SELF-CARING IT SYSTEMS: A PROOF-OF-CONCEPT IMPLEMENTATION IN VIRTUALIZED ENVIRONMENTS

Selvi Kadirvel and José A. B. Fortes
Outline

- Motivation
- Goals
- Problem Scope
- Solution Overview
  - Self-Caring IT systems
  - Health management components
  - Overview of approach
- Solution
  - Goal 1: Modeling methodology and framework
  - Goal 2: Remaining-Useful-Life management
- Proof-of-concept implementation
  - Experimental Results
- Summary
Motivation

- Dependence on Information Technology (IT) services is common to all domains
- Prevalence and cost of failures
- Increased likelihood of failures: Scaling up, heterogeneity, complexity, geographical distribution of IT systems

- The BP Oil Spill: Could Software be a Culprit?
  - Mishandled Software Alarms
- Cloud-based IT failure halts Virgin flights
- Microsoft Zune's New Year Crash
- Google App Engine Hit By Outage
- **Cloud Goes Boom, T-Mo Sidekick Users Lose All Data**
- Apple, AT&T data leak protection issues latest in cloud failures
Motivation – Current Literature

- Reliability and fault tolerance
  - Redundancy in time, space and information
  - Checkpoint/Recovery
  - Reactive in nature
- Some Proactive approaches
  - Component or system specific
  - Examples:
    - Hard disk failures – SMART (Self-Monitoring Analysis and Reporting Technology)
    - IBM BlueGene - RAS (Reliability, Availability, Service) logs
Goals

• Goal 1: Generic, systematic approach to design and develop IT systems that are aware of their health state and can manage this health

• Goal 2: Proactively handle health deteriorations
Goals

• Goal 1: Generic, systematic approach to design and develop IT systems that are aware of their health state and can manage this health
  • Define self-caring IT systems
  • Use a modeling framework to simulate and control IT systems

• Goal 2: Proactively handle health deterioration
  • Feedback controllers to observe system health, extend useful life and invoke recovery/remedies
Problem Scope

• Type of environment: Virtualized environment
  • Basic component in increasingly popular paradigms - clouds, server consolidation, high performance computing
  • Powerful paradigm – Control, customization

• Type of faults
  • Resource Exhaustion faults
  • Quite common as can be seen in US Government’s National Vulnerability database
  • Observed in all types of software - Web servers, DNS servers, operating systems
Problem Scope: Resource Exhaustion Faults

- **Resource** – Any type of entity that is *consumed* and is available in *finite* supply

- **Causes include**
  - Improperly executing software
  - Unanticipated workloads
  - Malicious code invocations and intrusions
  - Software aging
  - Hardware faults

- **Examples**
  - Memory leak over time leads to memory exhaustion
  - File descriptors, socket descriptors not managed well
  - Abandoned processes, threads
Applications

- Clouds
  - Resources – CPU, Memory, Storage
  - Example: Google App Engine

\[\text{When a Resource is Depleted}\]

When an app consumes all of an allocated resource, the resource becomes unavailable until the quota is replenished. When the resource is depleted, App Engine by default returns an HTTP 403 Forbidden status code.

**Per-minute Quotas**

In addition to the daily quotas described above, App Engine moderates how quickly an app can consume a resource, using per-minute quotas.

- Data Store API calls, Memcache API calls, Task queue API calls

- High Performance Computing
  - Resource limits
  - PBS or Torque directives

\[
\text{#PBS -l walltime=01:00:00} \quad \# \text{Specify the walltime} \\
\text{#PBS -l pmem=100mb} \quad \# \text{Memory Allocation for the Job} \\
\text{#PBS -l nodes=4} \quad \# \text{Number of nodes to Allocate}
\]

- Job simply aborted
- Shared infrastructures – ensure fairness
Solution:
A. Self-Caring IT systems
B. Health management components
C. Overview of approach
Self-Caring IT Systems

- IT Systems
  - Aware of health and *proactively* manage health deteriorations in addition to *reactively* responding to failures
  - Complement to Self-Healing IT Systems
- Capability to observe trends in health deterioration and managing them - “Health management”
- Benefits
  - Scope of damage
  - Choice in remedies
  - Avoid faults
  - Less expensive
Health Management Components

- Includes
  - Monitoring & Detection
  - Diagnosis
  - Prognosis
  - Remaining-Useful-Life extension
  - Planning
  - Remediation
Overview of Approach
Overview of Approach

System Model/Global Manager

Portal Servers

User

Head Node

Network

Storage

Application

Middleware

Compute Nodes
Overview of Approach

PORTAL SERVERS

HEAD NODE

APPLICATION

MIDDLEWARE

COMPUTE NODES

USER

SYSTEM MODEL/
GLOBAL MANAGER

HEALTH MANAGEMENT
MODULES

RUL MANAGER

RUL MANAGER

RUL MANAGER

RUL MANAGER

DIAGNOSIS

PROGNOSIS

PLANNING

REMEDIES

DIAGNOSIS

PROGNOSIS

PLANNING

REMEDIES

DIAGNOSIS

PROGNOSIS

PLANNING

REMEDIES

DIAGNOSIS

PROGNOSIS

PLANNING

REMEDIES

ACIS Advanced Computing and Information Systems Laboratory
Goal 1: Modeling Framework
Selection of modeling tool

- Model type: Discrete Event Systems (DES)
  - Events determine state changes, rather than time
  - Capture dependencies, ordering of events and activities
  - Supports concurrency, asynchrony
- Petri nets: A graphical DES model
  - Rich theory with many extensions
  - Analysis – Verify system properties
  - Simulation – Effects in production systems
  - Execution – Build a system manager
- Alternatives
  - Finite State Machines, Formal languages (LOTOS, CSP), UML
- Uses: Computer networks, Process control plants
Modeling Methodology

• Progressive construction of Petri net model capturing functionality and health management

• Sample mapping:
  • Activities and resources → Places
  • Events → Transitions
  • Order/dependency → Arcs

**System model augmented with health management**
Goal 2: Extending Remaining-Useful-Life
Remaining-Useful-Life Extension

- An estimation of time after which there is a high probability that component will fail
- Different from MTTF Eg: bulb with MTTF = 10K hours
- Factors that determine RUL: workload, environmental interactions, configuration parameters, component faults
- Techniques: statistical, machine learning approaches
- Importance:
  - Insufficient useful life may prevent recovery action
  - Example:
    - Time to migrate VM
    - Time to start up a new server
Feedback controller

- Apply feedback control theory
- System modeling
  - Identify input and output variables
  - Determine relationship
  - Linear first order model approximation works
- Controller design
  - Modulate system input parameters (resource allocation) to control health metrics (performance)
  - Use a feedback loop to converge to the acceptable depletion rate
Proof-of-Concept Implementation
Batch-based Job Submission in HPC

- Sequence of activities and dependent resources:
  - **Job Creation**
    - Portal Server
    - Database Server
    - Storage Server
  - **Job Transfer**
    - Portal Server
    - Head Node
    - HPC Storage Server
  - **Job Queued**
    - Head Node
    - Resource Manager
    - Job Scheduler
  - **Job Execution**
    - Compute Node
    - HPC Storage Server
  - **Results Transfer**
    - Portal Server
    - Head Node
    - Storage Server

- Virtual Cluster Test bed
  - Platform - VMware ESX servers
  - Middleware - Torque Resource Manager, Maui Job Scheduler, MySQL database backend
  - Application – Sequence of Matrix multiplication operations
  - VMware Perl API
(1) Results – Petri Net Model of System

- IT system mapped to Petri net model
- Designed and constructed using PIPE-2 tool (Imperial College, London)
(2) Results – Analysis and Simulation using Model

• Analysis
  • Ensure addition of health management does not violate system properties.
  • Structure captures semantics – Deadlock free, bounded

• Simulation
  • Set request arrival rates, queue sizes, resource levels
  • Help identify thresholds for anomalous resource consumption
  • Other uses: Identify Bottlenecks
(3) Results – Petri Net Model as Global Manager

- Represent model structure and functionality in XML, Java
- Generic Petri Net execution engine
  - Manage job submission and execution to a cluster of virtual machines
(4) Results - RUL Manager for Job Execution

Application processing a stream of requests, fault injection – memory leak

Step 1. Detection - Health deterioration through threshold, trend, event alarms

Step 2. Diagnosis

Step 3. Prognosis/Useful Life Extension

- Desired Useful Life = s
- Resource depletion takes place at rate X
- Throttle workload to change depletion to rate Y
(5) Results – RUL manager design

- Proportional-Integral Controller

- Pole placement design for initial controller gain (P, I) values
- Empirical tuning
- SASO properties:
  - Stability, Maximum Accuracy, Minimum Settling Time, Minimum Overshoot
(6) Results – Remediation

- Step 4: Planning and Remediation
  - Feedback controller designed to gain useful life time
  - Gained useful life “s” is then used to invoke remediation

<table>
<thead>
<tr>
<th>REMEDIATION</th>
<th>CAUSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rejuvenation</td>
<td>Software Aging</td>
</tr>
<tr>
<td>Dynamic Resource</td>
<td>Unanticipated load, External</td>
</tr>
<tr>
<td>Increase</td>
<td>interference</td>
</tr>
<tr>
<td>VM Migration</td>
<td>Hardware fault, External</td>
</tr>
<tr>
<td></td>
<td>disturbance</td>
</tr>
</tbody>
</table>
Summary and Conclusions

• Systematic Approach to Self-Caring IT systems:
  • Identify a suitable modeling tool and defined the methodology
  • Construct a Model-based system manager

• Proactive handling of health deteriorations
  • Design, develop and deploy feedback controller for RUL extension of the application execution
  • No application changes, no Operating System changes, Only augmentation to middleware
Ongoing Work

- **Online Control**
  - Auto-tuning and self-tuning based controllers to accommodate both new systems and changing system operation
- **Directly estimate useful life through the use of machine learning approaches.**
- **Multiple resources**
  - Capture correlation between multiple resources using Multiple-Input-Multiple-Output (MIMO) modeling of target system
Thanks!
(B) Results – RUL extension

- Further instrumentation to observe RUL extension
- “Functional state” determined by level of resource consumption
- Duration of time spent in different states:

![Diagram showing duration of time spent in different states](image)

- Was RUL extension sufficient to avoid failure?
  - Desired RUL is uniformly distributed between 10 to 600 seconds
  - 82% of the cases – remediation was possible seamlessly