Cloud Computing

Lecture 22
Autonomic Computing
2011-2012

Growing Complexity

• Systems are getting more and more complex:
  • Heterogeneity and device interconnection and dependency are constantly growing. It is harder and harder to predict all interactions and their consequences.
  • There is pressure for late and dynamic decisions: e.g. plug-ins, virtual machines, JIT compilers, service discovery, mobile agents, Reflection...
• Complexity management has growing costs:
  • Human interventions and IT costs.
Large Scale Computing: Growing Complexity

- **System Uncertainty:**
  - Very large scales.
  - Ad hoc structures/behaviors:
    - p2p, hierarchical, etc, architectures.
  - Dynamic:
    - entities join, leave, move, change behavior.
  - Heterogeneous:
    - capability, connectivity, reliability, guarantees, QoS.
  - Lack of guarantees:
    - components, communication.
  - Lack of common/complete knowledge (LOCK):
    - number, type, location, availability, connectivity, protocols, semantics, etc.

- **Information Uncertainty:**
  - Availability, resolution, quality of information.
  - Trust in data, data models.
  - Semantics.

- **Application Uncertainty:**
  - Dynamic behaviors:
    - space-time adaptivity.
  - Dynamic and complex couplings:
    - multi-physics, multi-model, multi-resolution, ....
  - Dynamic and complex (ad hoc, opportunistic) interactions.
  - Software/systems engineering issues:
    - Emergent rather than by design.

Complexity Management

- But complexity is beyond that human can handle:
  - **Human out of the control loop => autonomic.**

- Even though we are moving along this direction, is there any systematic way of addressing this issue?
  - **Autonomic Computing.**
Integrating Biology and Information Technology: The Autonomic Computing Metaphor

- Current programming paradigms, methods, management tools are inadequate to handle the scale, complexity, dynamism and heterogeneity of emerging systems.
- Nature has evolved to cope with scale, complexity, heterogeneity, dynamism and unpredictability, lack of guarantees:
  - self configuring, self adapting, self optimizing, self healing, self protecting, highly decentralized, heterogeneous architectures that work !!!
- Goal of autonomic computing is to build a self-managing system that addresses these challenges using high level guidance:
  - Unlike AI duplication of human thought is not the ultimate goal!

Adaptive Biological Systems

- The body’s internal mechanisms continuously work together to maintain essential variables within physiological limits that define the viability zone.
- Two important observations:
  - The goal of the adaptive behavior is directly linked with the survivability.
  - If the external or internal environment pushes the system outside its physiological equilibrium state the system will always work towards coming back to the original equilibrium state.
Self-Adaptive Software

- “...software that evaluates its own performance and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do...” (Laddaga, 1997)
- To adapt, the system reacts to environmental change - the problem is recognizing the need for change, then planning, enacting and verifying the change.
- Progress to date has been informed by three guiding metaphors:
  - control systems theory.
  - dynamic planning systems.
  - self-aware or reflective systems.
- “Managing complexity is a key goal of self-adaptive software. [...] Somehow we need to be able to write software that is less complex than the environment in which it is operating yet operate robustly.” (Robertson, Laddaga et al, 2000).

Relevance for Cloud Computing

- Cloud computing systems have many components, very varied and many participants. The cloud is an ecosystem.
Characteristics of an Autonomic System

- In the original IBM proposal ("An architectural blueprint for autonomic computing", a de facto standard):
  - Self-Configuration
  - Self-Optimization
  - Self-Repair
  - Self-Protection

Self-* or Self-X Properties

Self-Configuration

- Adapt automatically to the dynamically changing environment
- Internal adaptation
  - Add/remove new components (software)
  - configures itself on the fly
- External adaptation
  Systems configure themselves into a global infrastructure
Self-Repair

- Discover, diagnose and react to disruptions without disrupting the service environment

- Fault components should be
  - detected
  - isolated
  - fixed
  - reintegrated

Self-Optimization

- Monitor and tune resources automatically
  - Support operating in unpredictable environment
  - Efficient maximization of resource utilization without human intervention

- Dynamic resource allocation and workload management
  - Resource: Storage, databases, networks
  - For example, Dynamic server clustering
## Self-Protection

- Anticipate, detect, identify and protect against attacks from anywhere
  - Defining and managing user access to all computing resources
  - Protecting against unauthorized resource access, e.g. SSL
  - Detecting intrusions and reporting as they occur

## Autonomy Levels

<table>
<thead>
<tr>
<th>Benefits</th>
<th>Basic Requirements Met</th>
<th>Managed Level 2</th>
<th>Predictive Level 3</th>
<th>Adaptive Level 4</th>
<th>Autonomic Level 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Characteristics</td>
<td>Multiple sources of system generated data</td>
<td>System monitors, correlates and recommends actions</td>
<td>IT staff analyzes and takes actions</td>
<td>IT staff approves and initiates actions</td>
<td>Integrated components dynamically managed by business rules/policies</td>
</tr>
<tr>
<td>Skills</td>
<td>Requires extensive, highly skilled IT staff</td>
<td>IT staff manages system performance against SLAs</td>
<td>IT staff manages system performance against SLAs</td>
<td>IT staff focuses on enabling business needs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Manual</td>
<td>Autonomic</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Basic Level 1
- Managed Level 2
- Predictive Level 3
- Adaptive Level 4
- Autonomic Level 5
How are systems going to change?

<table>
<thead>
<tr>
<th></th>
<th>Today</th>
<th>The Autonomic Future</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-configure</td>
<td>Corporate data centers are multi-vendor, multi-platform. Installing, configuring, integrating systems is time-consuming, error-prone.</td>
<td>Automated configuration of components, systems according to high-level policies; rest of system adjusts automatically. Seamless, like adding new cell to body or new individual to population.</td>
</tr>
<tr>
<td>Self-heal</td>
<td>Problem determination in large, complex systems can take a team of programmers weeks</td>
<td>Automated detection, diagnosis, and repair of localized software/hardware problems.</td>
</tr>
<tr>
<td>Self-optimize</td>
<td>WebSphere, DB2 have hundreds of nonlinear tuning parameters; many new ones with each release.</td>
<td>Components and systems will continually seek opportunities to improve their own performance and efficiency based on workloads, history and prediction.</td>
</tr>
<tr>
<td>Self-protect</td>
<td>Manual detection and recovery from attacks and cascading failures.</td>
<td>Automated defense against malicious attacks or cascading failures; use early warning to anticipate and prevent system-wide failures.</td>
</tr>
</tbody>
</table>

Autonomic Elements: Structure

- Fundamental atom of the architecture:
  - Managed element(s):
    - Database, storage system, server, software app, etc.
  - Plus one autonomic manager.
  - Responsible for:
    - Providing its service.
    - Managing its own behavior in accordance with policies.
    - Interacting with other autonomic elements.
Alerts, events & problem analysis request interface

SLA/Policy interface, interprets & translates into "control logic"

Autonomic Manager Substructure

Sensors

Analyzer

Plan

Policy Interpreters

Policy Transforms

Plan Generators

Knowledge

Workflow Engine

Service Dispatcher

Scheduler Engine

Distribution Engine

Effectors

Monitor

Analysis Engines

Policy Validations

Policy Resolution

Rules Engines

Filter

Simple Correlators

Metric Managers

Recent Activity Log

Topology

Calendar

Metric Managers

Sensors

Effectors

Rules Engines

Analysis Engines

Policy Validations

Policy Resolution

Rules Engines

Filter

Simple Correlators

Metric Managers

Recent Activity Log

Topology

Calendar

Monitoring

• Capture environment properties (physical or virtual) that are relevant for decision.
• Highly implementation dependent component.
• Sensors read properties from the managed component: requests per second, power consumption,…
• Passive monitoring (no changes to system): e.g. /proc folder in Unix.
• Active monitoring (with changes to the system):
  • e.g. ProbeMeister, Pin (code injection for monitoring).
  • May require adaptive monitoring so as not to influence performance.
Analysis

- Combine monitoring symptoms into higher level descriptions.
- Process event streams according to policies:
  - Which sequences are to be interpreted?
  - Which sequences are discarded?
- May include the ECA (event-condition-action) component.
- Borders between Monitoring/Analysis/Planning are not rigid.

Planning

- Based on monitoring data decide changes to apply to the managed element.
- Two approaches:
  - ECA: a set of stateless rules to decide what actions to take. Often require conflict resolution mechanisms.
  - Model based approach: represent the managed element in a model. Actions are apply first on the model in order to detect problems and inconsistencies.
Execution

- Execution of the planning decisions.
- Based on dependencies between actions and intervention opportunities:
  - Convert the plan into a workflow.
  - Schedule executable actions.
  - Execute local and eventual remote actions.

MAPE-K in Large-Scale

- Human supervision
- Self-* and system aggregation
- Local self-* managers (replication, reports)
- Elementary MAPE-K control
AC: Main applications

• Data center energy management:
  • CPU throttling.
  • Detailed node monitoring (at component model).
  • CPU/cooling interactions.
• Resource allocation based on the SLA (and its variants):
  • e.g. Nimrod/G: cluster/grid scheduling based on an economic utility model and system monitoring.

Example: Processing Allocation in Cloud

• How to distribute and adapt job execution in clouds? (Paton, 2009)
• Consider:
  • Job characteristics.
  • SLA.
  • Resource competition.
• Tool: utility function.
Development Steps

- Selection of utility properties.
- Definition of the utility function.
- Development of cost model.
- Representation of the system state.
- Selection of the optimization algorithm.
- Implementation of the control cycle.

Implementation for Queries over Clouds

- Utility property: response time and % of met SLA goals (QoS).
- Utility function 1 (response time):
  \[ Utility_{q}^{RT}(w, dp) = \left(1/\sum_{q \in w} PRT_q(q, dp(q))\right) \]
  - PRT is the query’s predicted duration.
  - w is a workflow of queries.
  - dp is the distribution of queries supplied by the nodes.
Implementation for Queries over Clouds (2)

• Utility function 2 (QoS):

\[ Utility_{QoS}^q(w, a) = \sum_{q \in u} QoSEstimate(q, dp(q)) \]

• QoS is the probability of the query finishing on time: PRT < SLA.

• Development of the cost model:

  • Create PRT functions based on query characteristics, examples or on the progress of current queries.

Implementation for Queries over Clouds (3)

• Represent the system state:

  • A vector representing the system nodes containing the percentage of queries running there.

• Optimization algorithm:

  • Search in vector space, e.g. hill-climbing, simulated annealing, quadratic programming.

• Implementation of the control cycle:

  • Suspend, migrate e restart jobs.
Implementation for Queries over Clouds (4)

- Four studied algorithms:
  - 1: ECA rules to minimize response times.
  - 2: Utility functions to minimize response times.
  - 3: Use method 2 to predict when the SLA will be missed.
  - 4: Minimize the number of SLA violations.

Implementation for Queries over Clouds (5)

- 4 queries submitted to a 12 node simulator. 1 of the nodes is running other jobs.
Kinesthetics eXtreme

- Most development efforts in AC is for new systems, but making legacy systems autonomic is also needed.
- Kinesthetics eXtreme (KX): Columbia Univ. project to bring autonomic properties to legacy systems.

Adapting Legacy Components

- Adding autonomic mechanisms to existing code is: hard, error-prone, expensive and not reusable.
- KX goals:
  - A solution that is orthogonal to the business logic and communication of the legacy system.
  - A common external decentralized infrastructure for all managed elements.
  - An infrastructure that interacts closely with the managed elements.
  - A solution that operates at the level of the local configuration and of the general system reconfiguration.
1. Data is gathered from the running system by non-invasive probes. *(Monitor)*

2. **Metrics** map probe data onto the architecture model. *(Analyze)*

3. The **decision** and control (**controllers**) analyses the implications of interpreted data for the general system operation and performance and determines when and how to *(Plan)*:
   - Change (add/remove) probes.
   - Operate actuators to reconfigure or adapt the managed element or system.

4. **Actuators** effectively perform the changes *(Execute)*.
Probes (Monitor)

- Probes can be:
  - **Probelets** which are Java agents (worklets) that run on a WVM (Worklet Virtual Machine).
  - Modified application DLLs, changed using Teknowledge Mediator Connector (**Probe**).
  - Modified application classes (**Probe**). Modified Java code using ProbeMeister.
- Probes generate events that are transmitted to the node running the autonomic manager. KX supports several event description specifications.
**Worklets**

- Mobile software agents supporting a micro-workflow.
- Contain one or more units of code: *worklet junctions*.
- *Worklets* move between nodes, adding, removing and copying *junctions*.
- They run on the Worklet Virtual Machine (WVM).
- Eventually, they return to the node that generated them.
Metrics (Analyze)

• The metrics module corresponds to the MAPE-K analysis phase. It has two event processing components:
  • Event Packager: pre-processing.
  • Event Distiller: inference and pattern detection.
• Communication between Packager, Distiller and the controllers that do the planning uses an event publish/subscribe model.

Event Packager

• It is the system’s black box ("Flight recorder"). Normalizes the events coming from the probes:
  • Attributes global timestamps to the events.
  • Writes events into a relational BD.
  • Supports replaying of the event sequence (auditing, data mining).
Event Packager

- Aggregates and filters sequences of low level events and performs transformations written in ECA rules.
- Allows adding/removing plugins, containing descriptions of event transformations into higher abstractions:
  - Key/value pairs.
  - XML SmartEvents
  - Others...

XML SmartEvents

- XML-described events that are dynamically interpreted using a DB of semantic elements.
- When the autonomic manager finds an event (or part of its XML description) which it doesn’t understand it searches a DB of XML snippets searching for a translation of the event into its components.
The Event Distiller detects time patterns among events and feeds the planning process with its observations.

It uses a state machine and temporal logic to correlate events from different sources.

New patterns can be added dynamically.

For example, it has been used to detect bursts of spam:

```xml
<state name="a" timebound="-1" children="b">
  <attribute name="from" value="true"/>
  <attribute name="subject" value="2"/>
</state>
<state name="b" timebound="100" count="1" children="">
  <attribute name="from" value="true"/>
  <attribute name="subject" value="2"/>
</state>
```
Workflakes: Worklet Management

• Workflakes corresponds to the MAPE-K execution component.
• Worklets distribute code associated to a worklet jacket – pre-conditions, activation scripts, activation and repetition calendars, and conditions under which the worklet will be replaced by an update (micro-workflow).
• The Workflakes module, depending on planning, emits multiple worklets that autonomously adapt the system (macro-workflow).
• (The implementation of the adaptation mechanism wouldn’t necessarily have to use an agent mechanism.)

Functional Separation

• The worklet junction code defines and executes computations.
• The workflow modelling language defines coordination and the Workflakes engine puts coordination into practice by:
  • Specifying the worklets path.
  • Generating worklet jackets.
  • Selecting the code junctions to be included in the worklet.
Architecture Model
(Planning)

- KX uses the Acme language which is a proposed norm for architecture description.
- Client-server example:
  ```
  System simple_cs = {
      Component client = { Port send-request; };
      Component server = { Port receive-request; };
      Connector rpc = { Roels { caller, callee} );
      Attachments {
        client.send-request to rpc.caller;
        server.receive-request to rpc.callee;
      }
  }
  ```
- Actually, the logic is in the Event Distiller but in the future events will be used to dynamically change the architecture’s ACME model.
- Additionally, it will be possible to infer parts of the system architecture from the incoming events and gradually fill in the model.
- And test changes on the model before applying them on the managed system.
KX Applications:
Fault Detection and Load Balancing in GeoWorlds

- GeoWorlds is a geographical information system from US PACOM (USA Pacific military command).
- It uses multiple external real time sources to get information, e.g. CNN, BBC, Yahoo, etc.
- These sources sometimes fail (service interruptions, format changes,...)

Fault Detection and Load Balancing in GeoWorlds

- GeoWorlds’ Java code is instrumented to generate XML events and send them to a KX server.
- The Event Packager receives events and orders them.
- The Event Distiller checks whether the start and finish of calls to external services are within time limits.
- In case anomalies are detected, the components that call the faulty services are restarted or deactivated.
- In case the Event Distiller detects recurring errors, other recovery plans are attempted.
Next Time...

- Conclusion & Exam examples