

# Performance Contracts: A Fuzzy Logic Perspective\*

Dan Reed                      Ruth Aydt  
{reed,aydt}@cs.uiuc.edu

Department of Computer Science  
University of Illinois  
Urbana, Illinois 61801

## 1. Introduction

Historically, performance analysis has focused on monolithic applications executing on large, stand-alone parallel systems. In such a domain, measurement and post-mortem analysis and code optimization suffice to eliminate performance bottlenecks and optimize applications. Most existing performance analysis systems (e.g., SvPablo [1], Medea [2], and Paragraph [5]) use only post-mortem analysis. To tune the emerging distributed applications, a new generation of on-line performance measurement and optimization tools is needed that can adapt application behavior dynamically, based on changing resource availability.

To meet this goal, while providing guaranteed levels of performance, any adaptive grid infrastructure must address the following challenges:

- *Noisy performance measurements* – precise performance contract specification with exact numerical values for performance is inappropriate. Only with behavioral ranges can one successfully validate contracts.
- *Multiple resources* – specifications must accommodate multiple resources (e.g., computation, network, memory, and I/O).
- *Temporal specification* – multiple behavioral phases will occur in computations and in available resources. Specifications must be rich enough to enable temporal description of dependencies.
- *Performance contract validation* – general mechanisms are needed to evaluate performance contracts and take action when they are violated.

Based on these challenges, we believe fuzzy logic specification is an attractive mechanism for specifying and evaluating performance contracts. Similarly, the *Autopilot* control model elegantly integrates measurement, fuzzy logic evaluation, and action.

## 2. Performance Instrumentation and Distributed Control

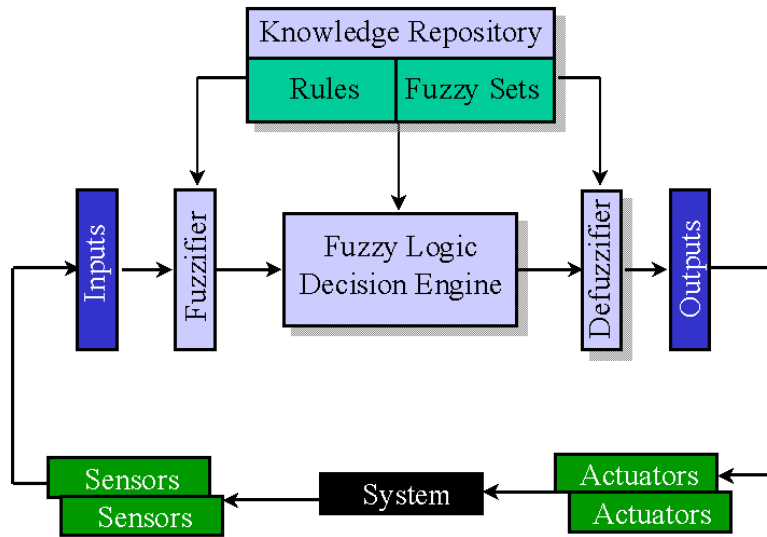
To optimize distributed applications, performance measurement software must capture data from multiple sites, a variety of hardware and software platforms, and application software components written in a variety of languages. Complementing measurement, distributed control mechanisms should allow users or control software to modify run-time parameters or resource allocation during the application's execution.

### 2.1 Autopilot: Closed Loop Control

As Figure 1 suggests, our *Autopilot* toolkit [1,2] for distributed measurement of executing software components relies on fuzzy logic for specification of system behaviors. *Autopilot* is built atop the Globus system for computational grids [7] and defines a set of software sensors and actuators for instrumenting application code and controlling code behavior via fuzzy logic specifications.

---

\* This material is based upon work supported by the National Science Foundation under Grant No. 9975020. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.



**Figure 1** Fuzzy Logic Specification and Evaluation

*Autopilot* sensors are low overhead routines designed to capture real-time performance data from distributed software components and can be extended via user-defined functions to process raw performance data before transmission (e.g., computing a profile from event trace data). To minimize the complexity of sensor specification and configuration, *Autopilot* relies on the *SvPablo* [1] system for sensor insertion in distributed application source code.

In addition to data capture, *Autopilot* supports adaptive control of remote applications and systems. Software actuators, inserted in application code or embedded in runtime libraries, provide mechanisms for controlling functions in software modules. At the application level, actuators can be used to change variable values or choose among algorithms for a particular task. Similarly, runtime system actuators can change resource management policies and their parameters (e.g., file system prefetching or caching policies).

Together, *Autopilot* sensors and actuators provide the performance data needed to measure dynamic, distributed behavior and the mechanism to modify that behavior. The *Autopilot* decision infrastructure accepts data from distributed sensors as inputs and relies on actuators to realize the results of the decision process. There are a variety of classic techniques for implementing such distributed decision mechanisms, ranging from decision tables and trees to standard control theory; *Autopilot* relies on fuzzy logic specifications.

## 2.2 Fuzzy Logic Specification

In contrast to classic decision procedure techniques and their emphasis on consistent parameter space division, fuzzy logic targets precisely the attributes of the resource management problem that challenge classic techniques [9], namely conflicting goals and poorly understood optimization spaces. The overall system allows manipulation of linguistically described concepts through use of common sense knowledge (e.g., file prefetching benefits small, sequential reads).

Figure 1 shows the basic flow of information through the *Autopilot* fuzzy logic decision mechanism. A fuzzy controller relies on fuzzy sets to represent the semantic properties of each input (sensor) and output (actuator). The variables are processed by a set of IF-THEN production rules that map the input values to the output space through a collection of fuzzy sets. These input sensor values may represent raw sensor

```

IF (Utilization == LOW)
    {DiskParallelism = HighPar;}

IF (Utilization == MEDIUM)
    {DiskParallelism = MediumPar;}

IF (Utilization == HIGH)
    {DiskParallelism = LowPar;}

```

**Figure 2** Example Fuzzy Logic Rule Base

metrics (i.e., unchanged since their point of capture), synthesized sensor metrics processed by one or more sensor attached functions, user hints, qualitative classifications, or any combination of these.

Unlike boolean logic values that are either true or false, the fuzzy variables of Figure 2 can assume any value ranging from 0 (false) to 1 (true) with a variety of different transition functions from 0 to 1. For example, the UTILIZATION fuzzy variable could be defined with a smooth transition from HIGH (completely true) if devices are always busy to LOW (completely false) if devices are always idle.

Fuzzy logic theory supports a wide variety of transition functions for fuzzy variables and many functions for the fuzzy analog of boolean operators [8]. These fuzzy operators combine continuous fuzzy variables to yield a fuzzy output (e.g., a fuzzy AND might take the minimum of two fuzzy logic values across their ranges).

In consequence, multiple fuzzy rules can be partially true simultaneously. The antecedent of each IF-THEN rule is first computed, yielding a fuzzy logic truth value. Then, the truth of the conclusion is derived by scaling each rule's conclusion by the degree of truth of the antecedent. Finally, taking the union of the outputs of all rules with the same consequent yields the actual result.

The appeal of the fuzzy logic approach to policy control is that policy transitions are smooth, rather than discrete as with decision tables or trees. Hence, one can quickly and easily change the rule set or the functions describing truth values to rapidly explore a variety of different management policies.

### 2.3 Fuzzy Logic Decision Example

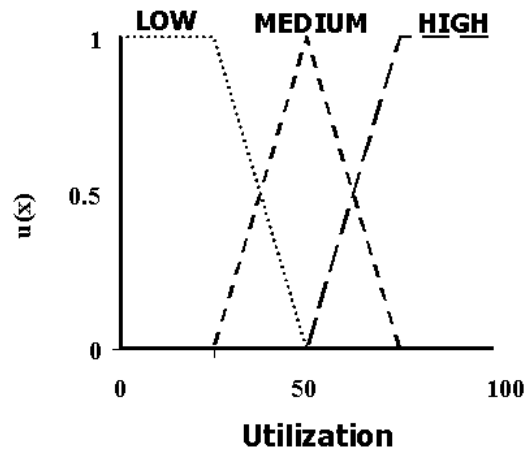
Figure 2 illustrates a simple set of fuzzy logic rules that might be used to control adaptive file striping (disk parallelism) in an adaptive input/output system. *Autopilot* sensors provide a time-varying stream of utilization metrics. After fuzzification, this sensor data stream defines the value of the *Utilization* fuzzy variable.

In the figure, all the rules whose conditions are non-zero (i.e., at least partially true) contribute to determining the value of the output fuzzy variable *DiskParallelism*. After defuzzification, the value of *DiskParallelism* defines the action taken by an *Autopilot* actuator to adjust the number of blocks that are prefetched via an actuator.

The advantage of decoupling decision mechanisms from sensors and actuators should now be clear. The rule set of Figure 2 is architecture independent; neither the source of fuzzy inputs nor the sink of the fuzzy outputs is specified. The value *Utilization* is an abstraction whose values can be bound to a sensor value, a user hint, or even the output of another decision procedure. Similarly, *DiskParallelism* is an abstraction of an actuator, with no implicit mapping.

In the example, the fuzzy sets *LOW*, *MEDIUM*, and *HIGH* are specified separately. In Figure 3, these sets are defined as trapezoidal, triangular, and trapezoidal, respectively. By changing the set specification and its tightness (e.g., making the triangle more acute), one can change the tightness of the bounds.

The decision procedure inputs and outputs are bound to specific sensors and actuators via global pointers



**Figure 3** Example Fuzzy Set Specification

from the sensor/actuator manager. By changing the mapping, one can apply the rule set using different sensors, choose different policies, or even control different systems.

### 3.0 Conclusions

Fuzzy logic provides an attractive mechanism for specifying imprecise performance contracts. Not only can multiple contracts be in force concurrently, the decision mechanism can readily balance conflicting goals via a formal process. Moreover, fuzzy logic separates specification of rules from fuzzy sets. Hence, one can tighten or loosen the specification simply by changing the set specification.

Finally, the fuzzy logic specification mechanisms are an integral part of Autopilot and Globus. This means that they can be applied quickly to target applications, allowing us to meet the basic GrADS goals.

### References

- [1] L. DeRose, Y. Zhang, and D. Reed, "SvPablo: A Multi-Language Performance Analysis System," *Computer Performance Evaluation Modeling Techniques and Tools*, pp. 352-355. Lecture Notes in Computer Science, Vol. 1469, Springer-Verlag, R. Puigjaner, N. Savino, and B. Serra (eds.), September 1998.
- [2] L. DeRose, M. Pantano, R. A. Aydt, E. Shaffer, B. Schaeffer, S. Whitmore, and D. A. Reed, "An Approach to Immersive Performance Visualization of Parallel and Wide-Area Distributed Applications," *Proceedings of the Eight IEEE International Symposium on High-Performance Distributed Computing*, pp. 247-254, August 1999.
- [3] M. Calzarossa, L. Massari, A. Merlo, M. Pantano, and D. Tessera, "Medea: A Tool for Workload Characterization of Parallel Systems," *IEEE Parallel and Distributed Technology*, 3(4), pp. 72-80, November 1995.

- [4] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan-Kaufmann, 1999.
- [5] M. T. Heath and J. A. Etheridge, "Visualizing the Performance of Parallel Programs," *IEEE Software*, pp. 29-39, September 1991.
- [6] D. Reed, K. Shields, W. Scullin, L. Tavera, C. Elford, "Virtual Reality and Parallel Systems Performance Analysis," *IEEE Computer*, 28(11), pp. 57-67, November 1995
- [7] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *International Journal of Supercomputer Applications*, 1997.
- [8] N. K. Kasabov, *Foundations of Neural Networks, Fuzzy Systems, and Knowledge Engineering*, The MIT Press, 1996.
- [9] L. A. Zadeh, "Fuzzy Sets," *Information and Control*, 8(3), pp. 338-353, June 1965