



Building a Fault-Aware Computing Environment

Xian-He Sun

Illinois Institute of Technology
sun@iit.edu

In collaboration with
Zhiling Lan, and the FENCE group at IIT



Reliability Concerns

- Systems are getting bigger
 - 1024-4096 processors is today's "medium" size (>54% on the recent TOP500 List)
 - $O(10,000) \sim O(100,000)$ 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



A needs for new fault management!
Simple checkpointing is not good enough!



Outline

- Motivation
 - The FENCE project
- Post analysis
 - Modeling and scheduling
 - Workflow
- Runtime analysis
 - Adaptation and diagnoses
 - Identify the location of the failure
- System support
 - Dynamic virtual machine
 - Live migration
- Conclusion



Fault Tolerance: The big picture

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

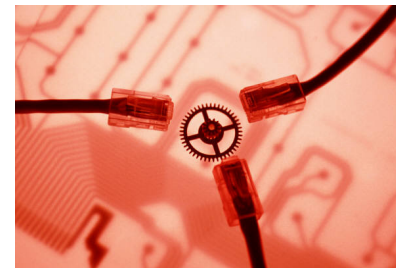
FENCE: Fault awareness EEnabled Computing Environment

- A “fence” to minimize failures via proactive avoidance
- Exploit the synergy between various methods to advance fault management



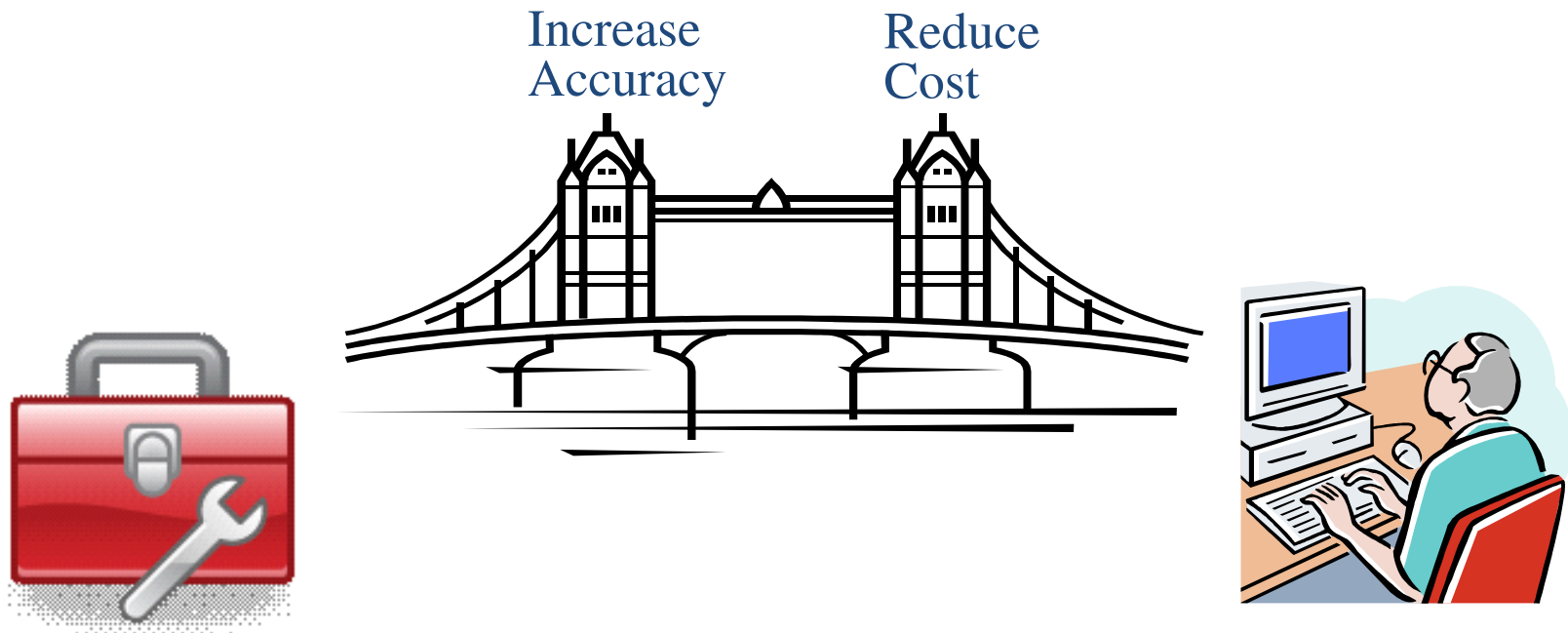
FENCE Overview

- Adopt a **hybrid approach**:
 - *Post analysis*: reliability modeling and scheduling enables intelligent system configuration & mapping
 - *Runtime analysis*: 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**
 - Reliability modeling & scheduling
 - Failure analysis & adaptation
 - Runtime support
 - Failure management





Basic Idea and Essential Components



Fault-aware mechanism:

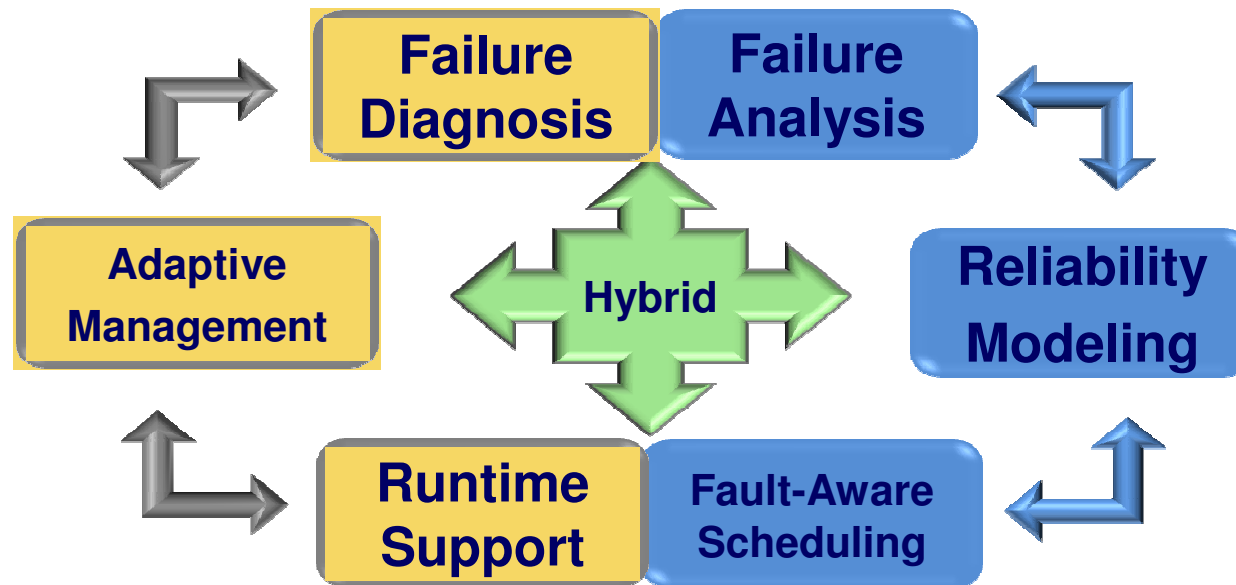
- Proactive: prediction/scheduling
- Proactive: prediction/adaptation
- Reactive: Checkpointing
- Combined approach
- Other resilience supports

Fault-aware environment:

- System monitoring tools
- Dynamic configuration
- Runtime support system
- Dynamic virtual machine



Key Components



Post analysis fault management:

System-level (Proactive) - checkpointing, replication, etc

Application-level (Proactive) - task scheduling

Runtime adaptation:

System-level (Reactive) – rescheduling, preemptive rescheduling

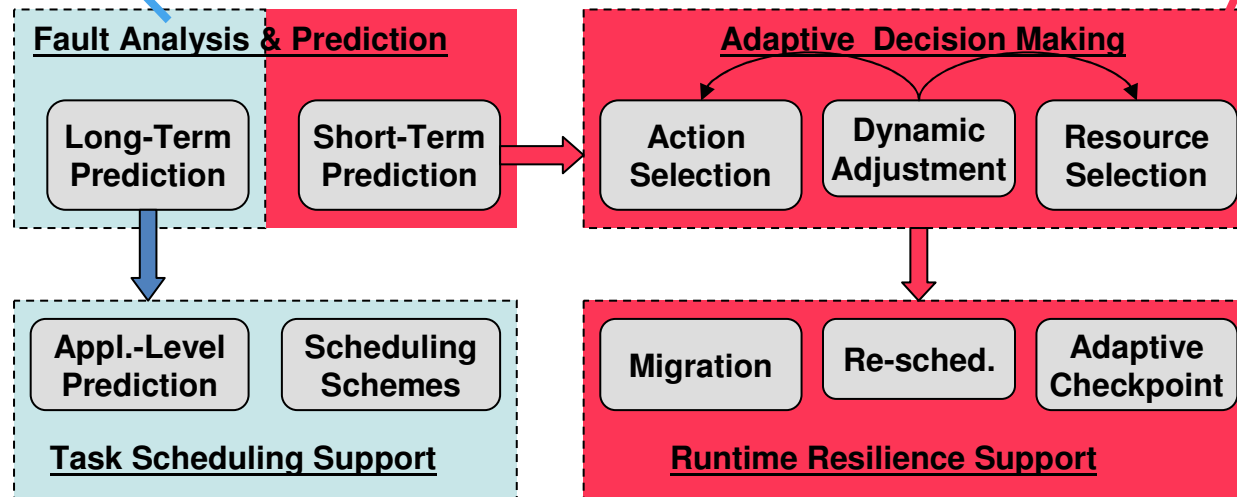
Application-level (Proactive) - checkpointing, migrating, etc.



FENCE Design

Long-term support

Short-term/runtime support



- Consists of both long-term and short-term support
 - Post analysis: Long-term failure modeling & failure-aware scheduling
 - Runtime analysis: Short-term failure prediction & runtime support
- Integrates reactive and proactive fault tolerance techniques

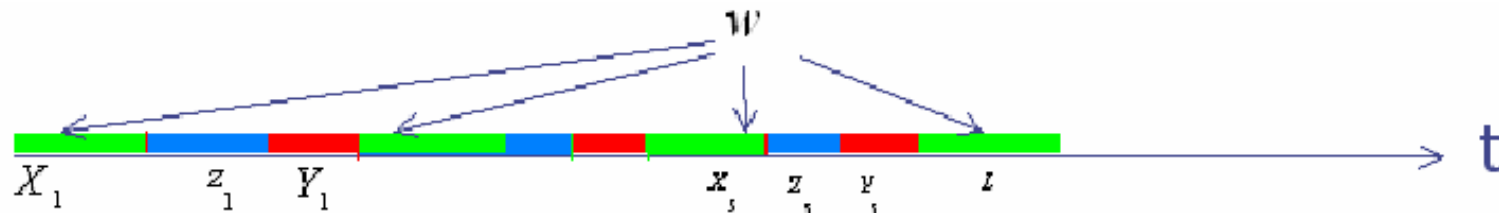


Post Analysis: The Grid Harvest Service (GHS) System

- A **long-term** application-level QoS prediction and scheduling system
- Started for performance (resource sharing), extended to fidelity, and then faulty tolerant (reliability)
- A new prediction model derived by probability analysis and simulation, another based on AI
- Non-intrusive measurement and a set of scheduling algorithms
- System design, implementation and testing

<http://meta.cs.iit.edu/~ghs/>

Accuracy: Model the Impact of Failure



The completion time of the application:

$$T = X_1 + Y_1 + Z_1 + X_2 + Y_2 + Z_2 + \dots + X_s + Y_s + Z_s + L$$

The whole system can be considered as M/G/1 queuing system

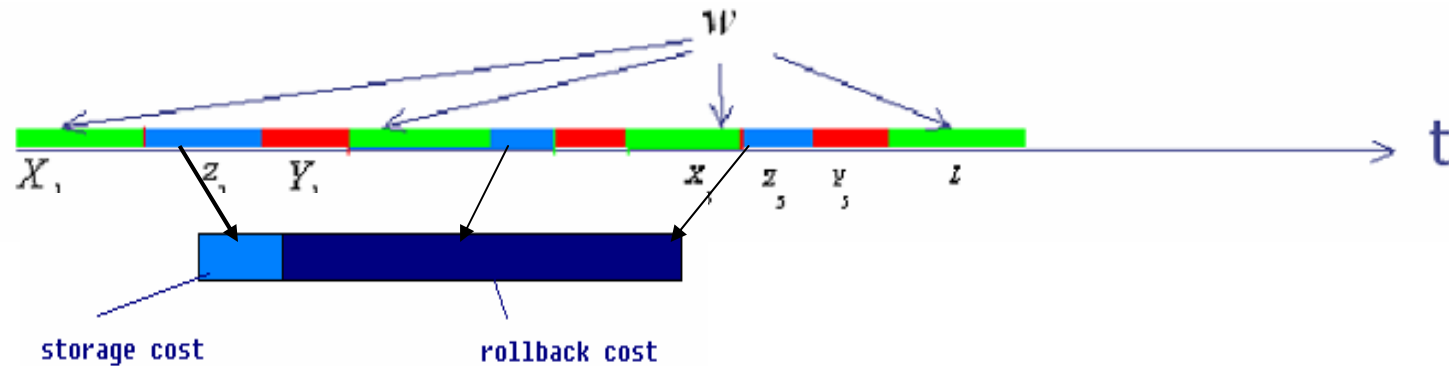
$$\Pr(T \leq t) = \begin{cases} e^{-\lambda_f w} + (1 - e^{-\lambda_f w}) \Pr(U(S) \leq t - w \mid S > 0) & \text{if } t \geq w \\ 0, & \text{otherwise} \end{cases}$$

$$\Pr(T \leq t) = \Pr(\text{Max}\{T_k, k = 1, 2, \dots, m\} \leq T)$$

$$= \prod_{k=1}^m \Pr(T_k \leq t)$$



Model the Impact of Checkpointing



$$E(T) = \left(\frac{1}{1 - \lambda_f \mu_f} + \frac{\alpha}{\gamma} + \frac{1 - e^{-\lambda_f \gamma} - \lambda_f \gamma e^{-\lambda_f \gamma}}{(1 - e^{-\lambda_f \gamma})} \right) w$$

$$V(T) = \left(\frac{\mu_f^2 + \sigma_f^2}{(1 - \lambda_f \mu_f)^3} + \mu_c^2 + \sigma_c^2 + 2 \frac{\mu_f \mu_c}{1 - \lambda_f \mu_f} \right) \lambda_f w$$

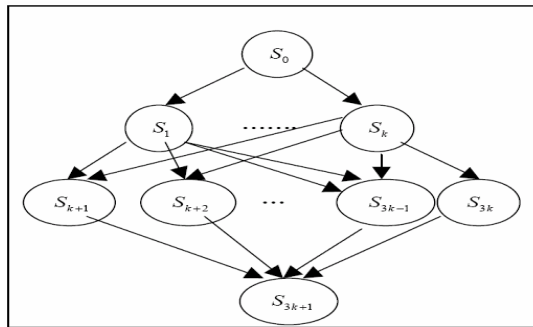
By solving the function $\frac{\partial E(T)}{\partial \gamma} = 0$

We can find the **optimal checkpointing period**

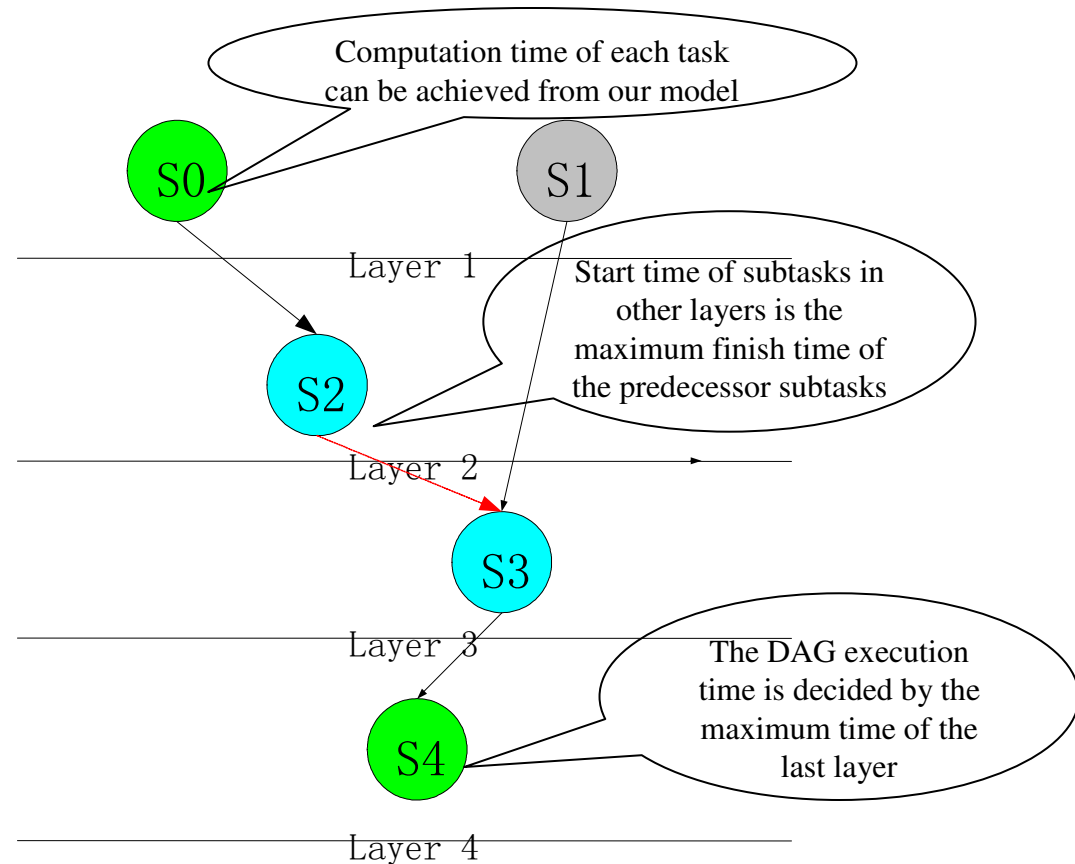
Combine the failure impact and fault management cost we can optimize the system configuration and application scheduling



Extended to DAG and Workflow



LQCD workflow

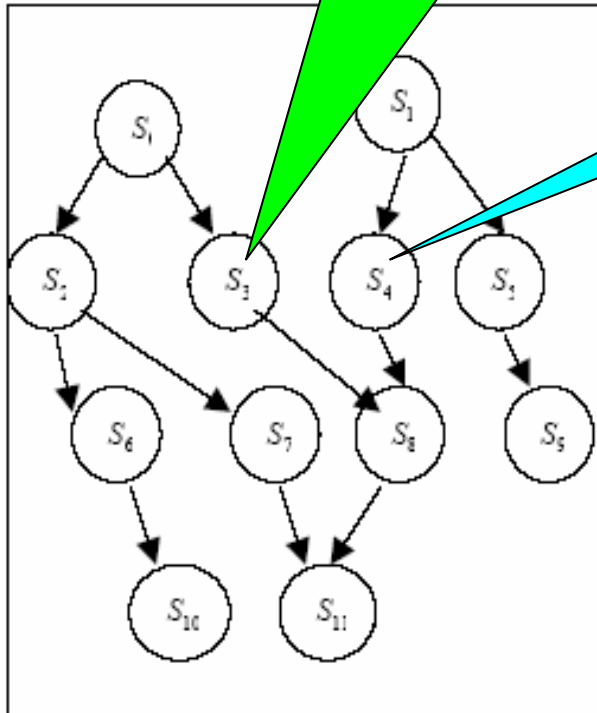


The layer depends on scheduling
The model can find the "weak point" of the workflow



$(\Pr(T_{S_3}^F \leq 500) = 80\%) > (\text{threshold} = 75\%)$
not a weak node

$(\Pr(T_{S_4}^F \leq 600) = 60\%) < (\text{threshold} = 75\%)$
a weak node!



a. A DAG Task Graph

Subtask	Base Workload
S_1	100
s_1	200
s_2	250
s_3	300
s_4	200
s_5	350
s_6	200
s_7	300
s_8	200
s_9	300
s_{10}	300
s_{11}	200

b. Base Workload (Hours)

Algorithm to find weak nodes

for each $S_i \in \Omega(S)$

if $(\Pr(T_{S_i}^F \leq E_{S_i}^F) < \text{threshold})$

S_i is a weak node(subtask)

end if

end for



A Heuristic Fault-Aware Scheduling Algorithm

Begin

List a set of idle machines in the order of their reliability over an observed time period,

$$M = \{m_1, m_2, \dots, m_q\}$$

Sort the list of idle machines in an decreasing order with

$$M' = \{c_1, c_2, \dots, c_q\} \quad ; \quad \frac{(1 - \rho_{c,k})\tau_k}{1 + \rho_{c,k} - \rho_{c,k}\rho_{f,k}} ;$$

$$a = 1, \quad b = \min\{|M'|, \frac{w}{4 * (\mu_{f,k} + \mu_{c,k})}\}$$

Repeat

$$c = \lfloor (a+b)/2 \rfloor$$

/ f(x) denotes $E(T_{C(x)})(1 + Coe.(T_{C(x)}))$ where $C(x) = \{c_1, c_2, \dots, c_x\}$ */*

If $f(a) = \min\{f(a), f(b), f(c)\}$ **then** $b = c$

Else If $f(b) = \min\{f(a), f(b), f(c)\}$ **then** $a = c$

Else If $f(c) < f(c+1)$ **then** $b = c$

Else $a = c$

Until $a+1 = b$

If $f(a) < f(b)$ **then** Assign parallel task to the machine set $C(a)$;

Else Assign parallel task to the machine set $C(b)$;

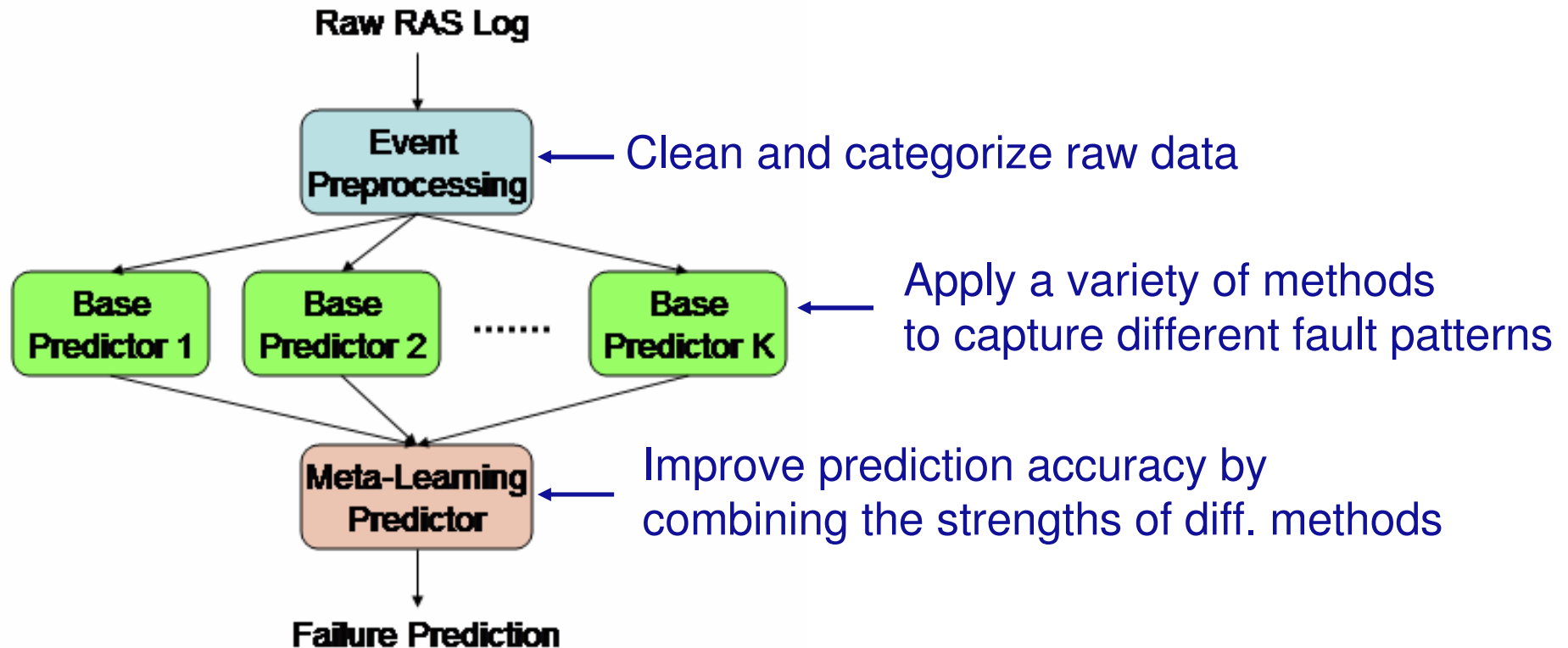
End



Runtime Analysis: Short-term Diagnosis & Prediction

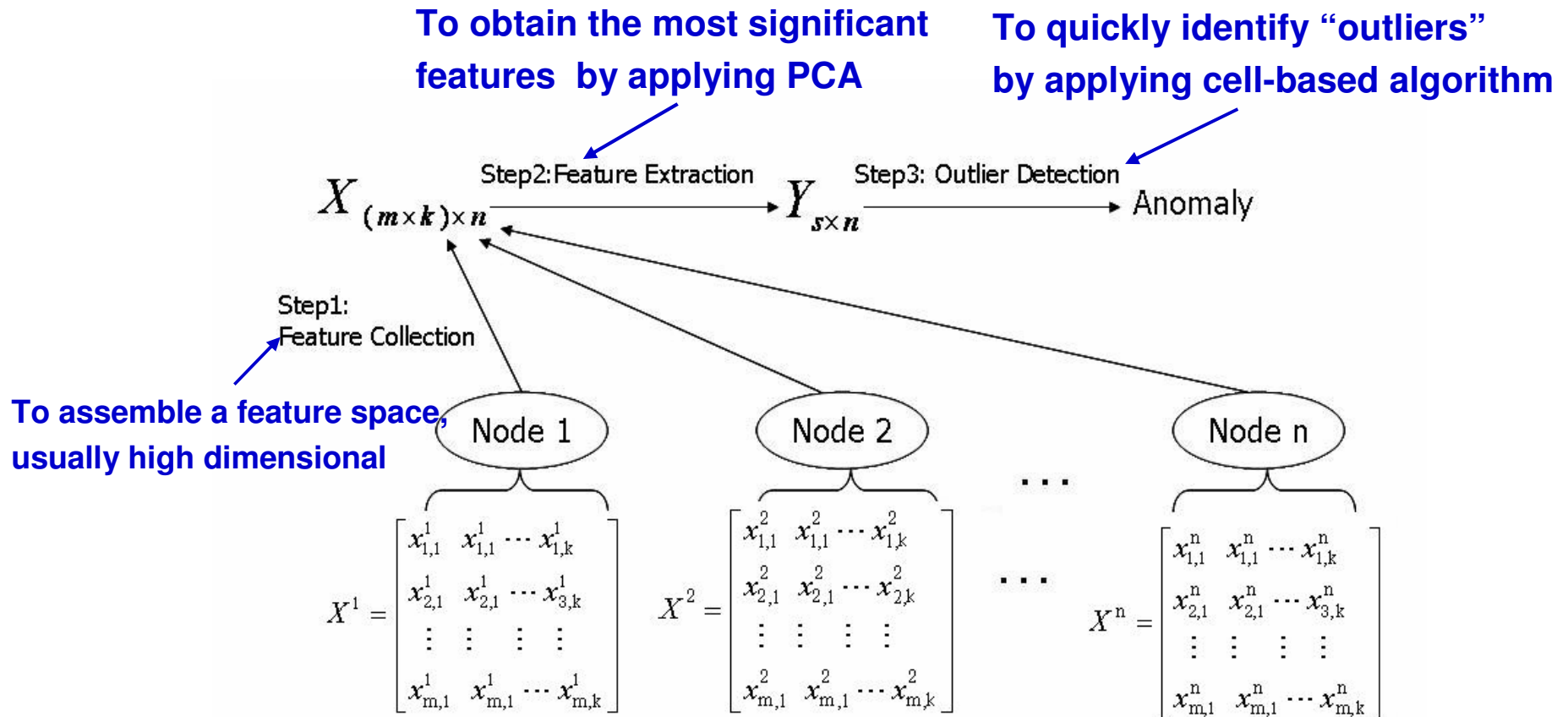
- Better accuracy:
 - Integrate multiple data sources: RAS log, perf data, sensor readings, ...
 - Coordinate data-driven methods: statistical learning, data mining, pattern recognition, ensemble learning (meta-learning)
- Answer the “when” question
 - Ensemble learning based prediction
- Answer the “where” question
 - PCA (Principal component analysis) based localization
- Adaptive action

Ensemble Learning Based Prediction





PCA based Localization



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



Adaptive Fault Management

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

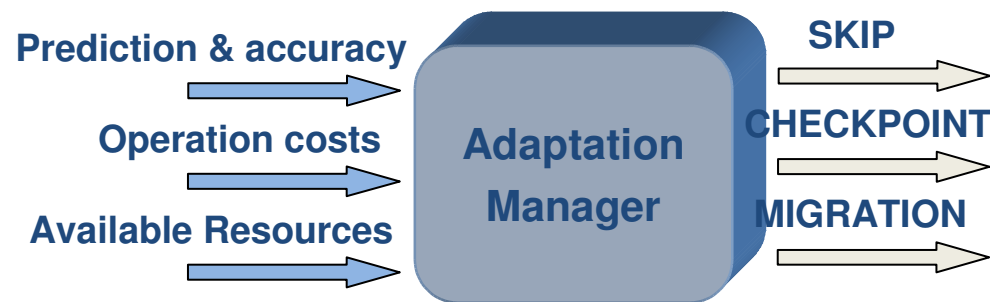
$$\text{where } f_{appl} = \begin{cases} 1 - \prod_{i=1}^{N_W^f - N_S^h} f_p & \text{if } N_W^f > N_S^h \\ 0 & \text{if } N_W^f \leq N_S^h \end{cases}$$

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

- **SKIP:** $\text{where } f_{appl} = 1 - \prod_{i=1}^{N_W^f} f_p$

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

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





Rescheduling & Preemptive Rescheduling

- Trigger job rescheduling & preemptive rescheduling
- Transform into a general 0-1 Knapsack model
 - To determine a binary vector $X = \{x_i \mid 1 \leq i \leq J_s\}$ such that
 - maximize $\sum_{1 \leq i \leq J_s} x_i \cdot v_i$, $x_i = 0$ or 1
 - s.t. $\sum_{1 \leq i \leq J_s} x_i \cdot p_i^s \leq S$
- 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 driven*, to minimize the slowdown caused by failures



System Support

- Development /optimization of fault tolerance techniques
 - Dynamic virtual machine
 - Live migration support
 - Fast fault recovery
- Fault management strategy
 - Customized (virtual) computing environment
 - System incarnation
 - System configuration (number of spare nodes)
 - Scheduling, rescheduling, preemptive rescheduling
 - Rescheduling trigger system, destination determination



Virtual Machine & Dynamic Virtual Machine

- Why VM?
 - A customized environment
 - Security and isolation
 - Several OS's on a single physical machine
 - Finer resource control
 - Checkpoint and migration
- Why dynamic?
 - Flexibility (dynamic in configuration, customization)
 - Different users may have different requirements
 - Adaptation (dynamic in configuration)
 - Frequent deployment and redeployment to different platforms
 - Mobility (dynamic in location)
 - Dynamic resource availability
 - Reliability and system management



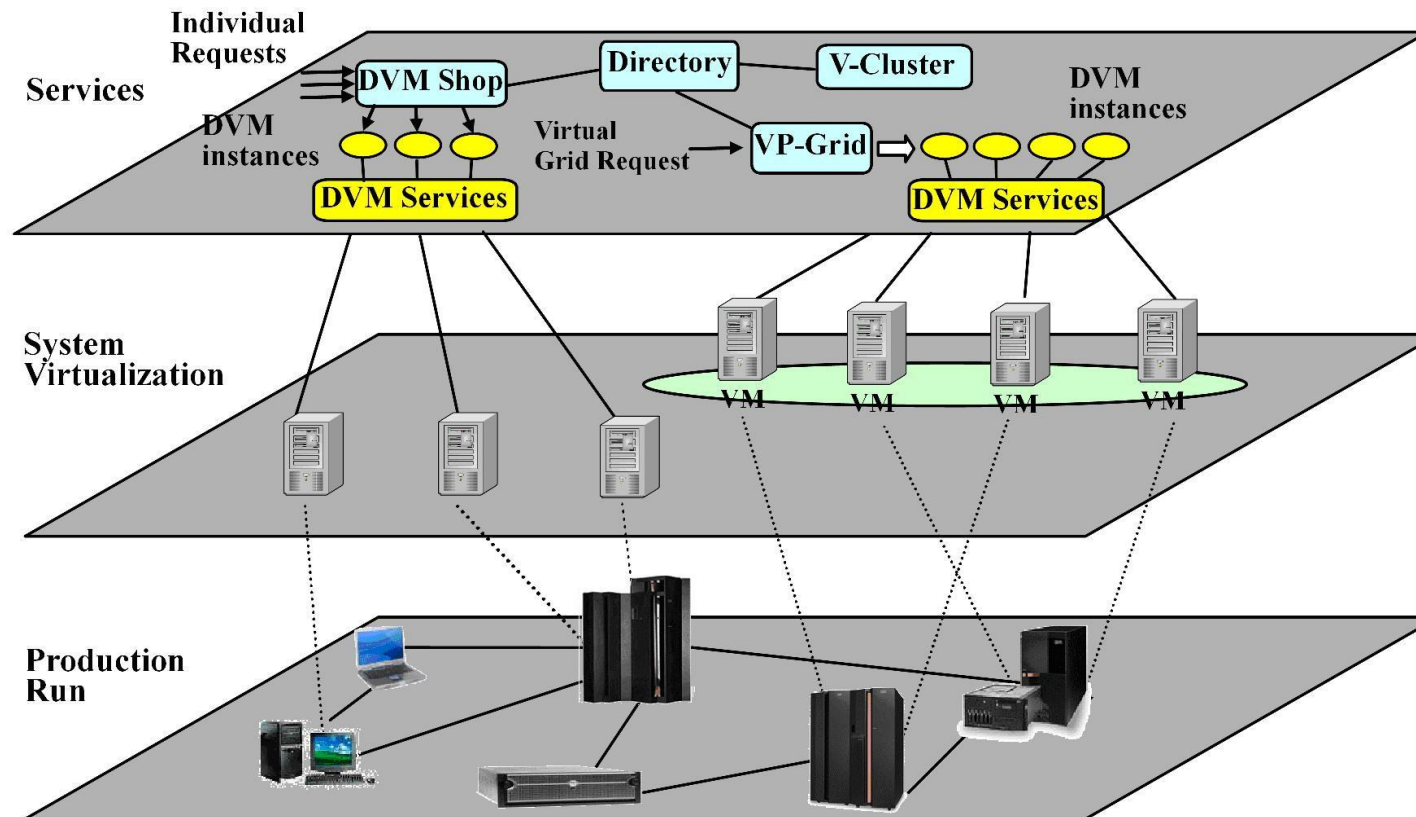
Dynamic Virtual Machine System

- Dynamic Virtual Machine as an abstract of a virtual machine that can be ***instantiated and migrated dynamically***.
- DVM system as a middleware that encapsulates computational resources in a ***secure, isolated*** and ***customized*** VM environment, ***manages*** the lifetime of one, or more, VM dynamically, and enables ***transparent service mobility*** and ***service provisioning***.



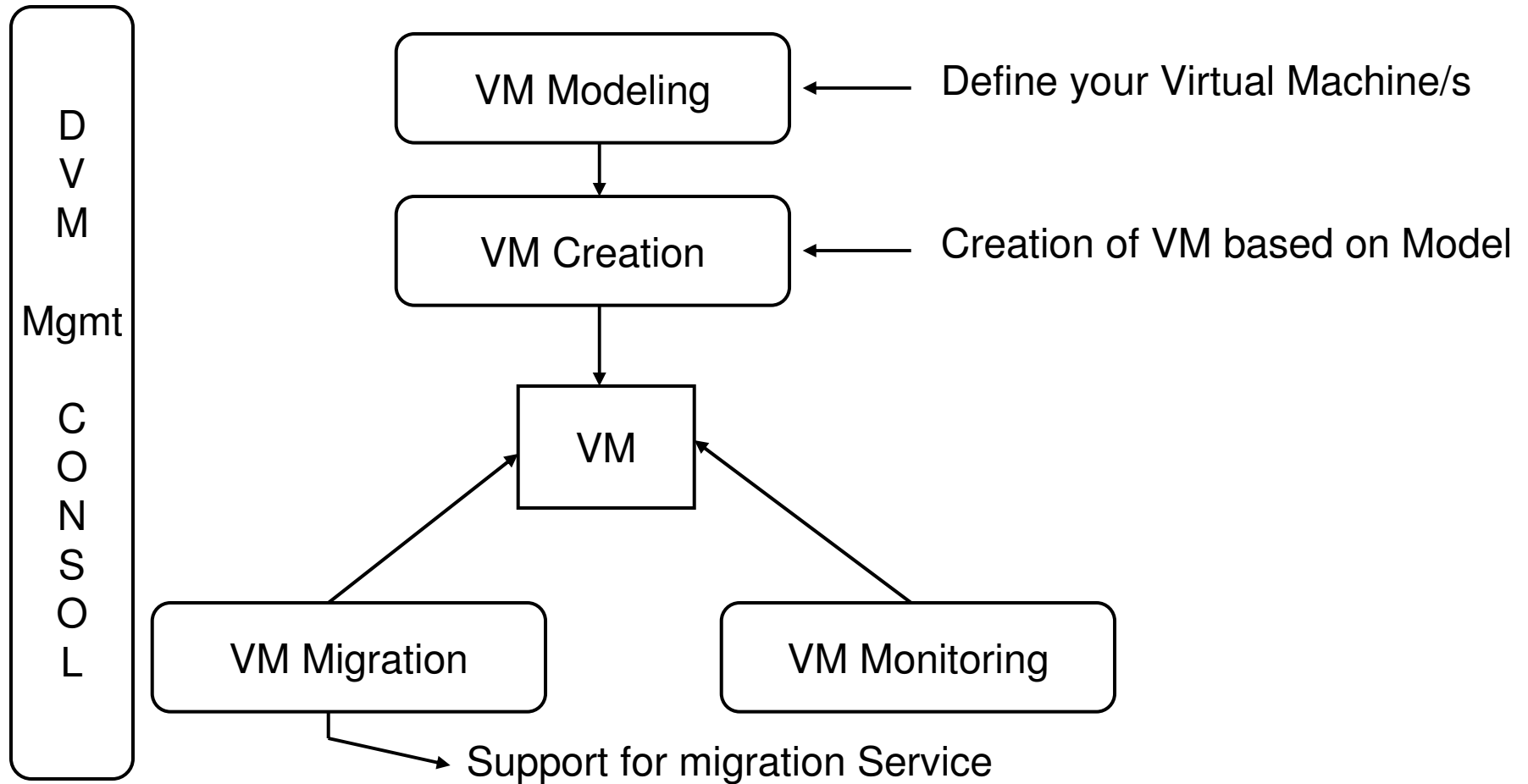
Dynamic Virtual Machine System

(Virtual cluster, virtual working space)





Key Components of a DVM System





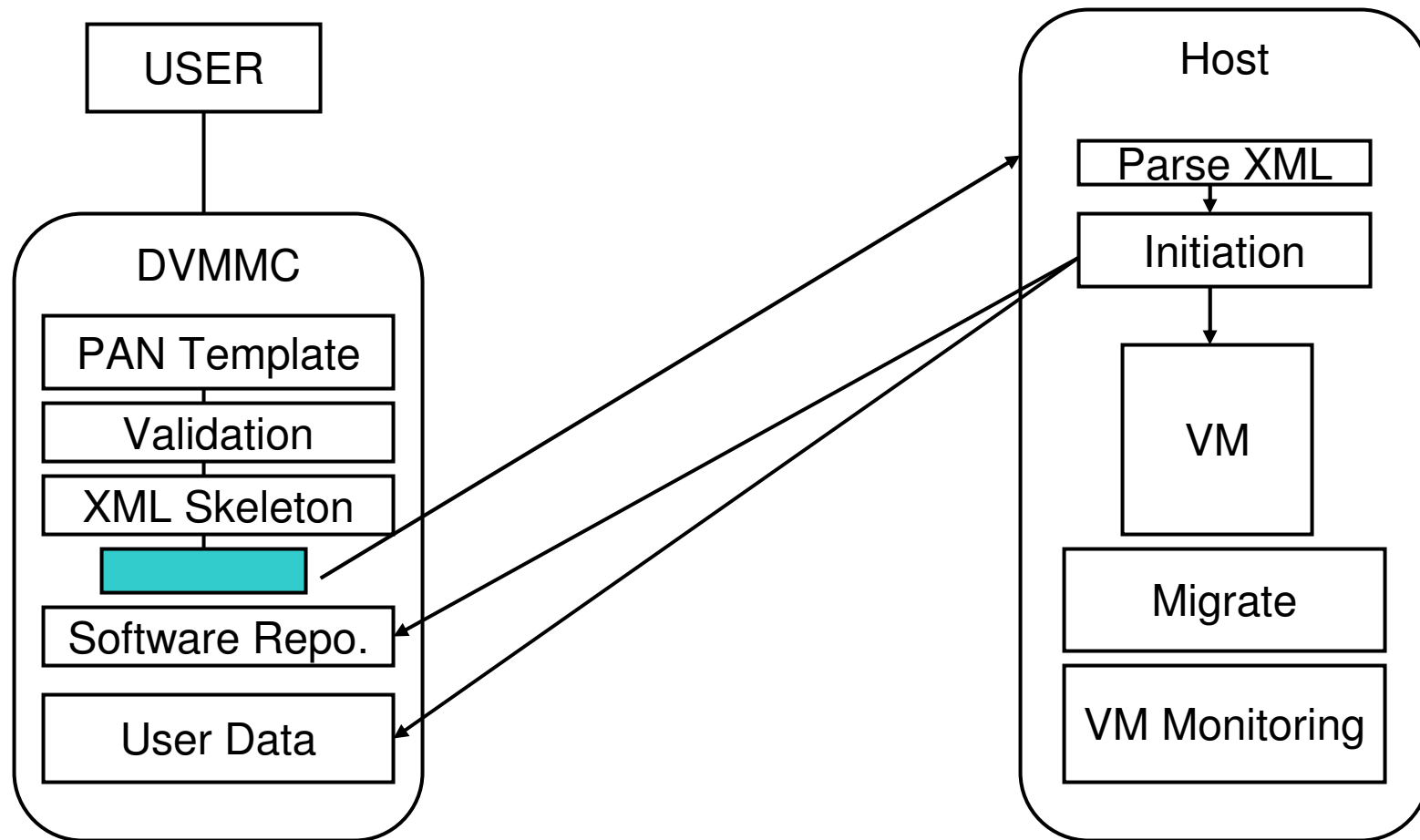
VMD: VM Description (Script)

- Script with modeling information. Models Include:
- Network Model : Includes Host name, Domain name, IP address and MAC address of each interface.
- System Model : Includes flavor and version of operating system, hardware mapping and logical volumes.
- Software Model : Includes list of software requested and their configuration information
- Storage Model : Includes location of user applications and data, Environment settings and variables etc.

Could generated when a VM is generated
Does not need to be complete



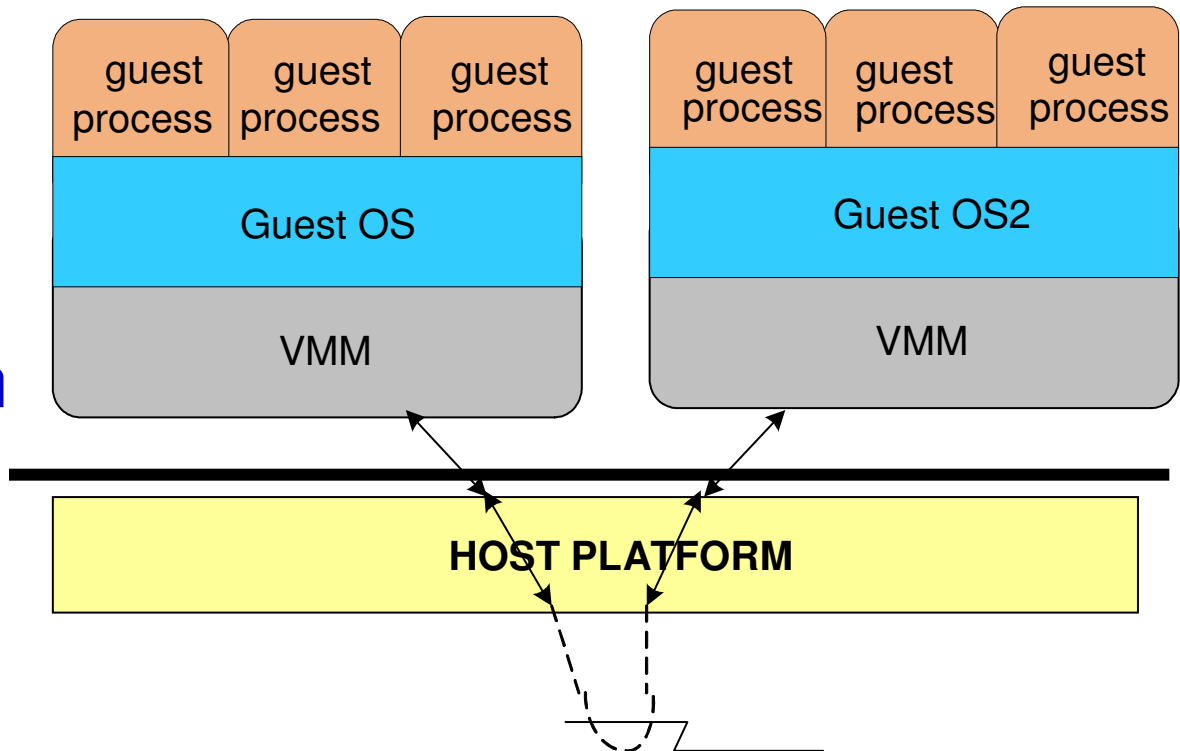
The Working of DVM system





DVM in Fault Tolerance

- Most faults are software fault
- A VM or Virtual Cluster is customized for each application
- Can be re-incarnated swiftly from the model
- Support VM migration



Virtual network communication



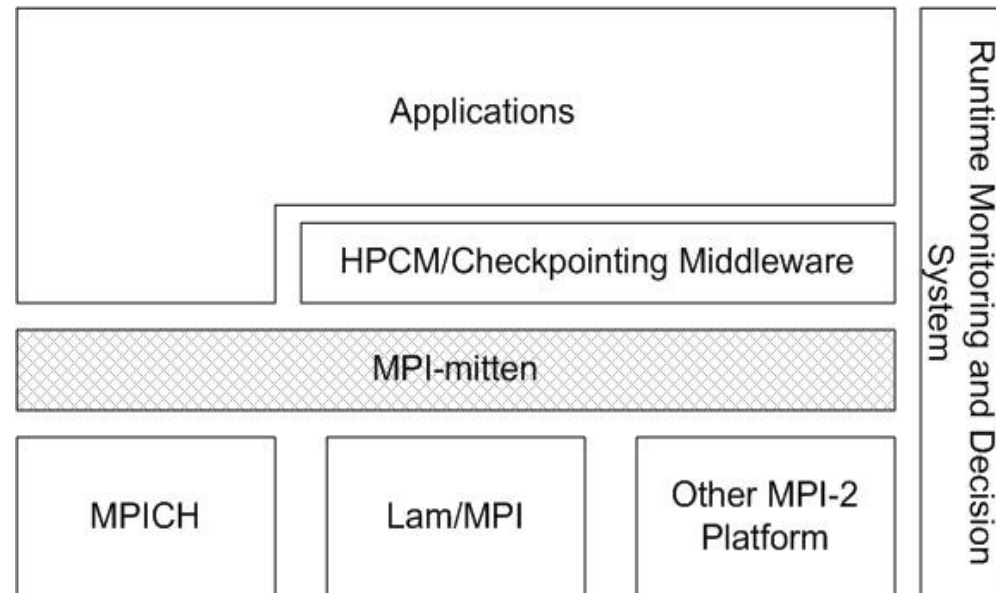
Communication and Group Membership

- With VM, process migration is easy
- But, how to handle communication, especially group communication
- A set of protocols has been developed to handle one-to-one and group communication during and after process/VM migration
- A software system, named MPI-Mitten, is developed to support group membership management for MPI applications.
- No rollback, no stopping, new group membership management protocol
- The performance optimization for hardware and communication channels is preserved.



MPI-mitten: the software system

- A portable communication library
- High-level add-on layer to various existing implementations of MPI
 - LAM/MPI and MPICH2
- Support dynamic group membership management
- Support communication state transfer
- Optimize application level VM mobility





Experimental Testing

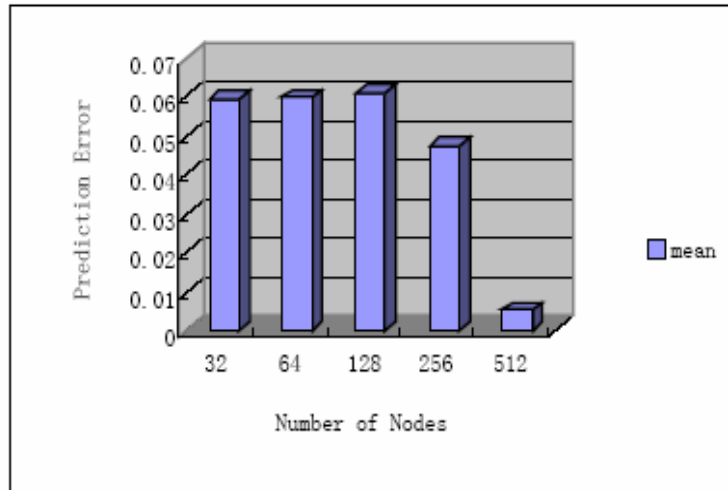
Long Term: Experimental Environment

- Experiments conducted on the failure trace collected at the Los Alamos National Lab
- The data spans 22 high-performance computing systems, these systems have been in production at LANL between 1996 and November 2005.
- In total the systems include 4750 nodes and 24101 processors.

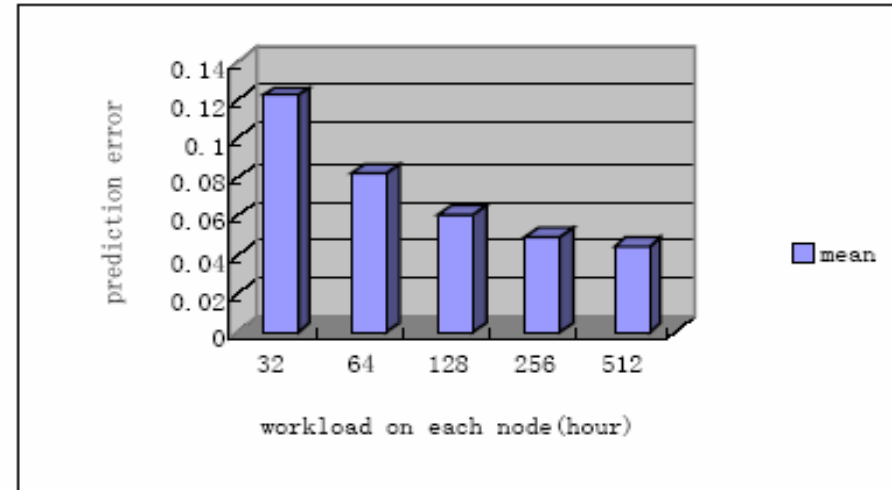


Prediction Results

- LANL failure trace
- Storage cost $\alpha = 0.2$ hour.
- Checkpointing period = 2 hours



Prediction error with different machine number



Prediction error with different workloads



Scheduling Results

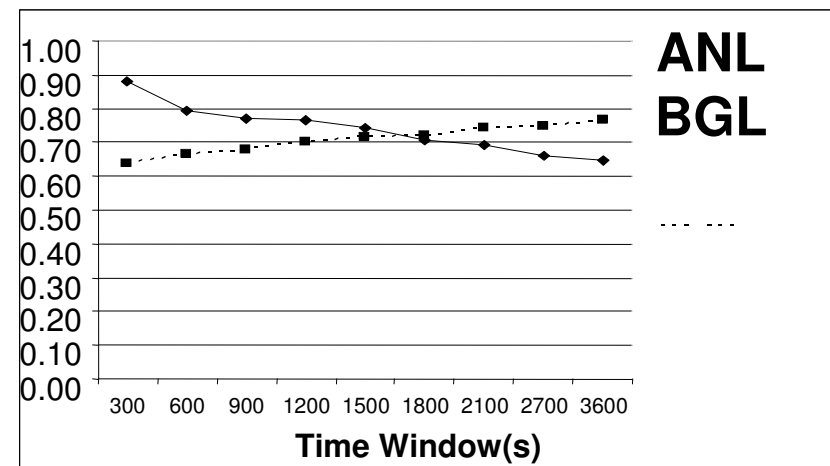
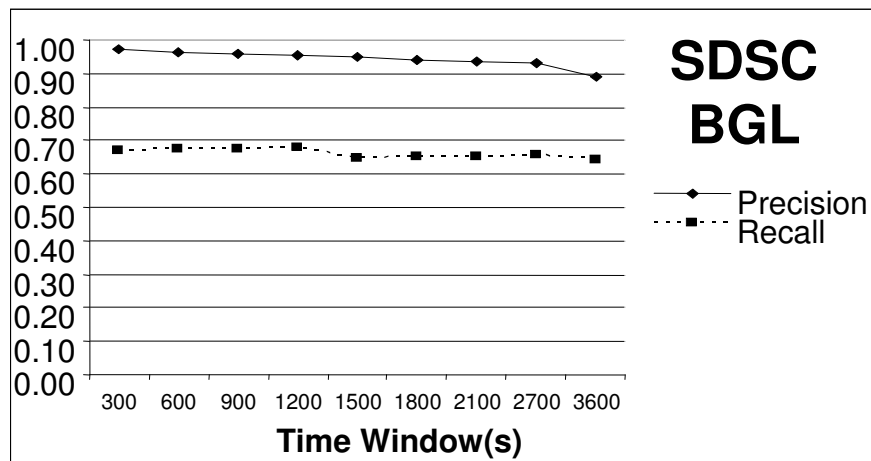
Execution time with different scheduling strategies

Scheduling Methodology	Application Execution Time					
	100 nodes			1000 nodes		
	10	20	40	100	200	400
Fault-aware	314.7	207.857	168.438	370.2	256.0	228.5
Random	612.0	351.655	221.357	695.4	451.9	372.3
Speed-only	362.5	239.473	182.814	486.7	348.8	287.1
Reliability-only	472.2	277.202	185.127	545.7	339.6	262.4
Speed-reliability	449.0	275.652	184.358	524.2	332.8	259.5



Short Term: Prediction Results

	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



- 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.

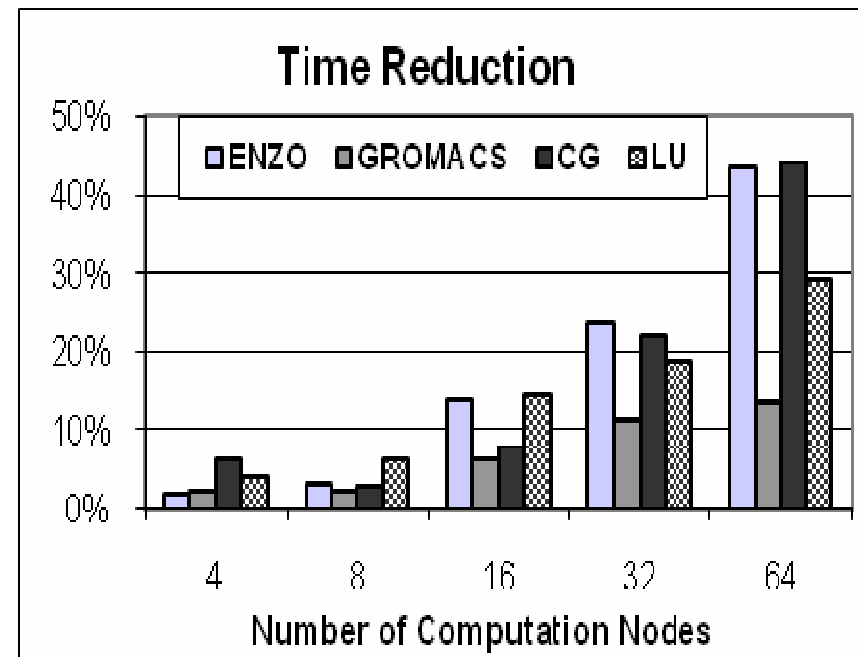


Adaptation Results

- Case studies
 - Implemented with MPICH-VCL
 - Test applications: ENZO, Gromacs, NPB
 - Failure trace is from a NCSA Linux cluster
 - Platform: TeraGrid/ANL IA32 Linux Cluster

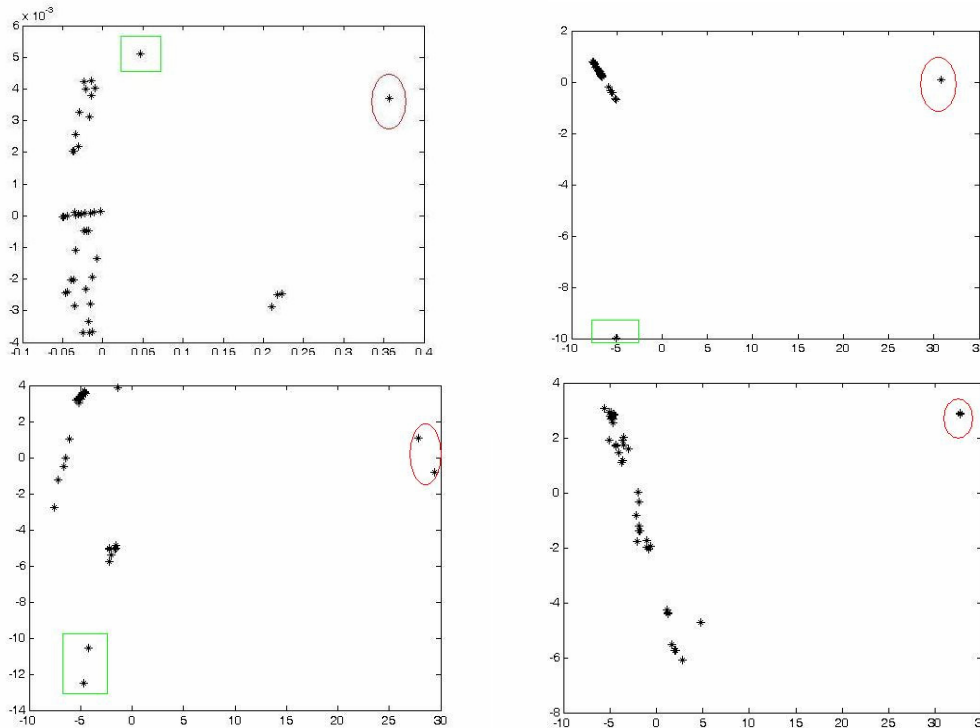
- Results:

- Outperforms periodic checkpointing as long as recall and precision are higher than 0.30
- A modest allocation of spare nodes (i.e. <5%) is sufficient
- Lower than 3% overhead





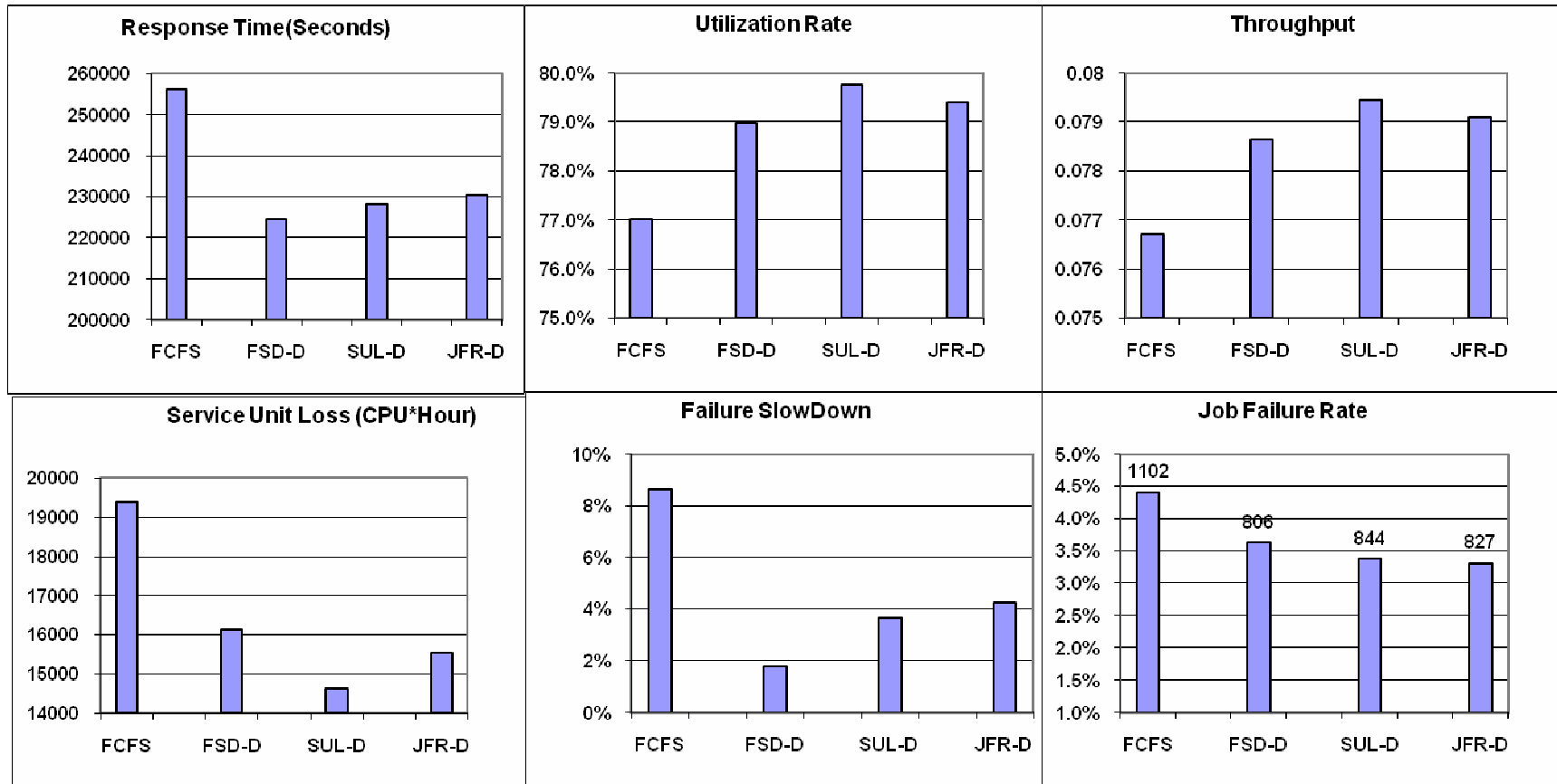
Localization Results



- The ones in the ELLIPSE are correct predictions, and the ones in the RECTANGLE are false alarms
- For 256-node systems, the location method took less than 1.0 second

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

Preemptive Rescheduling Results



Positive improvement on system productivity, if the failure predictor can capture 20% of failure events with a false alarm rate lower than 80%



Experiments: VM Migration Comparisons

		Suspension & Resume	Suspension & Resume (compressed)	Shared Storage	DVM
linpack	Response (secs)	449.2	431.8	38.25	198.5
	OOS (secs)	449.2	431.8	6.4	1.99
	Comm (M)	4698	570.6	382.25	0.359
	Perf degradation (%)	0	0	23.5%	0
bitonic	Response (secs)	449.9	432.9	29.3	198.6
	OOS (secs)	449.9	432.9	0.6	0.03
	Comm (M)	4698	566.1	275.73	0.011
	Perf degradation (%)	0	0	20.8%	0
test_tree	Response (secs)	448.9	430.9	39.21	198.4
	OOS (s)	448.9	430.9	6.48	6.38
	Comm (M)	4698	565.0	382.25	2.08
	Perf degradation (%)	0	0	25.6%	0
gzip	Response (secs)	449.1	438.9	67.3	201.1
	OOS (secs)	449.1	438.9	4.78	0.10
	Comm (M)	4698	644.65	664.19	0.055
	Perf degradation (%)	0	0	28.1%	0

Compared with Xen

DVM: Average size of skeleton = 5KB

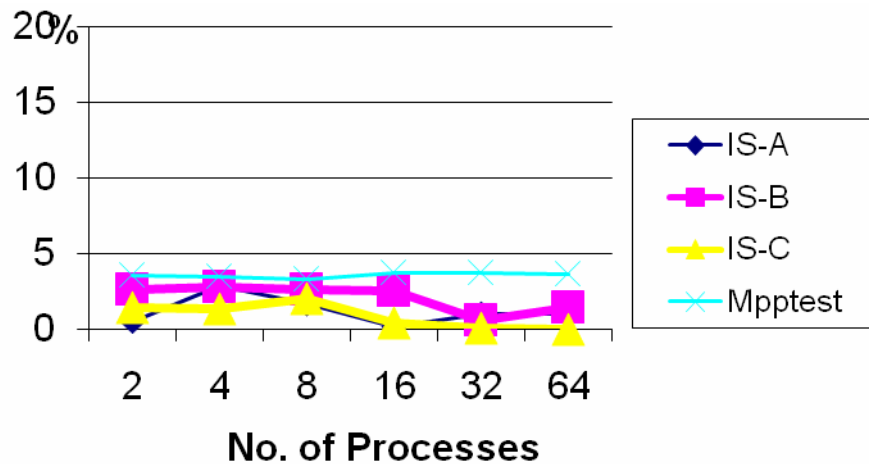
Average regeneration from skeleton = 200 seconds



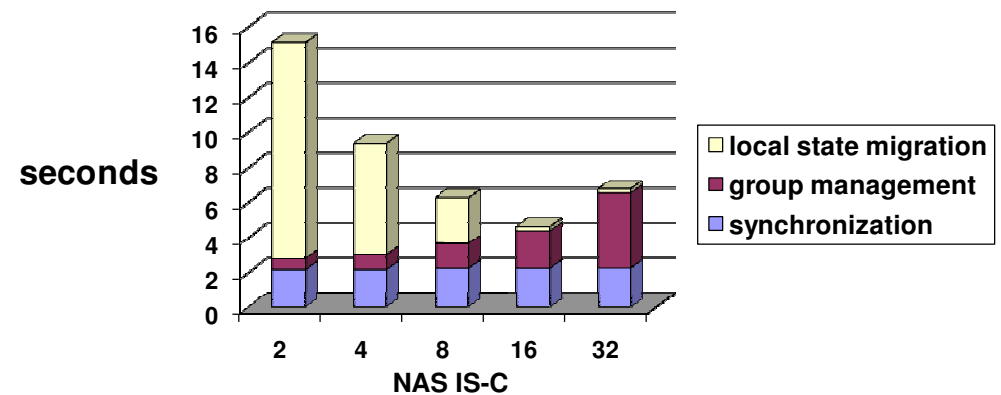
Experiment: MPI Mitten

Preliminary results with NAS Parallel Benchmarks and mpptest

– Less than 4% overhead



Overhead



Migration Time



Conclusions

- FENCE to advance fault management
- A comprehensive approach
 - Long term modeling: system configuration and task scheduling
 - Short term adaptation: adaptation and rescheduling
 - System support : DVM and MPI Mitten
- Has shown great potential
- Many issues remain open
 - Proof of concept implementation
 - System is only available piece by piece



Questions?



SCS Lab Website:

<http://www.cs.iit.edu/~scs>



**To our sponsor: National Science
Foundation**