FENCE: Fault awareness ENabled Computing Environment

Zhiling Lan

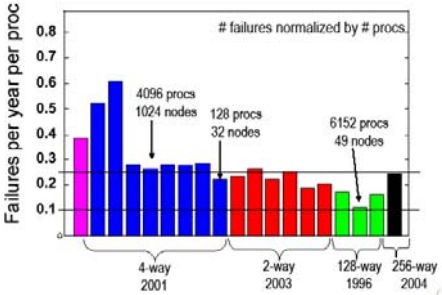
Illinois Institute of Technology (Dept. of CS)

lan@iit.edu

Reliability Concerns

- Systems are getting bigger
  - 1024-4096 processors is today's "medium" size (73.4% of TOP500)
  - 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

☞ Need for fault management
Losing the entire job due to one node's failure is costly in time and CPU cycles!



From "Simplicity and Complexity in Data Systems at Scale", Garth Gibson, Hadoop Summit, 2008

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

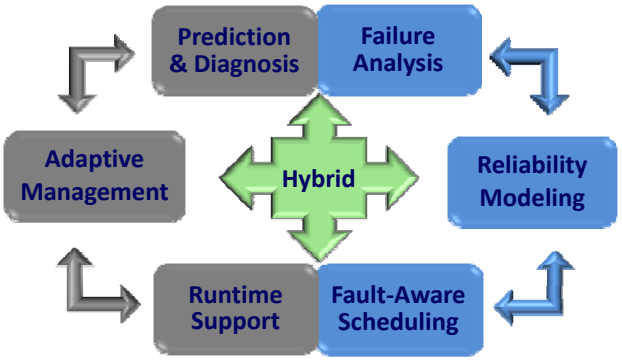
☞ **FENCE: Fault awareness ENabled Computing Environment**

- **A "fence" to protect system and appl. from severe failure impact**
- **Exploit the synergy between various methods to advance fault management**

FENCE Overview

- Adopt a hybrid approach:
  - *Offline analysis:* reliability modeling and scheduling enables intelligent system configuration and mapping
  - *Runtime support*: to avoid and mitigate imminent failures
- Explore runtime adaptation:
  - *Proactive actions* prevent applications from anticipated failures
  - *Reactive actions* minimize the impact of unforeseeable failures
- Address fundamental issues
  - Failure prediction & diagnosis
  - Adaptive management
  - Runtime support
  - Reliability modeling & scheduling

Key Components



Outline

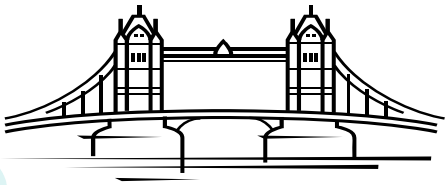
- Overview of FENCE Project
  - Project website: http://www.cs.iit.edu/~zlan/fence.html
- This talk will focus on runtime support
  - Failure prediction and diagnosis
  - Adaptive management
  - Runtime support

Outline

- Overview of FENCE Project
  - Project website: http://www.cs.iit.edu/~zlan/fence.html
- This talk will focus on runtime support
  - Failure prediction and diagnosis
  - Adaptive management
  - Runtime support

Prediction & Diagnosis

Failure Prediction and Diagnosis



Health/perf monitoring:

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

e.g. Blue Gene series and Cray XT series

Fault tolerance methods:

- Checkpointing

(open MPI, MPICH-V, BLCR, ….)

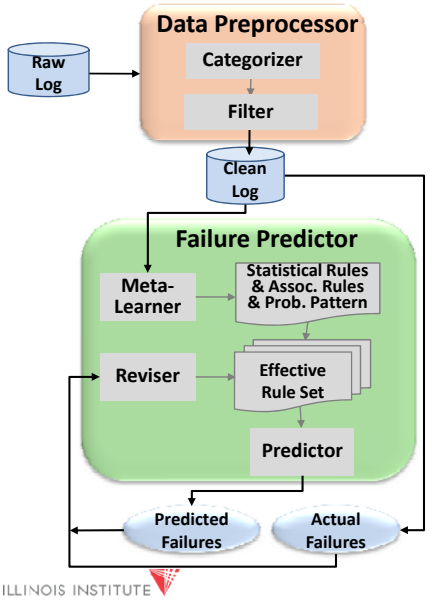
- Process/object migration
- Other resilience supports

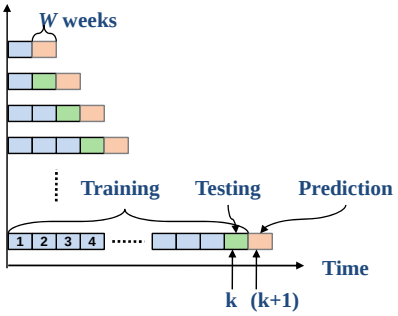
Failure Prediction and Diagnosis

- Challenges:
  - Potentially overwhelming amount of data collected across the system
    - Fault patterns and root causes are often buried like needles in a haystack!
  - Faults are many and complex
    - There is no one-size-fit-all predictive method!

- Our approaches:
  - Runtime failure prediction
    - Exploit ensemble learning, data mining, and statistic learning
  - Automated anomaly localization
    - Utilize pattern recognition techniques

Runtime Failure Prediction





1) Meta-learner: dynamic training by integrating multiple base classifiers
2) Reviser: dynamic testing by tracing prediction accuracy
3) Predictor: event-driven prediction

Case Studies on Blue Gene/L

	SDSC BGL	ANL BGL
Start Date	12/6/2004	1/21/2005
End Date	06/11/2007	2/28/2007
Log Size after clean	704 MB	1.36 GB

- Evaluation metrics:

$$precision = \frac{T_p}{T_p + F_p}$$

$$recall = \frac{T_p}{T_p + F_n}$$

- A good prediction should achieve a high value (close to 1.0) for both metrics

Preprocessing

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

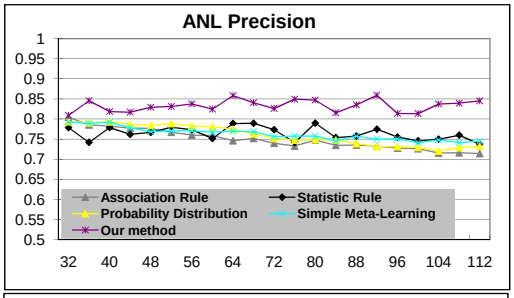
BGL Event Categories

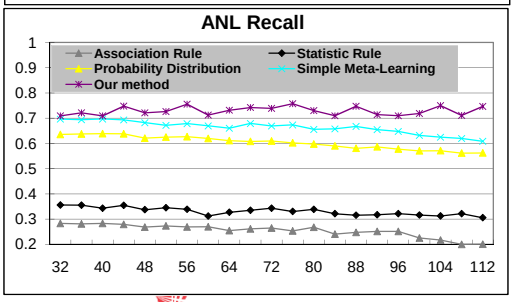
Main Category	Subcategories	Examples
Application	12	loadProgramFailure, loginFailure, nodemapCreateFailure,…
Iostream	8	socketReadFailure, streamReadFailure,…
Kernel	20	alignmentFailure, dataAddressFailure, instructionAddressFailure, …
Memory	22	cachePrefetchFailure, dataReadFailure, dataStoreFailure,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

Base Classifiers

- Statistical rules:
  - Discover *statistic correlations among fatal events, i.e.,* how often and with what probability will the occurrence of one failure influence subsequent failures
- Association rules:
  - Examine *causal correlations between non-fatal and fatal events,* where rules are in the form (*X =>Y)*
    - If X occurs, then it is *likely* that Y will occur
- Probability distribution:
  - Use maximum likelihood estimation (MLE) to obtain the distribution of failures
    - For both logs, Weibull is a good fit

Prediction Results (1)

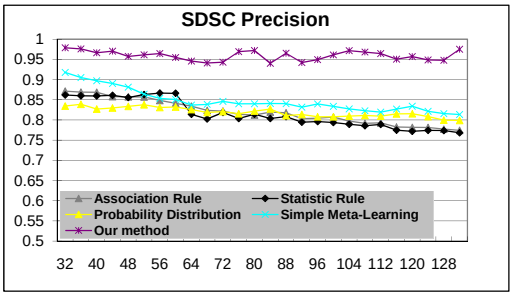


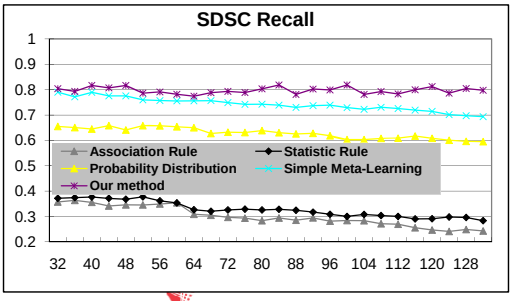


- Precision is between 0.8-0.9, meaning less than 20% false alarms
- Recall is between 0.7-0.8, meaning being capable of capturing over 70% failures

W=4 weeks
support=0.06
confidence=0.6

Prediction Results (2)



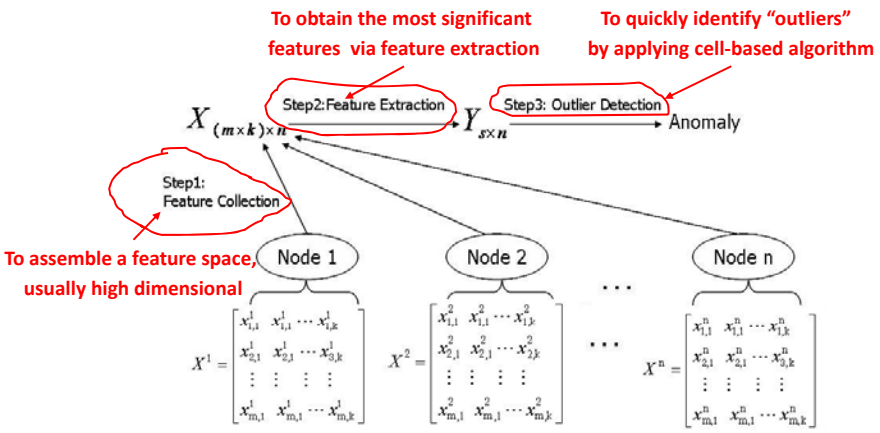


- Precision is between 0.9-1.0, meaning less than 10% false alarms
- Recall is around 0.8, meaning being capable of capturing 80% failures

Automated Anomaly Localization

- To quickly locate faulty nodes in the system with little human interaction
- Primary objectives:
  - High *precision,* meaning low false alarm rate
  - Extremely high *recall* (close to 1.0)
- Two key observations:
  1) Nodes performing comparable activities exhibit similar behaviors
  2) In general, the majority is functioning normally since faults are rare events

Automated Anomaly Localization



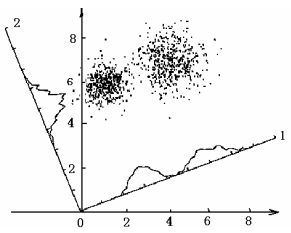
Feature Collection

- Construct *feature space* of the system:
  - A feature is an individual measurable property of the node being observed
  - Examples: CPU usage, memory usage, IO performance, …, using system calls *vmstat, mpstat, iostat & netstat*
- 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
- So Feature space X is a *(m×k) ×n* matrix

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

Feature Extraction

- Goal:
  - Dimension reduction
  - Independent features
- Principal Component Analysis (PCA)
  - A linear transformation onto new axes (i.e. principal components) ordered by the amount of data variance that they capture



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*

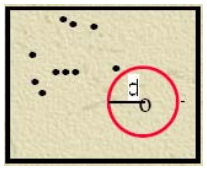
$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_s$$

- 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

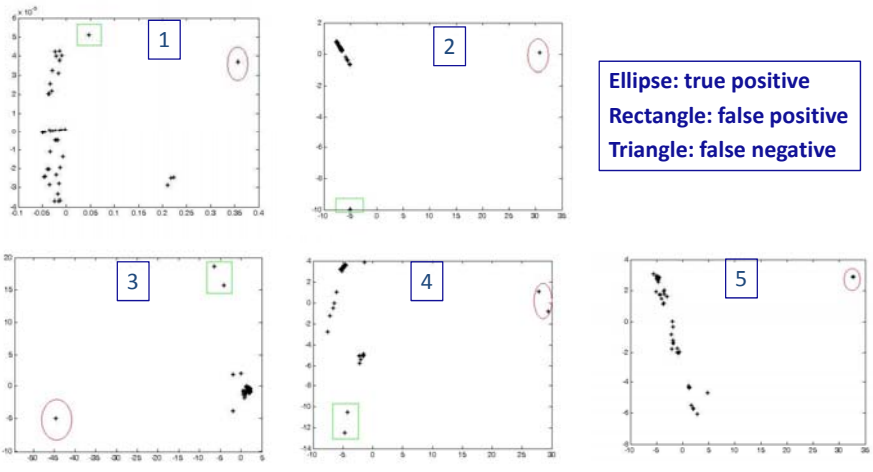
- Outliers are data points that are quite "different" from the majority based on Euclidean distance
- We choose the cell-based method due to its linear complexity
  - *DB (p, d):* Point *o* is a distance-based outlier if at least a fraction *p* of the objects lie at a distance greater than *d* from *o*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



Experiments

- Use 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
- Manual fault injection
  1) Memory leaking
  2) Unterminated CPU intensive threads
  3) High frequent IO operations
  4) Network volume overflow
  5) Deadlock

Result: Single Fault



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

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

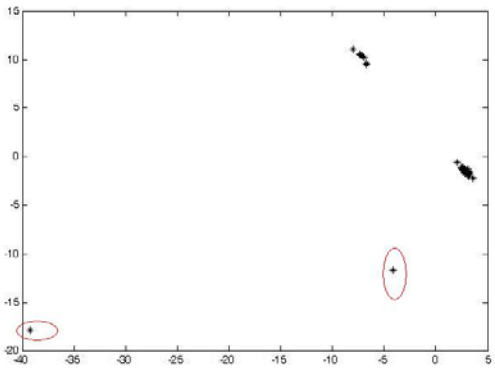
$p = 0.9565, d = 0.4\sigma, \sigma =$ the variance of Y

Result: Single Fault

Faults	Recall	Precision
Memory leaking	1	0.98
Unterminated CPU intensive threads	1	0.80
High frequency IO	1	0.94
Network volume overflow	1	0.85
Deadlock	1	0.94

Type #1 faults (precision>0.90): memory leaking and high frequent IO operations, deadlock;
Type #2 faults (precision<0.90): unterminated CPU intensive threads and network volume overflow.

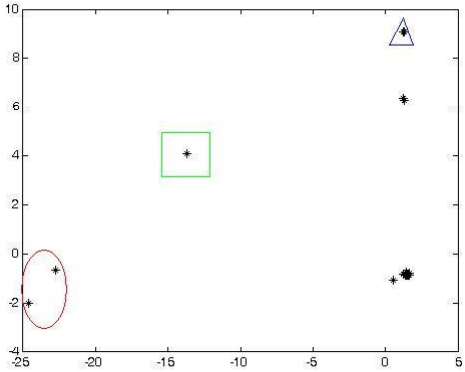
Result: Multiple Faults



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.

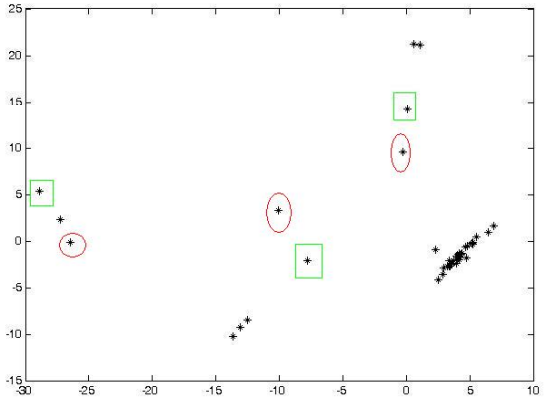
Result: Multiple Faults



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.

Result: 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.

Result: Multiple Faults

Faults	Recall	Precision
Memory leak & high frequency I/O	1	1.0
Memory leak & network volume flow	0.87	0.85
Unterminated CPU intensive threads & network volume flow	1	0.80

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

Outline

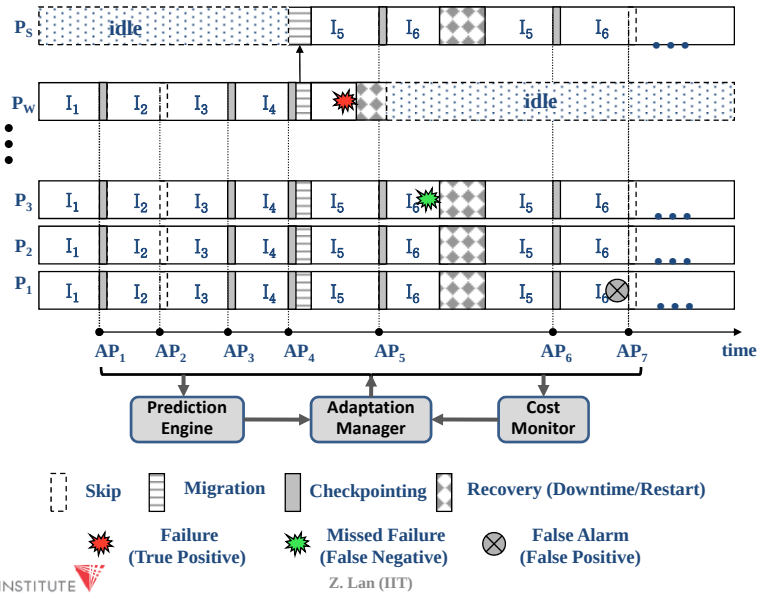
- Overview of FENCE Project
  - Project website: http://www.cs.iit.edu/~zlan/fence.html
- This talk will focus on runtime support
  - Failure prediction and diagnosis
  - Adaptive management
  - Runtime support

Adaptive Management

Adaptive Management

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

Main Idea



Adaptation Manager

- MIGRATION:

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

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

- CHECKPOINT:

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

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

- SKIP:

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

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

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

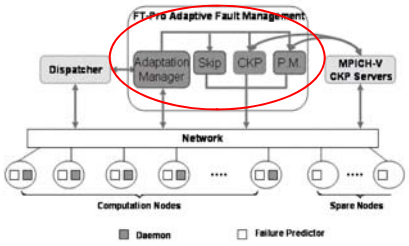
- An enforced FT window: $\frac{MTBF}{I \cdot (1 - recall)}$

Adaptive Manager

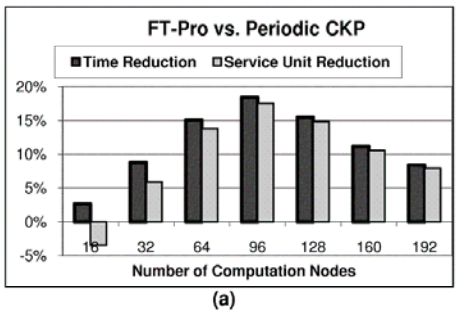
- It can be easily implemented with existing checkpointing tools
  - MPICH-V, LAM/MPI, …
- Being fully operational with
  - Runtime failure prediction
  - Checkpoint support
  - Migration support
    - Currently, a stop-and-restart approach

Experiments

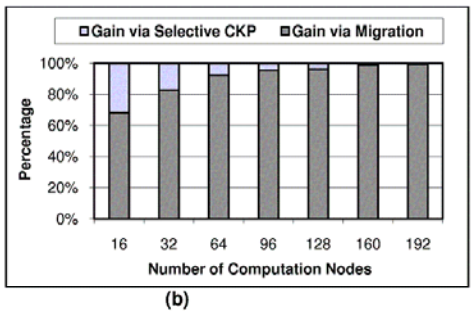
- Fluid Stochastic Petri Net (FSPN) modeling
  - Study the impact of computation scales, number of spare nodes, prediction accuracies, and operation costs
- Case studies
  - Implementation with MPICH-V, as a new module
  - Applications: ENZO, GROMACS, NPB
  - TeraGrid/ANL IA32 Linux Cluster



Impact of Computation Scale

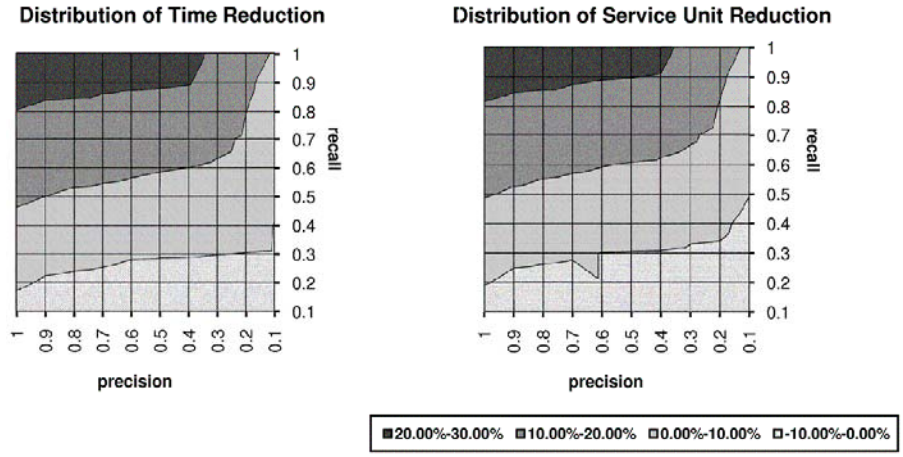


(a)



(b)

Impact of Prediction Accuracy



It outperforms periodic checkpointing as long as *recall* and *precision* are higher than 0.30

Outline

- Overview of FENCE Project
  - Project website: http://www.cs.iit.edu/~zlan/fence.html
- This talk will focus on runtime support
  - Failure prediction and diagnosis
  - Adaptive management
  - Runtime support

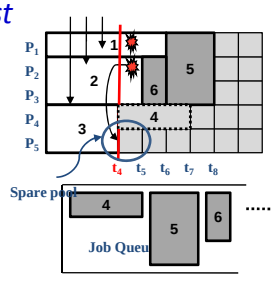
Runtime Support

Runtime Support

- Fault-aware runtime rescheduling
  - Focus on re-allocating *active jobs* (i.e., running jobs) to avoid imminent failures
    - Allocate spare nodes for failure prevention
    - Select active jobs for rescheduling in case of resource contention
- Fast failure recovery
  - Enhance checkpoint/recovery to reduce restart latency
  - To appear in the Proc. of DSN'08

Spare Node Allocation

- Observation:
  - Idle nodes are common in production systems, even in the systems under high load
  - Our studies have shown that *prob(at least 2% of system nodes are idle) >= 70%*
- A dynamic and non-intrusive strategy
  - Spare nodes are determined at runtime
  - Guarantee job reservations made by batch scheduler



Job Rescheduling

- Transform into a general 0-1 Knapsack model

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

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

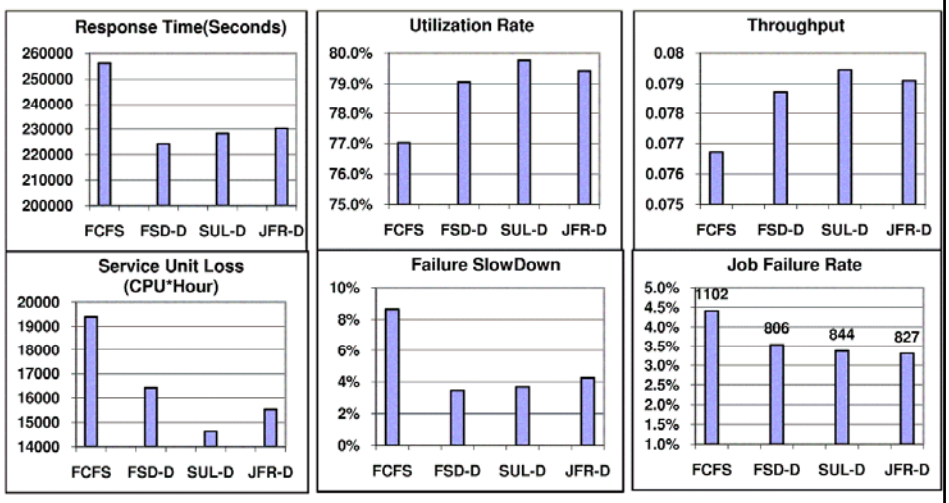
s.t. $\sum_{1 \le i \le J_s} x_i \cdot p_i^s \le S$

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

Experiments

- Event-based simulations
  - FCFS/EASY Backfilling
  - Compare system productivity W/ vs. W/O fault-aware runtime rescheduling
  - Evaluation metrics
    - Performance metrics -- system utilization, 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 rescheduling

Results with System Traces



Summary (1/2)

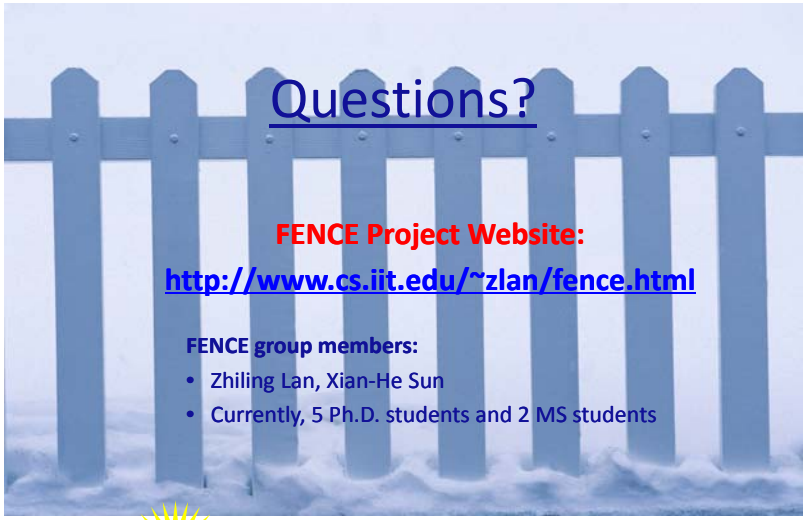
- Preliminary results are very encouraging
  - It's possible to capture failure cause and effect relations by exploring data mining and pattern recognition technologies
    - Towards automated failure prediction and diagnosis
  - Runtime adaptation can significantly improve system productivity and application performance
- But, many issues remain open
  - Development of automated failure prediction and diagnosis engine
    - Stream data processing
    - Extensive evaluation with production systems, such as Blue Gene series

Summary (2/2)

- Development of fault-aware resource management and job scheduling
  - Resource allocation for failure prevention & recovery
  - Automated job recovery
    - Fault-aware batch scheduler
    - Simulation study
- A close collaboration with national labs and supercomputing centers is essential!
- Integration with Fault Tolerant Backplane

Selected Publication

- Z. Lan and Y. Li, "Adaptive Fault Management of Parallel Applications for High Performance Computing", to appear in ***IEEE Trans. on Computers***, 2008.
- Y. Li, Z. Lan, P. Gujrati, and X. Sun, "Fault-Aware Runtime Strategies for High Performance Computing", ***IEEE Trans. on Parallel and Distributed Systems***, under minor revision
- Y. Li and Z. Lan, "A Fast Recovery Mechanism for Checkpointing in Networked Environments", *Proc. of* ***DSN'08***, 2008.
- J. Gu, Z. Zheng*, Z. Lan, J. White, E. Hocks, B.H. Park, "Dynamic Meta-Learning for Failure Prediction in Large-Scale Systems: A Case Study", *Proc. of* ***ICPP'08,*** 2008.
- Z. Lan, Y. Li, Z. Zheng, and P. Gujrati, "Enhancing Application Robustness through Adaptive Fault Tolerance", *Proc. of* ***NSFNGS Workshop (in conjunction with IPDPS)***, 2008.
- X. Sun, Z. Lan, Y. Li, H. Jin, and Z. Zheng, "Towards a Fault-Aware Computing Environment", *Proc. of* ***High Availability and Performance Computing Workshop***, 2008.
- Z. Zheng, Y. Li and Z. Lan, "Anomaly Localization in Large-scale Clusters", *Proc. of IEEE* ***Cluster'07***, 2007.
- P. Gujrati, Y. Li, Z. Lan, R. Thakur, and J. White, "A Meta-Learning Failure Predictor for Blue Gene/L Systems", *Proc. of* ***ICPP'07***, 2007.
- Y. Li, P. Gujrati, Z. Lan, and X. Sun, "Fault-Driven Re-Scheduling for Improving System-Level Fault Resilience", *Proc. of* ***ICPP'07***, 2007.
- Y. Li and Z. Lan, "Exploit Failure Prediction for Adaptive Fault-Tolerance in Cluster Computing", *Proc. of* ***CCGrid'06***, 2006.



Questions?

FENCE Project Website:

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

FENCE group members:

- Zhiling Lan, Xian-He Sun
- Currently, 5 Ph.D. students and 2 MS students

To our sponsor: National Science Foundation