Artificial intelligence in the service of system administrators

Christophe Haen
V.Barra, E.Bonaccorsi and N.Neufeld

Université Blaise Pascal, CLERMONT-FERRAND cedex, FR
LHCb Online team (CERN), GENEVA, CH

22th May 2012
Plan

1. The LHCb Online System
2. State of the Art
   - Expert Systems
   - Autonomic Systems
3. Our Architecture
   - Methods
   - Analysis of a problem
4. Results
5. Outlook
LHCb

- One of the four large experiments of the LHC.
- Relies on large and heterogeneous IT infrastructure.
- Thousands of servers, different hardware configurations, great variety of tasks
- Small team
The LHCb Online System

problems

- Huge workload per person.
- Night on-call duty.
- Potential loss of knowledge when a student leaves.

Solution

A software that would

- act as a knowledge base
- act as a history database
- improve with experience

Final goal: ease the work of our system administrators
Expert Systems

**Expert System**

- Computer system that emulates the decision-making ability of a human expert.
- Started in 70’s.
- Consists of a knowledge base and an inference engine.

**Pro**

- Knowledge very modular
- All conclusions can be explained

**Cons**

- Difficult to maintain the base up to date
- No improvement with experience
Computer systems are very complicated: need a lot of manpower.

Similar problem in the 1920s in telephony with the manual switchboards.

Human body has autonomic nervous system for unconscious reflexes.

⇒ Need to give autonomic capacity to computing systems
Autonomic Systems

**Self-X**

1. Self-Configuration
2. Self-Optimization
3. Self-Healing
4. Self-Protection

**MAPE-K loop**

- Monitor
- Analyze
- Plan
- Execute
- Knowledge
Autonomic Systems

How to apply those principles

- Generic approach vs. Particular case
- Access to the source code vs. Hermetic source code

Common points

1. Rule based system in the background between measurements and actions
2. Need to describe what to monitor
3. Need to describe how to monitor
4. Need to specify the rules to trigger an action
5. Need to specify what to do in case of problems
Context

A generic approach

- No software restrictions
- Linux only
- No network diagnostics

Linux Process & Files properties

- \( \text{Software} = \{\text{Files}\} \cup \{\text{Processes}\} \)
- Interfaces defined by the standards
- Software independent
- Flip-side: low level directives
Using the MAPE-K loop

Existing solutions
- One MAPE loop per subsystem
- Multi agent theory for communication
- Need for synchronization, dialogue and negotiation
- Split the complexity of knowledge

Our solution
- One MAPE loop shared between all the subsystems
- No complexity of communication
- Allow to discover dependencies
- All the knowledge centralized
Monitoring

**Ideal monitoring**
- Fast
- Reliable
- Easy to interpret

**Model Based approach**
- Abstract representation
- Test changes
- Unrealistic in our environment
  ⇒ Neural network, fuzzy logic, etc

**Classic monitoring**
- Thousands of checks
- Never everything ok
- Need to sort the information

**Service monitoring**
- Much fewer checks
- User oriented
- Independent of the internal mechanism
Reinforcement Learning

- Basic principle: elementary entities OK $\Rightarrow$ application OK
- Too many entities to check: minimize the amount of tests
- Problems are often recurrent

Convention Over Configuration

- Encourage default behavior
- Reduces the workload of the user
- Well adapted in that case
Experience Sharing

- Reinforcement learning requires many replications
- Similar behavior for similar entities
- Copy object model: composition/aggregation, inheritance/instance
- Optimizes the convergence speed
- Reduces the workload for the user
Agents

ProcessAgent httpd {
    procName -> /usr/sbin/httpd;
    user -> root;
    server -> $varSrv;
}

ProcessAgent httdpChild {
    parent -> httpd;
    user -> apache;
}

FileAgent Site1Index{
    server -> $varSrv;
    filename -> /var/www/html/index.php;
    owner -> root;
}
Coordinator WebSite {
    needs Coordinator WebData;
    needs Coordinator WebConf;
    needs ProcessAgent httpd;
}
Coordinator Site1Data {
    needs FileAgent Site1Index;
}
Coordinator Site1 : WebSite {
    Data = Site1Data;
    set $varSrv webServer01;
}
Analysis of a problem

1) problem list
8) monitor request

AGGREGATOR

2) Top problems
5) Value request

SCHEDULER

4) Call

AbstractAgent

9) Add rules
3) Use rules
6) Use rules

RULES

7) Update

Agent

Coordinator
Testbench

- Up to 4 simultaneous random problems
- Each block is monitored as a service
- Unknown dependencies
- Different activation levels (1, 200) for the rules
- With and without Experience Sharing between Site1 & Site2
Diagnosed problems

- Nothing: no rule, no Experience Sharing
- Shared: Experience Sharing
- Rules $n$: Rule activated after $n$ occurrences

Percentage of problems diagnosed
For $n$ simultaneous problems, the software needs at least $n$ interactions.

Additional interactions with respect to the optimal solution
Visited agent

Additional checks with respect to the optimal solution
Experience Sharing

If not optimal, in average:

- not shared: 5.37 checks
- shared: 4.30 checks

Additional checks on the second simulation
Experience Sharing

Optimal solving (in %) vs. replications
### Real Test

- Real web setup: a dozen websites, one NFS server, one DB
- 1 and half month
- All diagnostics were correct:
  - Web servers down
  - Problem with underlying application
  - NFS share unavailable
  - DB down
Outlook

Plans

- Improve agent code
- Add suggestions to recover
- Data mining functionalities
- Tools to manipulate the software (CLI/GUI/API)
- Automatic profile generation
- Profile generation based on UML graphic tool