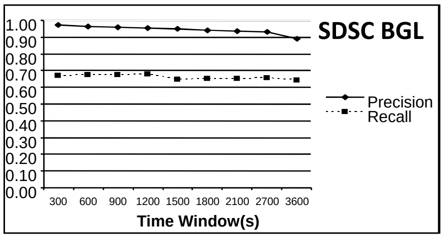


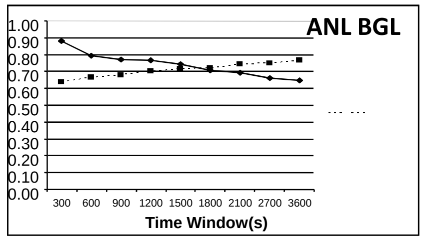

# Meta-Learning Prediction

- To boost prediction accuracy
  - By applying ensemble learning techniques

- A coverage based meta-learner
  - If there are *nonfatal* events, apply the rule based method for the discovery of fault patterns and produce a warning in case of matching rules
  - If *no nonfatal* event is observed, examine the occurrence of fatal events and apply the statistical based method for failure prediction

# Prediction Results




- Captures 65+% of failures, with the false alarm rate less than 35%
- The pattern generation process varies from 35 seconds to 167 seconds; and the matching process is trivial.

ILLINOIS INSTITUTE OF TECHNOLOGY



# Discussion

- Capable of detecting what is likely to occur in a near future (5 min - 1 hr)
  - The "when" question
- While it is useful by detecting when the system functions abnormally, it is equally important to find out which part of the system is the resource of the problem (aka *root cause localization*)

ILLINOIS INSTITUTE OF TECHNOLOGY

# Two Observations

1. Nodes performing comparable activities exhibit similar behaviors
   – HPC clusters often have such a property, e.g. parameter sweep, cluster management tools, web servers, …

2. In general, the majority is functioning normally since faults are rare events

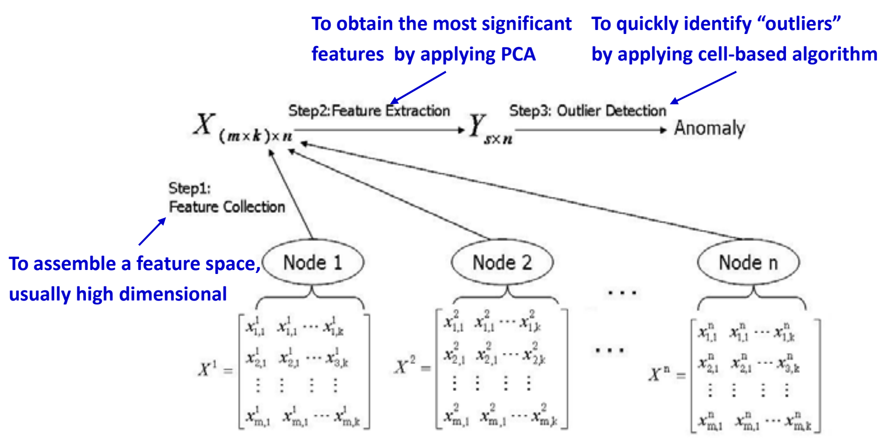


# Localization Method

1. Feature collection to assemble a feature space (usually high dimensionality) for the system
2. Feature extraction to obtain the most significant features out of the original feature space via PCA (principal component analysis)
3. Outlier detection to quickly identify the nodes that are "far away" from the majority

- Three interrelated steps, with a linear complexity
- A reduced feature space after PCA, e.g. ~97% reduction

# Localization Method






# Feature Collection

- Currently we use four system calls (*vmstat, mpstat, iostat & netstat*) to collect 12 features

| Feature | Description |
|---|---|
| CPU_System | Percent CPU in kernel |
| CPU_User | Percent CPU in user |
| CPU_Wait | Percent CPU blocked for I/O |
| Memory_Free | Amount of free memory (KB) |
| Memory_Swapped | Amount of virtual memory used (KB) |
| Page_In | Number of pages in (KB/s) |
| Page_Out | Number of pages out (KB/s) |
| IO_Write | Device writes per second |
| IO_Read | Device reads per second |
| Contect_Switch | Number of context switches per second |
| Packet_In | Number of packets into the network per second |
| Packet_Out | Number of packets out of the network per second |

# Feature Collection

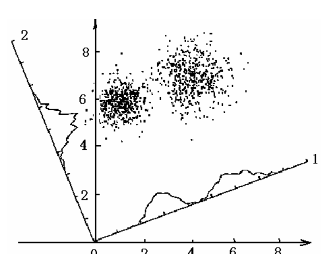
- Let *m* be the number of *features* collected from *n* nodes and *k samples* are obtained per node
  - $X^i$ (i = 1, 2, · · · , n), each representing the feature matrix collected from the *ith* node
  - Reorganize each matrix $X^i$ into a long *(m×k)* column vector
- Feature space X, a *(m×k) ×n* matrix

$$X = [x^1, x^2, ..., x^n]$$



# Feature Extraction

- Simple comparison does not work well
  - Fluctuation and noise may exist in the feature space
- Principal Component Analysis (PCA)
  - Maps a given set of data points onto new axes (i.e. principal components) ordered by the amount of data variance that they capture
- Benefits:
  - Dimension reduction
  - Independent features



# Feature Extraction

$$X_{(m\times k)\times n} \xrightarrow{\text{Normalization}} X'_{(m\times k)\times n} \xrightarrow{\text{Zero Mean}} X''_{(m\times k)\times n} \xrightarrow{\text{PCA}} Y_{s\times n}$$

PCA steps:

- Calculates the covariance matrix of *X''*

$$C = \frac{1}{n} X'' X''^{T}$$

- Calculates the *s* largest Eigenvalues of *C*

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_s$$

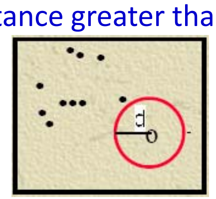
- Get projection matrix $W = [w_1, w_2, \cdots, w_s]$ and $Cw_i = \lambda_i w_i$
- Project *X''* into a new space

$$Y = W^T X$$



# Outlier Detection

- Outliers are data points that are quite "different" from the majority based on some criteria
  - Euclidean distance in the reduced feature space
- We choose the cell-based method due to its linear complexity
  - *DB (p, d):* Point *o* is a distance-based outlier if at least a fraction *p* of the objects lie at a distance greater than *d* from *o*
  - *p & d* are predefined parameters



## Outlier Detection

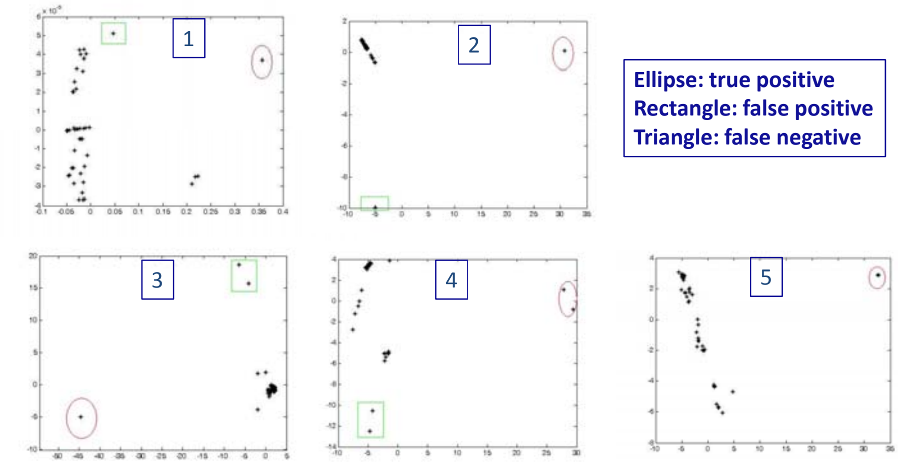
- For each outlier, calculate its anomaly score - its distance from the center of the normal nodes
- The anomaly score indicates the severity of anomaly; and the node with high anomaly score has high probability of failure.



## Experiments

- Use the Sunwulf cluster at SCS lab
  - Each node is a SUN Blade 100, 500MHz CPU, 256KB L2 Cache and 128MB main memory, 100Mbps Ethernet
  - Execute a parameter sweep application on 47 nodes (also testing on the IA64 at TG/NCSA)
- Fault injection
  1. Memory leaking
  2. Unterminated CPU intensive threads
  3. High frequent IO operations
  4. Network volume overflow
  5. Deadlock

# Experiment 1: Single Fault




$m = 11, k = 5, n = 46$

$p = 0.9565, d = 0.4\sigma, \sigma = \text{the variance of } Y$



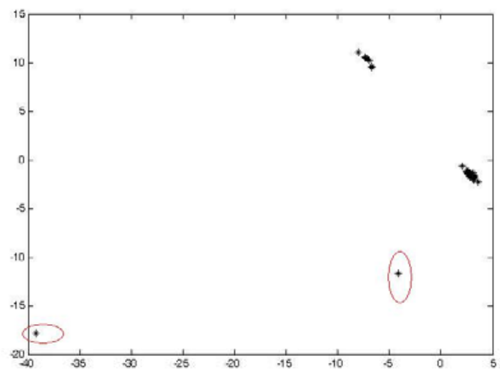
# Experiment 1

| Fault(s) | $f_n$ | $f_p$ |
|---|---|---|
| memory leak | 0 | 0.02 |
| unterminated CPU intensive threads | 0 | 0.2 |
| high frequency I/O operations | 0 | 0.06 |
| network volume overflow | 0 | 0.15 |
| deadlock | 0 | 0.06 |

***Note: $f_n$=(1-recall); $f_p$=(1-precision)***

**Type #1 faults ($f_p$<0.10): memory leaking and high frequent IO operations, deadlock;**
**Type #2 faults ($f_p$>0.10): unterminated CPU intensive threads and network volume overflow.**

# Experiment 2: Multiple Faults

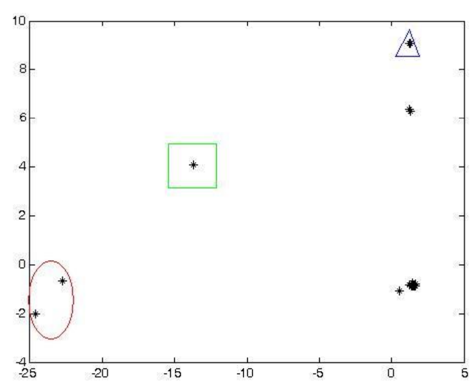



Ellipse: true positive
Rectangle: false positive
Triangle: false negative

**Results of localizing simultaneous Type #1 faults. The left point is from the node injected with high frequent IO operations, and the right one is from the node injected with a memory leaking error.**



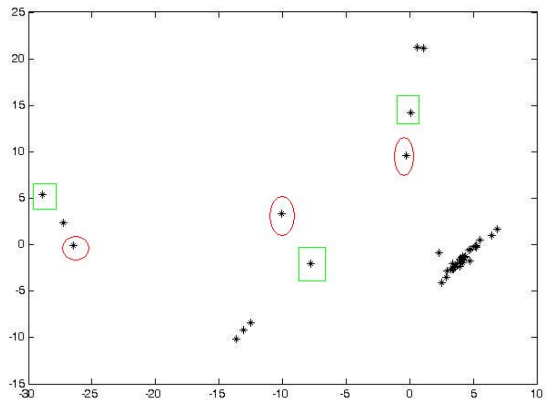
# Experiment 2: Multiple Faults




Ellipse: true positive
Rectangle: false positive
Triangle: false negative

**Results of localizing simultaneous Type #1 and #2 faults. The points inside of the ellipse are true outliers caused by network volume overflow, the point in the rectangle is a false alarm, and the point in the triangle is the missed fault caused by memory leaking.**

# Experiment 2: Multiple Faults




**Ellipse: true positive**
**Rectangle: false positive**
**Triangle: false negative**

**Results of localizing simultaneous Type #2 faults. The identified outliers, including both true outliers and false alarms, spread across the space.**



# Experiment 2: Multiple Faults

| **Fault(s)** | $f_n$ | $f_p$ |
|---|---|---|
| memory leak & high frequency I/O operation | 0 | 0 |
| memory leak & network volume flow | 0.13 | 0.15 |
| unterminated CPU intensive threads & network volume flow | 0 | 0.2 |

**Conclusion: mixed Type #1 and #2 faults are difficult to identified; and multiple Type #2 faults could lead to a high cost for finding the real faults**

# Discussion

- Benefits
  - Quick and scalable
  - Significantly reduce manual processing
- Work-in-progress
  - Hierarchical design
    - Grouping system components involved in comparable work
  - Noise reduction
    - Human involvement to reduce false positives
    - Refinement of feature matrix, e.g. FFT,….
  - Feature expansion
    - OS-level, HW-lever, and application-level



# Summary: Failure Analysis & Diagnosis

- Preliminary results are encouraging, but
  - Many issues remain open: scalable data collection, online learning, false alarms, …
  - What data is needed for better prediction/diagnosis? Which predictive methods are appropriate for online usage? …
- 100% accuracy is hardly achievable in practice, but
  - It's possible to capture cause and effect relations, with a certain accuracy
  - This can significantly improve system management
- A close collaboration between universities and supercomputing centers/labs is essential!

# Outline

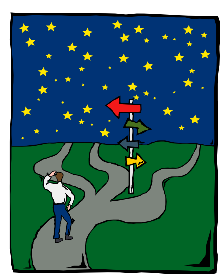
- Overview of FENCE Project
- This talk will focus on the short-term support
  - Failure analysis and diagnosis
  - Adaptive management
  - Runtime support

ILLINOIS INSTITUTE OF TECHNOLOGY



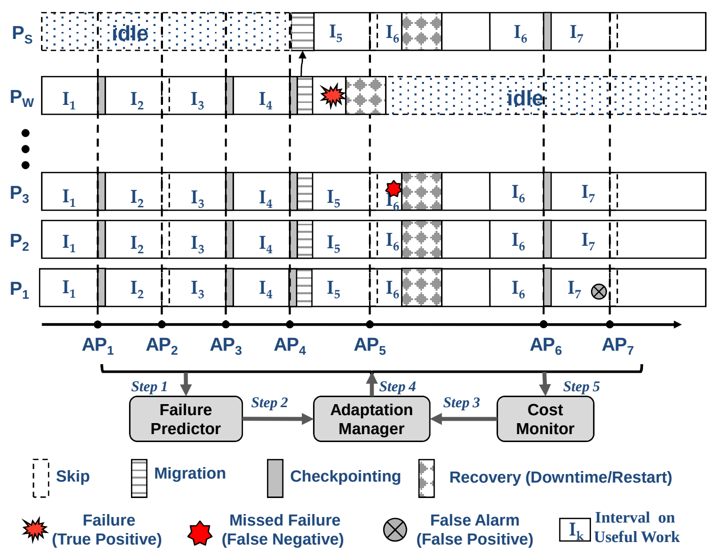
# Adaptive Fault Management

- Runtime adaptation:
  - *SKIP*, to remove unnecessary overhead
  - *CHECKPOINT*, to mitigate the recovery cost in case of unpredictable failures
  - *MIGRATION*, to avoid anticipated failures
- Challenge:
  - Imperfect prediction
  - Overhead/benefit of different actions
  - The availability of spare resources



ILLINOIS INSTITUTE OF TECHNOLOGY

# Main Idea






# Adaptive Fault Management

- MIGRATION:

$$E_{next} = (2I + C_r + C_{pm}) * f_{appl} + (I + C_{pm}) * (1 - f_{appl})$$

$$\text{where } f_{appl} = \begin{cases} 1 - \prod_{i=1}^{N_W^f - N_S^h} (1 - precision) & N_W^f > N_S^h \\ 0 & N_W^f \le N_S^h \end{cases}$$

- CHECKPOINT:

$$E_{next} = (2I + C_r + C_{ckp}) * f_{appl} + (I + C_{ckp}) * (1 - f_{appl})$$

$$\text{where } f_{appl} = 1 - \prod_{i=1}^{N_W^f} (1 - precision)$$

- SKIP:

$$E_{next} = (C_r + (2 + l_{current} - l_{last}) * I) * f_{appl} + I * (1 - f_{appl})$$

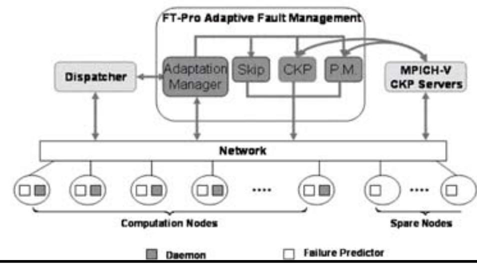
$$\text{where } f_{appl} = 1 - \prod_{i=1}^{N_W^f} (1 - precision)$$

*Select the action with min($E_{next}$)*

- An enforced FT window: $\dfrac{MTBF}{I \cdot (1\text{-}recall)}$

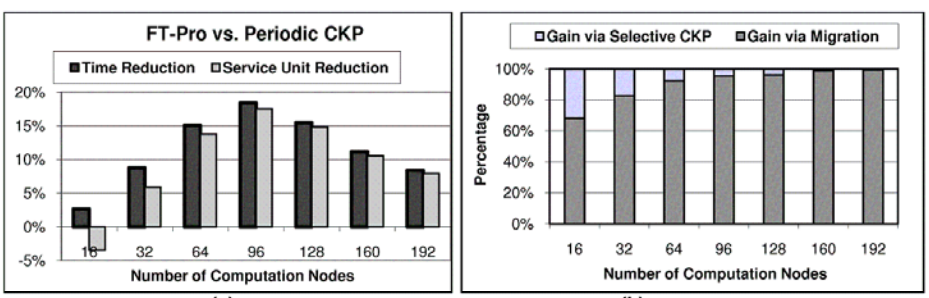
# Experiments

- Fluid Stochastic Petri Net (FSPN) modeling
  - Study the impact of computation scales, number of spare nodes, prediction accuracies, and operation costs
- Case studies
  - Implementation in MPICH-VCL, as a new module
  - Applications: ENZO, GROMACS, NPB
  - TeraGrid/ANL IA32 Linux Cluster



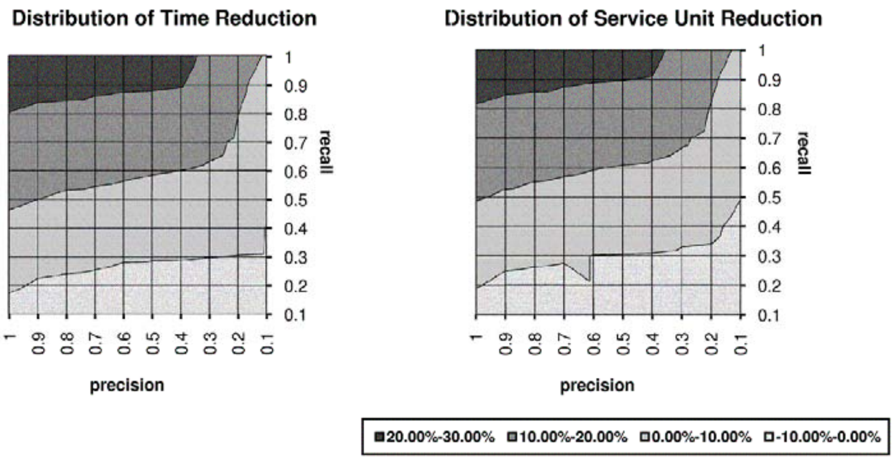



# Impact of Computation Scale




(a) (b)

# Impact of Prediction Accuracy






# Summary: Adaptive Management

- The adaptation manager can be easily implemented on top of existing checkpointing tools
  – MPICH-V, LLNL's CKP tool, LAM/MPI, …
- Results indicate that:
  – It outperforms periodic checkpointing as long as recall and precision are higher than 0.30
  – Lower than 3% overhead
- A better migration support is needed
  – Currently, a stop-and-restart approach

# Outline

- Overview of FENCE Project
- This talk will focus on the short-term support
  - Failure analysis and diagnosis
  - Adaptive management
  - Runtime support

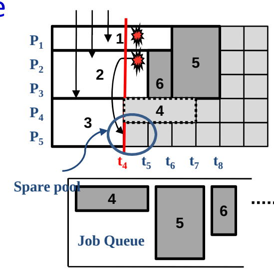


# Runtime Support

- Development /optimization of fault tolerance techniques
  - Live migration support
  - Dynamic virtual machine
  - Fast fault recovery
- System-wide node allocation and job rescheduling
  - Nodes for regular scheduling vs. spare nodes for failure prevention
  - Selection of jobs for rescheduling in case of multiple simultaneous failures

# Resource Allocation

- Observation:
  - Spare nodes are common in production systems, even in the systems under high load
  - E.g. prob(at least 2% of system nodes are idle) >= 70%
- A dynamic and non-intrusive strategy
  - Spare nodes are determined at runtime
  - Always keeps the original scheduling reservation






# Job Rescheduling

- Transform into a general 0-1 Knapsack model

To determine a binary vector $X = \{x_i \mid 1 \le i \le J_s\}$

maximize $\sum_{1 \le i \le J_s} x_i \cdot v_i \;,\; x_i = 0 \text{ or } 1$

s.t. $\sum_{1 \le i \le J_s} x_i \cdot p_i^s \le S$

- Generate three different strategies by setting V:
  - *Service unit loss driven*, to minimize the loss of service units
  - *Job failure rate driven*, to reduce number of failed jobs
  - *Failure slowdown driv*en, to minimize the slowdown caused by failures

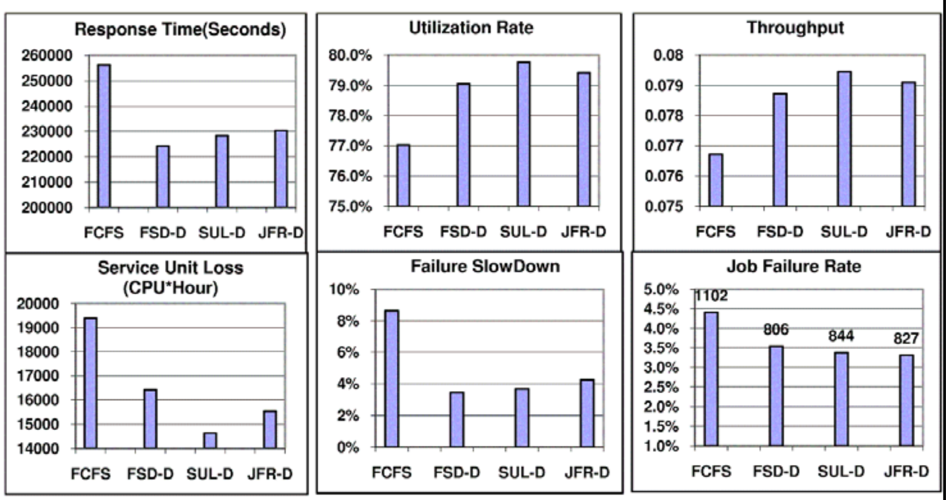
# Experiments

- Event-based simulations
  - Synthetic data & system traces
  - Evaluation metrics
    - Performance metrics (system utilization,  avg. response time, throughout) &  reliability metrics (service unit loss, job failure rate, failure slowdown)

- As long as failure prediction is capable of predicting 20% of failures with a false alarm rate lower than 80%, a positive gain is observed by using fault-aware runtime management



# Results with System Traces



# Related Work

- Failure analysis and prediction
  - Hardware sensors, e.g. lm_sensor, SMART, …
  - Predictive methods: model based or data driven [Trivedi’99, Weiss’98,Hoffmann’04,Ma’02, Sahoo’03, Liang’06,…]
- Checkpointing/restart
  - A detailed description in [Elnozahy’02]
  - Checkpointing tools: libckpt, BLCR, LAM/MPI, MPICH-V, LLNL’s tool, …



# Related Work (cont.)

- Process/object migration
  - Stop-and-restart [Tannenbaum’95]
  - Live migration [Chakravorty’05, Du’06,Wang’07,Clark’05,..]
- Other resilience supports
  - HA-OSCAR for high availability of head nodes
  - Failure-aware scheduling [Alberts’01, Hariri’86, Kartik’97, Shatz’92, Srinivasan’99, Oliner’05,Zhang’04,…]
  - Reliability modeling [Young’74, Garg’96, Wu’07,…]

# Conclusions

- FENCE (Fault awareness ENabled Computing Environment)
  - Potential for better failure analysis and diagnosis
    - Captures 65+% of failures, with the false alarm rate less than 35%
  - Up to 50% improvement in system productivity
  - Up to 43% reduction in application completion time

**“Adaptation is key” (D. Reed)**

**“It is not cost-effective or practical to rely on a single fault tolerance approach for all applications and systems” (Scarpazza, Villa, Petrini, Nieplochar, …)**



# References


- “Adaptive Fault Management of Parallel Applications for High Performance Computing”, to appear in *IEEE Trans. on Computers*.
- “Fault-Aware Runtime Strategies for High Performance Computing”, *submitted*.
- "Performance under Failure of High-End Computing", *SC'07*
- “Anomaly Localization in Large-scale Clusters”, *Cluster’07*.
- “A Meta-Learning Failure Predictor for Blue Gene/L Systems”, *ICPP’07*.
- “Fault-Driven Re-Scheduling for Improving System-Level Fault Resilience”, *ICPP’07*.
- “A Fault Diagnosis and Prognosis Service for TeraGrid Clusters”, *The 2nd TeraGrid Conference*.
- “Exploit Failure Prediction for Adaptive Fault-Tolerance in Cluster Computing”, *CCGrid06*.
- "MPI-Mitten: Enabling Migration Technology in MPI", *CCGrid'06.*
- “FT-Pro: Adaptive Fault Management for High Performance Computing”, *research poster at SC’05*, 2005.

# Questions?

**FENCE Project Website:**

**http://www.cs.iit.edu/~zlan/fence.html**