Autonomic Computing and Grid

Grid Computing : Fran Berman, Geoffrey C. Fox, Anthony J. G. Hey

Chapter 13 : Pratap Pattnaik, Kattamuri Ekanadham. Joefon Jann

Catherine Ruby

February 26, 2004

CSE 718 presentation

Image courtesy of the Buddha Dharma Education Association Inc
(http://www.buddhanet.net/index.html)

The ‘Four Noble Truths’ of Grid Computing

- Managing a large computing system is very complicated.
- The cause of this is the constant growth of computing and its increasing complexity as more heterogeneous components are added.
- There is a way of managing large scale computing environments.
- This is by following the “Noble Eightfold Path” of autonomous computing.

IBM's Eight Defining Characteristics of an Autonomic System (Part 1)

- The system must know its own components and their details.
- The system must change its configurations constantly with changing environments.
- The system must continuously look for ways to optimize its process.
- The system must recognize abnormal conditions or problems that may harm its workings and be able to recover from them.

IBM’s Eight Defining Characteristics of an Autonomic System (Part 2)

- The system must be protected against attacks.
- The system must know its environment, surroundings, and other resources available to it.
- The system must have open standards and operate in “heterogeneous world”.
- The system must be able to stay ahead of the user and guess intelligently what resources will be required and how to use them efficiently while maintaining a simple interface with the user.

Autonomic Grid Computing

MAIN GOALS:

- reduce the work and complexity associated with a large system
- be able to better respond to sudden changes in the system and adjust settings appropriately

First Issue : Component Management

- develop a strategy for merging various computer components with different capabilities
- make use of components like CPU, memory, disks and network as efficiently as possible
- dynamically make changes to management of components as the environment dictates

Second Issue : Changes Over Time

- deal with changes in demands over time.
- manage sudden or unexpected changes
- have some way to deal with long term changes in surrounding environment
- must both detect and appropriately choose responses to any considerable changes in the system

Goal-Oriented Model

- Make most autonomous decisions on the local level and establish clear responsibilities between local and global autonomous components (reduces complexity)
- Poor decisions on a smaller scale will not significantly affect the whole if corrected appropriately.

The Big Picture

A "Goal Oriented" autonomous system:

- Provides services to components in system
- Receives services from other components
- Adapts quickly to a rapidly changing environment or errors in previous behaviors
- Adjusts to new components that may be added over long periods of time

Common approaches to handling complicated systems

- Object Orientation – hides aspects of the system from components that don't require the information. Monitor system constantly and dynamically choose best response.
- Fault-tolerant Systems – maintains a set of faults and can detect or correct faults from that set in the system (fault = reduction in performance). Monitor constantly and readjust when behaviors of surrounding components are abnormal.

Agent-based Resource Management for Grid Computing

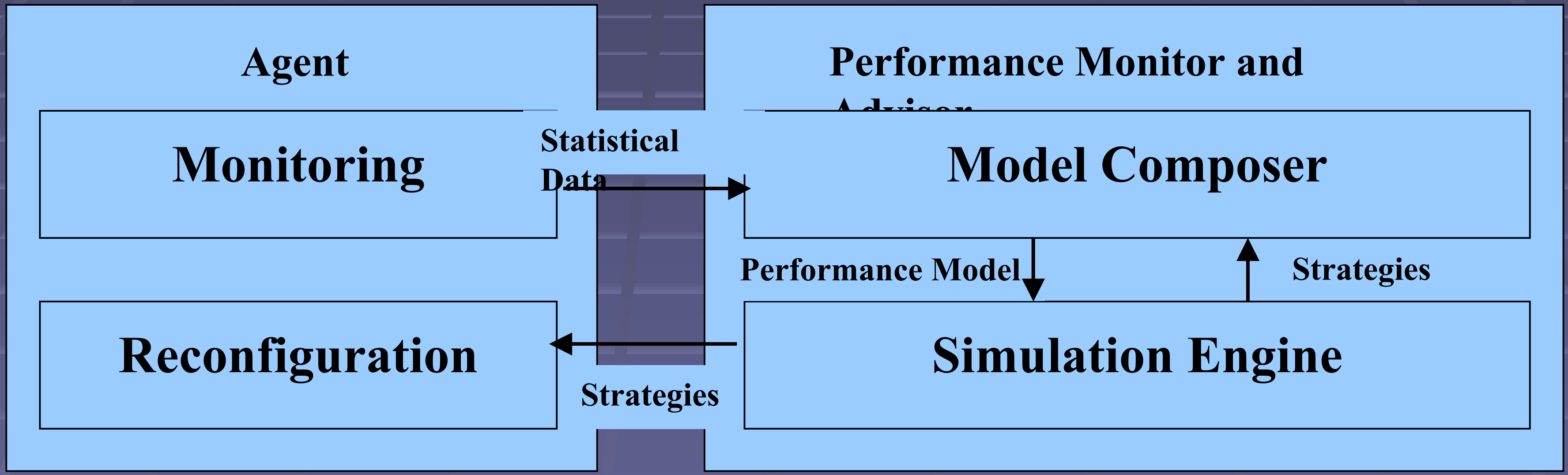
An example of autonomous individual components to manage a complex grid system

A4 (Agile Architecture and Autonomous Agents) make up a high level abstraction of a grid structure where individual agents both discover and advertise resources.

Implementation of A4 Agents

- Agents in the system exist with the ability to communicate with one another, the ability to adapt to sudden external changes, and are "self motivated"
- PMA's, or "Performance Monitors and Advisors" keep track of agents and reconfigure them periodically to keep them performing optimally.

The A4 Agent and Performance Monitor Advisor



examples …

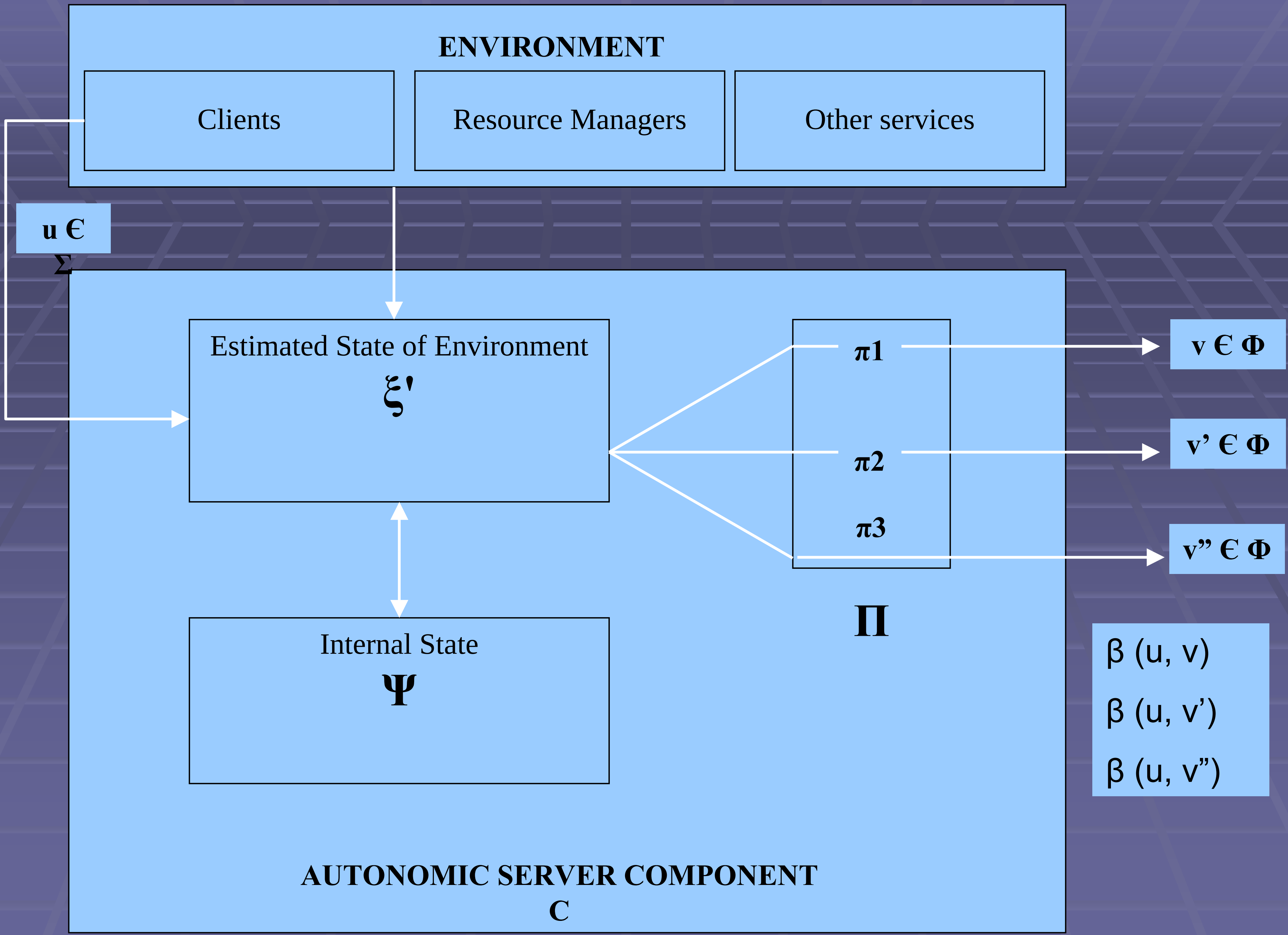
The basic server component

- Take an “Autonomic Server Component, C”
- Take all of the elements which interact with C, or its “environment”

…

Component C’s Greek Letters

- Σ : input alphabet of the component
- Φ : output alphabet of the component
- β (u, v) : relation satisfying appropriate I/O pairs
- Ψ : internal state of component (data structure to keep track of all elements of current state)
- ξ : external state of component (abstraction of external environment, exterior performance of other components. Dynamic, is not accurate, ξ’ is the estimate)
- π : an implementation translating internal and external states into input/output pairs
- Π : set of implementations in component C
- α : algorithm which chooses best implementation (is it worthwhile to switch?)
- η : efficiency of the implementations of C


ENVIRONMENT
Clients
Resource Managers
Other services
u Є Σ
Estimated State of Environment
ξ'
Internal State
Ψ
π1
π2
π3
Π
v Є Φ
v' Є Φ
v" Є Φ
β (u, v)
β (u, v')
β (u, v")
AUTONOMIC SERVER COMPONENT
C

α and the maintenance of ξ’

- α depends almost entirely on characteristics of entire system and the efficiency of its different implementations
- C must keep a fairly accurate ξ’ picture of the external environment to make good internal decisions
- C must be able to periodically update ξ’ from external information and keep ξ’ within reasonable limits of actual ξ.

Two Methods of Maintaining ξ’

- Self-Observation Approach
- Collective-Observation Approach

Maintenance of ξ' : Self-Observation

- No external state information received from other components
- All information about environment from C's interaction
- Maintained by logs of history & input/output interactions w/ clients and services, tracks quality
- ξ' continuously revised with all new inputs

Self-Observation (Pros & Cons)

- PROS :

  - Component is largely independent of external objects – it can easily be plugged into any environment and it will function

  - Adapts well to changes over a long period of time

- CONS :

  - Component cannot adapt quickly to sudden variations in input – it takes several interactions to recognize changes in the environment

Self-Observation Example : "Memory Allocator"

Autonomic Server :

input requests → efficient input handling

- Two inputs :

  Alloc (n) – request block of 'n' bytes

  Free (a) – returns previously allocated block

- Three Outputs : null, error, address
- Four Goals : quick turnaround, never deny requests, block locality and minimized fragmentation

“Memory Allocator” Implementations

“Linked-List Allocation”

- Maintains list of addresses & free blocks, searches for free blocks >= request size n
- If no blocks are big enough, it divides the block, removes it from the List and returns the address
- When a block is returned, an attempt is made to merge it with other free space

“Slab Allocation”

- Reserves several “slabs” of memory of size most frequently requested.
- When memory is requested a free block of the appropriate size from a “slab” is returned

“Memory Allocator” α strategy

- Internal state Ψ keeps track of free slabs and linked list, ξ’ tracks requests for memory
- α chooses “slab allocator” when slabs for requested size exist, otherwise it chooses the linked list implementation
- If input is continuously requesting one block size for which a slab does not exist, a new slab is created for it (faster)
- Slabs are returned to the list when they go unused for an extended period of time

$$\eta \text{ (From Slab)} < \eta \text{ (From List)} < \eta \text{ (New Slab)}$$

α must weigh the costs of each implementation with the costs of allocating from current implementation based on records of input block allocation requests

Maintenance of ξ' : Collective-Observation

- The system is connected by services to and from each component
- Component C can maintain ξ' by receiving updates from the components that surround it
- Components update surrounding components by broadcasting current states to other components
- Broadcasting is expensive!

Example of Component States

Component C_1

Has two internal states at time t :

- S_{11} (t) - the current state of component C_1
- S_{12} (t) - the current state of component C_2

Component C_2

Has two internal states at time t :

- S_{21} (t) – the current state of component C_1
- S_{22} (t) – the current state of component C_2

Δ^t_i : the estimated derivative of S_{ij} (t) at time t :

thus $S_{ij}(t + dt) = S_{ij}(t) + \Delta^t_i (dt)$

Collective-Observation Example 1 : "Subscriber Approach" (push)

- Component C_1 subscribes to component C_2 if it is interested in C_2's state, stores this subscription in ξ'
- C_1 can estimate the state of C_2 at time t because it grows at a rate of Δ^t_2.
- C_2 monitors its own state, if its state grows beyond what is expected ("tolerance rate"), it computes a new Δ^t_2 and sends this new information to all of its subscribers like C_1
- Broadcasts are proportional to rate of change of component states

Collective-Observation Example 2 : "Enquirer Approach" (pull)

- Components are only updated when they explicitly request information from the components they are subscribed to
- Each component sets "tolerance limit" of states of components it is subscribed to
- When the state of a component C_2 grows beyond the bounds of this "tolerance limit", C_1 requests updated information

Collective-Observation Example 3 : "Routing By Pressure Propagation"

- Entire system receives information from outside of the system – each component inside the system can process any incoming information (common in Web Services)
- When a component receives incoming information, it is passed to an input queue of a specific component
- Selection of this "specific component" is autonomic, aim is to minimize response time

Collective-Observation Example 3: Selection of Component

- Each component maintains information about the whole system
- $<\mu_1, \tau_{12}>$: μ is how long it take component C_1 to process any incoming information, and τ is how long it will take to send the request to the appropriate component C_1 from C_2
- Time for transaction from C_1 to processing at C_2 : $[\tau_{12} + (1 + Q_2) * \mu_2]$: Q = length of queue
- However, C_1 has no knowledge of queue length of C_2

Collective-Observation Example 3: Estimated Queue Length Maintenance

- Like other collective-observation examples, each component keeps a list of states with the estimated queue length with a variation with respect to time of all other components
- When component C_1 receives information, it finds the component C_2 which will minimize the transaction time and sends it there
- When a component's queue exceeds the "tolerance limit", it updates its estimated queue length among all other components in the system

The Grid

Grid system is heterogeneous, so protocols must be defined to share information

- Fabric Layer - defines protocols for accessing components of system
- Connectivity Layer – security protocols
- Resource Layer – protocols for getting resources
- Collective Layer – protocols for finding services and managing them on both user and grid levels

Service : an abstraction, a guarantee of a certain "behavior" between layers via these protocols, though the standard implications of a "behavior" are still being researched.

Services and Components

- A “service” is similar to the component model of the autonomic grid
- The service has an implementation that guarantees the behavior is met
- It must keep track of outside information and modify its behavior accordingly
- Thus a service should also have an algorithm α to change its behavior according to changes in the environment and choose the most performance-efficient resources

A Component-focused approach :

- Hides complexity from other elements in the system
- Contains a simple interface with other components
- Maintains the ability to detect and respond to changes in the system and fit its surroundings

Requirements of Autonomic Components

- Present interface between clients and server
- Monitor its environment as time progresses
- Appropriately and rapidly change behavior when changes in surrounding environment occur (changing demands or other components failing)

Future Research

- Generating new algorithms and stabilizing existing ones
- Managing behaviors of components
- Standardizing changes of these behaviors with respect to time

References

Buddha Dharma Education Association Inc. *The Four Noble Truths : Teachings by Ajahn Sumedho*. http://www.buddhanet.net/4noble.htm

Cao, Junwei, Daniel P. Spooner, James D. Turner, Stephen A. Jarvis, Darren J. Kerbyson, Subhash Saini, and Graham K. Nudd. *Agent-based Resource Management for Grid Computing*. http://www.dcs.warwick.ac.uk/research/hpsg/html/downloads/public/docs/CaoJ.ARMGC.pdf

Horn, P. Autonomic Computing, 25 February 2004, http://www.research.ibm.com/autonomic/.

Pattnaik, Pratap, Kattamuri Ekanadham and Joefon Jann. *Autonomic Computing and Grid*. Grid Computing.