Autopilot: Adaptive Control of Distributed Applications

R. L. Ribler, J. S. Vetter, H. Simitci, and D. A. Reed

Department of Computer Science
University of Illinois

Presenter:
Chia-Jui Hsu

November 10, 2005

Autopilot

- Autopilot is a software infrastructure for dynamic performance tuning of heterogeneous computational grids based on closed loop control.

Outline

- Introduction
- Autopilot Overview
- Autopilot Software Components
- Fuzzy Logic Control
- Autopilot Performance
- PPFS II: An Autopilot Testbed
- Current Status

Software Performance Tuning

- Traditional *a posteriori* approach
  1. Application instrumentation
     - Instrumented automatically by object code patching or compiler.
     - Manually insert instrumentation library calls.
  2. Execute, measure, and extract performance data
  3. Analysis and visualization
     - Identify performance bottlenecks
  4. Application optimization
     - Modify program, adjust runtime policies
- This tuning model presumes repeatability
  - Application has repeatable behavior.

Heterogeneous Computational Grids

- Grids
  - Hardware and software infrastructure.
  - Provide access to computational capabilities.
  - Integrate large numbers of geographically distributed resources.
- Grid application behavior is rarely repeatable
  - Complex and multidisciplinary.
  - Time varying resource demands / requirements.
  - Dynamically changing resource availability.
- The *a posteriori* performance tuning model is not suited to heterogeneous computational grids applications.

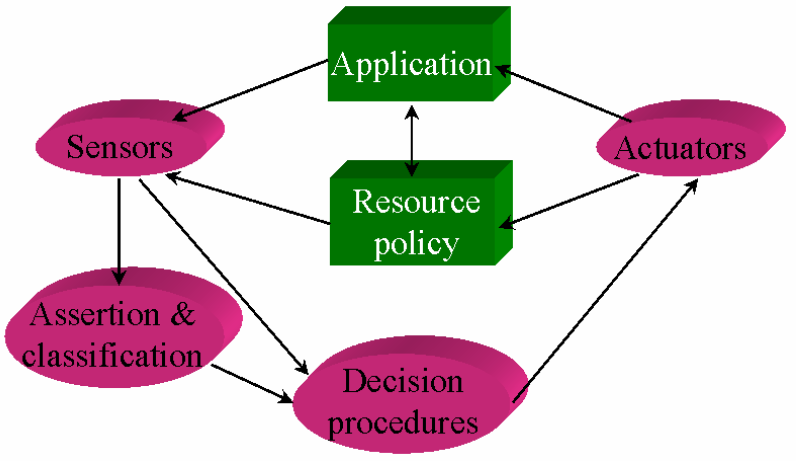
Real-time Adaptation

- Applying real-time adaptive control techniques to dynamically adapt the system to different resource demands and resource availability.
- Autopilot
  - Integrate dynamic performance instrumentation.
  - Configurable resource management algorithms.
  - Real-time adaptive control mechanisms.
  - Based on application request patterns and observed performance.
  - Automatically configure and manage resources.

Autopilot Software Components

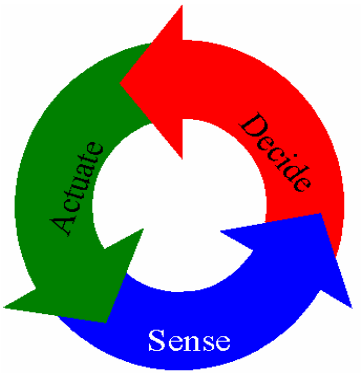
- Distributed sensors
  - Capture run-time data and send data to clients.
- Software actuators
  - Receive commands from clients, and adjust application behavior (parameter values) and resource policies.
- Clients (sensor and actuator clients)
  - Monitor sensor data and issue commands to actuators.
- Distributed name servers (Autopilot manager)
  - Remote sensors and actuators register in name servers.
  - Clients request for sensors and actuators (based on their properties) through name servers.
- Decision mechanism
  - Select right resource management policies (fuzzy logic) based on application requests and sensor data.

Autopilot Adaptive Control Infrastructure



Close Loop Control

- Sensor: sense and transmit data to client.
- Client: decide and send commands to actuator.
- Actuator: actuate application behavior and resource policies.

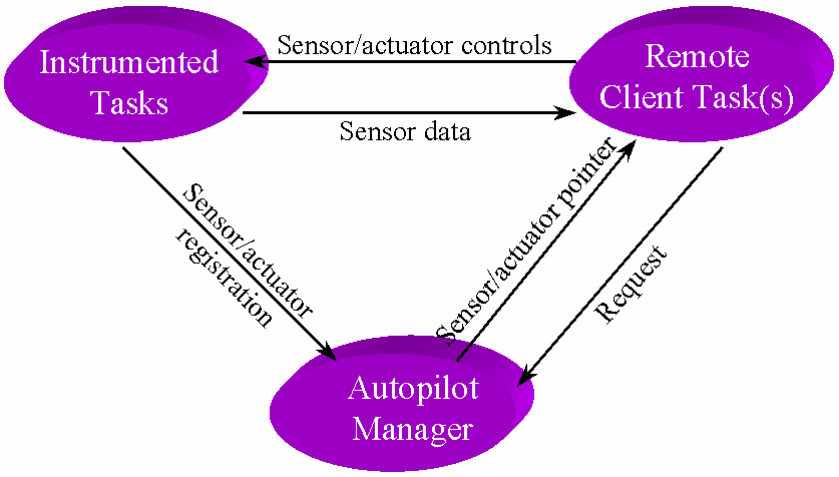


Nexus Toolkit

- The Autopilot software is built on Nexus Toolkit (communication substrate).
- Endpoint (Nexus term) refers to a global address and specifies a set of message handlers.
- Startpoint (Nexus term) identifies a pointer to an endpoint.

- The Autopilot manager is a daemon name server process.
- Sensors and actuators register at the Autopilot manager.
- A client obtains startpoints to sensors and actuators through the Autopilot manager, then it can communicate to sensors and actuators.
- Sensors and actuators also need to obtain startpoint to their clients before communication.

Autopilot Manager



Autopilot Sensor Design Principles

- Autopilot sensors should be lightweight, low overhead, minimally perturbing, remotely accessible, and applicable to multiple architectures and programming models.
- Minimize data capture / extraction / transmission overhead in order to less perturb the system.
- To avoid oscillating decision due to stale sensor data, the latency between data collection and decision making must be small.
- Transmit raw data v.s. compute and transmit derived metrics.
- Balance transmission frequency v.s. buffer size
  - Transmission Frequency ↓, transmission overhead ↓, buffer size ↑, latency ↑.

Autopilot Sensor / Actuator Features

- Property list
  - Every sensor and actuator is associated with a set of properties.
    - Name, type, network address, and user-defined attribute-value pairs.
  - Clients specify property lists in queries to the Autopilot manager. The manager then replies startpoints that satisfy the query.
- Activation modes
  - Threaded mode: a monitoring thread records data at specified interval.
  - Non-threaded mode: it relies on insertion of sensor monitoring calls.
- Attached functions
  - Sensors can have attached functions for data reduction.
    - E.g., Computing sliding window average.
  - Actuators can have attached functions to mediate remote commands.
- Dynamic activation and deactivation.

Distributed Name Servers

- Autopilot managers act as name servers and coordinate connections between sensors, actuators, and clients.
- Sensors and actuators register their properties and Nexus startpoints with an Autopilot manager after creation, and also inform the manager when they are destroyed.
- Clients specify a set of desired properties, and the manager provides startpoints to the sensors or actuators that match the request.
  - Clients can acquire remote sensors and actuators without knowledge of their location.
  - Clients can dynamically attach to distributed software components, exercise control, then relinquish the attachment.

Clients

- Clients establish direct communications to sensors and actuators.
- Sensors send data to all the connected remote clients.
- Clients process data, make decisions, and issue commands to remove actuators to implement decisions.
- Clients can change sensor / actuator behaviors.
  - Activation, buffer size, sampling rate.

Sensor Registration Code

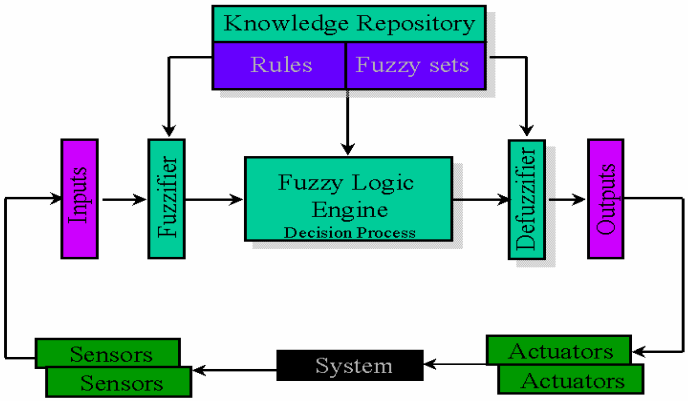
```
// Define Properties for RequestSize Sensor
ApProperties RequestSizeProperties(progName, mgrName);
RequestSizeProperties.addProperty("Name", "RequestSizeSensor");
RequestSizeProperties.addProperty("Application", "PPFS II");
// Construct RequestSizeSensor.
ApIntegerSensor RequestSizeSensor("RequestSizeSensor",
    RequestSizeProperties, requestSize, variableCount=1, bufferSize=8);
// Register Sensor with Autopilot Manager
RequestSizeSensor.registerStartPoint();
```

Decision Mechanisms – Fuzzy Logic

- Closed loop adaptive control.
- Fuzzy logic
  - Fuzzy logic incorporates an empirical rule-based IF - THEN approach to solve control problem rather than attempting to model a system mathematically.
  - Balance potentially conflicting goals (e.g., minimize response time and maximize throughput).
  - By changing the fuzzy logic rule base, they can adjust the control system or even retarget to a new domain without extensive software development.

Fuzzy Logic Decision Process

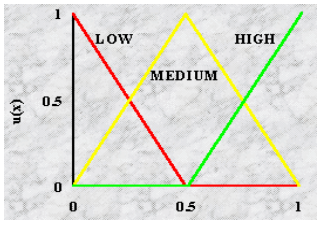
Autopilot includes a fuzzy logic engine that accepts sensor inputs, fuzzifies the values, computes the relative truth of each rule, and defuzzifies the consequents to activate remote actuators.

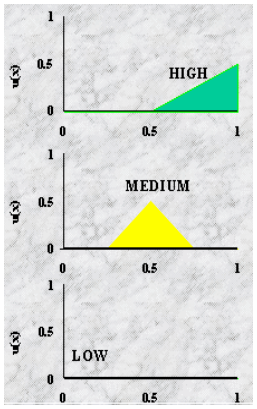


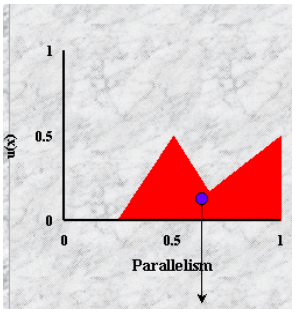
Fuzzy Logic

- Fuzzifier
  - Scales and maps input variables to fuzzy sets.
- Decision mechanism
  - Interpret IF-THEN rules.
  - If the premise (input) is true in a rule to some degree.
  - Then the consequent (output) of the rule is true to the same degree.
  - Multiple rules can be active for the same input.
- Defuzzifier
  - Combine multiple rule outputs through a defuzzification method (combination can yield complex behavior).
  - Generates final output.

Fuzzy Logic Example





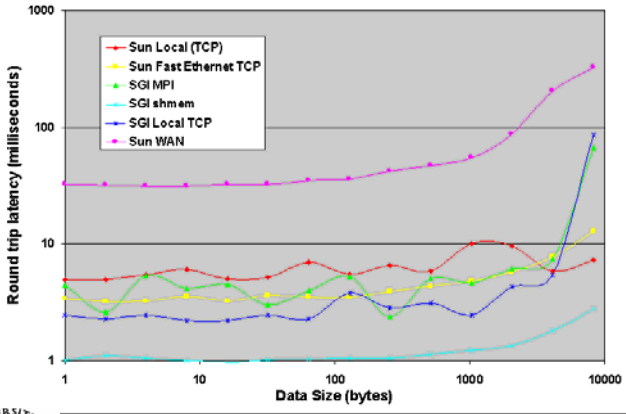


Autopilot Performance

- Performance depends on the following factors:
  - Sensor buffer sizes.
  - Threaded and non-threaded sensor modes.
  - Number of clients/sensor.
  - Dynamic data reduction via attached functions.
  - Complexity of fuzzy logic rules.
  - Distance of sensors/actuators/clients.
  - Available network bandwidth.
  - Data throttling via sensor activation/deactivation.
  - Actuator synchronization.
- This paper only discusses data buffering and fuzzy rule complexity.
  - Sun Ultra 1 Model 170 (170 MHz UltraSPARC processor) with 64 MB of memory, and an SGI Origin2000 with 32 195MHz R10000 processors and 4 GB of memory.

Buffer Size v.s. Round-trip Latency

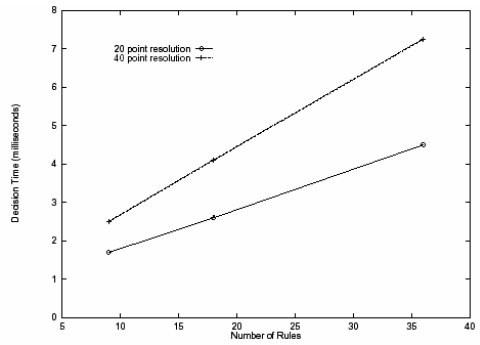
- The round trip delay including data buffering and transmission in wide area, local area, and intra-system contexts as a function of the sensor buffer size.



For small buffer sizes, communication overhead is dominated by communication latency, ranging from 1 ms between multi-processor in SGI share memory system to 40 ms in wide area network.

Fuzzy Rule Complexity v.s. Decision Time

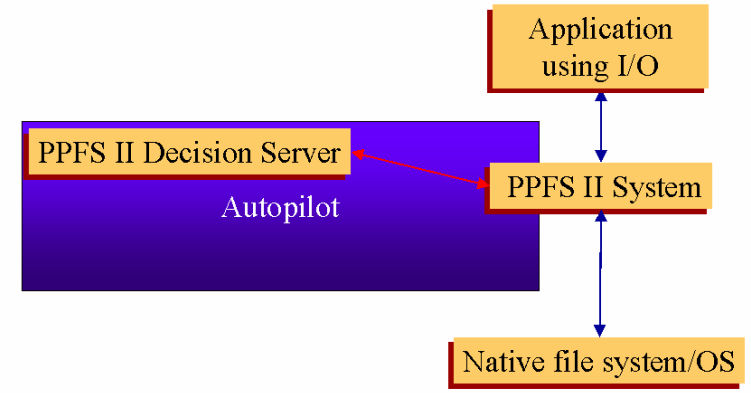
- The time to evaluate a rule base as a function of both the number of rules and the resolution of the fuzzy sets (the number of sample points).
- Rule evaluation overhead is linear in both the number of rules and the fuzzy set resolution.



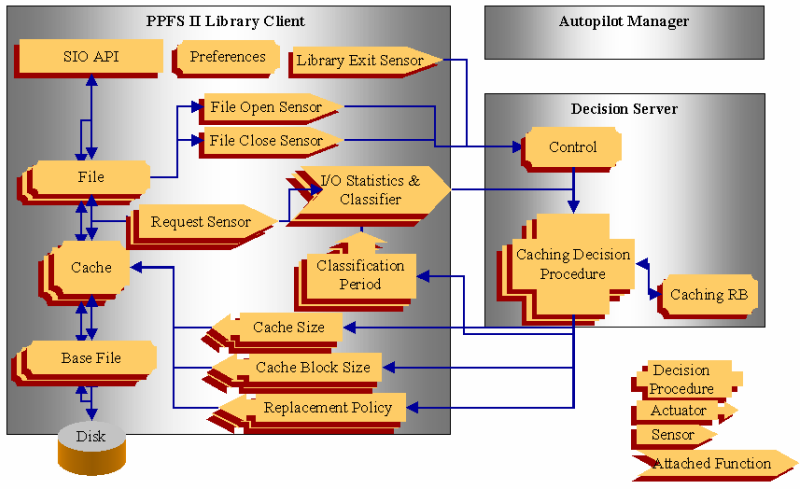
PPFS II

- PPFS II: an adaptive parallel file I/O system.
  - It is designed to operate on top of either parallel systems or PC/workstation clusters.
- Use Autopilot to adapt PPFS II in real-time .
- Sense
  - Quantitative data: I/O instrumentation (e.g., read size, write size, …)
  - Qualitative data: access pattern (e.g., sequential writes, …)
- Actuate
  - Cache block size, cache size, cashing policy.
- Fuzzy logic control
  - 9 rules.
- Application
  - PPFS II benchmark.

PPDS II Integration Overview



PPFS II Architecture



PPFS II Experiments

- Benchmark:
  - A 128 MB file is first written sequentially in 16 KB units. Then a 32 MB section of the file is read randomly four times using 16 KB access sizes.
- Compare non-adaptive to adaptive approach
  - For non-adaptive approach, PPFS II uses default caching parameters: 4 KB cache blocks and a 16 MB client cache.
  - For adaptive approach, a fuzzy logic decision procedure continuously monitors and adjusts the file system using Autopilot infrastructure.
    - Phase I (Write): 8MB cache and 60 KB block
    - Phase II (Read): 30MB cache and 80 KB block

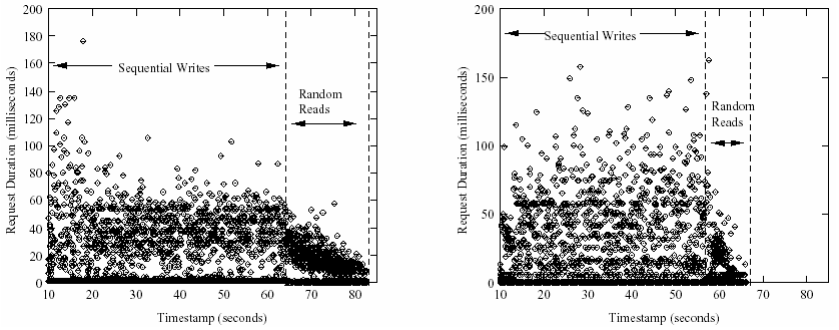
Adaptive I/O Rule Base

- Small input/output requests are best managed by aggregation, pre-fetching, caching, and write-behind.
- Large requests are better served by streaming data directly to or from storage devices and application buffers.

```
if ( ReadWriteMix == READONLY && Sequentiality == NONSEQUENTIAL &&
     RequestSize == LARGE )
{ CachingEnable = DISABLED; }
if ( ReadWriteMix == READONLY && Sequentiality == NONSEQUENTIAL &&
     RequestSize == TINY )
{ CachingEnable = ENABLED; CacheSize = HUGE; BlockSize = LARGE; }
... ...
if ( ReadWriteMix == WRITEONLY && Sequentiality == SEQUENTIAL )
{ CachingEnable = ENABLED; CacheSize = SMALL; BlockSize = LARGE;
ReplacementPolicy = MOSTRECENTLYUSED; }
```

PPFS II I/O Benchmark

- Dynamic adaptation decreases the benchmark execution time from over 80 seconds to less than 70 seconds.



Current Status

- Pablo research group in the Department of Computer Science at the University of Illinois at Urbana-Champaign.
- Randy L. Ribler, Jeffrey S. Vetter, Huseyin Simitci, and Daniel A. Reed, "Autopilot: Adaptive Control of Distributed Applications," Proceedings of the 7th IEEE Symposium on High-Performance Distributed Computing, Chicago, IL, July 1998.
- Software distribution:
  http://pablo.renci.org/Software/Autopilot/autopilot.htm

References

1. Randy L. Ribler, Jeffrey S. Vetter, Huseyin Simitci, and Daniel A. Reed, "Autopilot: Adaptive Control of Distributed Applications," Proceedings of the 7th IEEE Symposium on High-Performance Distributed Computing, Chicago, IL, July 1998.
2. http://pablo.renci.org/Project/Autopilot/AutopilotOverview.htm
3. http://pablo.renci.org/Publications/Presentations/HPDC7/index.htm
4. http://pablo.renci.org/Publications/Presentations/Autopilot%20Overview/index.htm