ISTORE: Introspective Storage for Data-Intensive Network Services

A. Brown, D. Oppenheimer, K. Keeton, R. Thomas, J. Kubiatowicz, and D.A. Patterson

Overview
- Introduction
- ISTORE Hardware Architecture
- ISTORE Software Architecture
- An Example
- Related Work
- Conclusions

Introduction
New Era of Computing
- PCs, workstations, servers > powerful, distributed, networked information services
- Information > Raw Computational Power
- Driven By
  - E Commerce
  - Information Search and Retrieval
  - Online Decision Support

Introduction (Cont')
- Availability that permits a mean time to failure measured in no less than years
- Performance that allows for simultaneously servicing millions of clients
- Scalability that keeps up with the rapid growth of the user base and the data those users need to access
Total Cost Of Ownership

- Studies have shown that maintenance and administration costs are enormous - amounting to anywhere from 2x to 12x the cost of the hardware

Introspection - Definition

- Looking into oneself
- Continuous monitoring
- Adaptation

Introspective Capabilities

- Self-Administrating
- Monitor for Exceptional Situations
- Changes in the Imposed Workload
- Maintain Availability and Performance Goals

Human Maintenance

- Low Human Maintenance
  - Add New Hardware When Resources Become Low
  - Replace Faulty Hardware After Failure
- Performance Tuning and Incorporation of New Hardware is Handled Automatically
  - Ie: Plug n Play
ISTORE

- Provides the necessary framework for
  - Application-Appropriate Semantics
  - Optimizations
  - Responses to External Stimuli
- ISTORE Intelligent Storage Architecture
  - Hardware/Software Framework
    - Customizable for each application
  - Full Cooperation Between All Levels of the System - from Hardware to Runtime

Introspective Capabilities

- Track Data Access Patterns
- Monitor Media Integrity
- Predicting & Responding to
  - Data Hot-Spots
  - Device Failures

ISTORE Hardware Architecture

- I/O Devices are 1st Class Citizens vs.
  - CPU centric model
- Hot-Swappable LEGO-Like Plug-and-Play Hardware Components (Bricks) w/
  - Embedded Processor
    - ie: Disks, Memory
    - Network Interface
- Intelligent Chassis

ISTORE Hardware Architecture

- Intelligent Chassis
  - Uninterruptable Power
  - Cooling
  - Environment Monitoring
  - Scalable
  - High Bandwidth
  - Redundant, Switched Network
Embedded Processor

- On-Device Intelligence
  - Devices Can
    - Automatically Check Themselves
    - Detect Fatal Errors
    - Fail-Fast Behavior - Disconnect Themselves from the System
    - React to Runtime Behavior
    - Collect Data

ISTORE Software Architecture

- ISTORE’s Runtime System Has 2 Goals
  - Framework to Allow Custom Monitoring Requirements with Related Adaptation Code
  - System generates Monitoring Code and Adaptation Algorithms

ISTORE Database

- Runtime System - Dynamic Database
  - Information (Hardware/Data)
    - Collection
    - Aggregation
    - Processing of Monitoring Data
  - Views - Present a Subset of the All Gathered Information to an Application

ISTORE Database

- SQL-Like Language
  - Database Views
  - Decide What is Interesting
    - Require reaction and adaptation
    - Triggers (database predicates)
      - call application specific handlers for adaptation policy
  - Dynamically Redefinable Views and Handlers
    - when workloads and environment change
**Policy Compiler**

- Each Application has Different Policies
- Built-In Reaction Algorithms
  - Mechanism Libraries
    - replication, transactions, object-based file-access
- Integrate Application-Specific Policies With Reaction Algorithms
- The Compiler’s Input are Integrity Constraints Over a Database View

**Code Generation**

- Translates the integrity constraints into the views and triggers needed to detect violations
- Adaptation Code Templates
  - based on built in reaction algorithms
  - to get system to again meet constraints
  - non-optimized code suitable for most apps

**Hardware Support for Software Introspection**

- Hardware and Software Interplay is Most Evident in the Monitoring Database
- Software assumes that the database is continually updated

**Hardware Support for Software Introspection (Cont’)**

- **Sources**
  - Simple Hardware Monitors
    - temperature sensors
    - performance counters
  - Local Triggers
    - “detect when utilization exceeds 90%”
  - Computational triggers
    - “detect when less than 10% of total disk space is free”
    - “detect when another brick stops responding to network messages”
An Example

- Maintain a Fixed Level of Data Availability in That There Are Always 3 Copies of All Data Objects Available
  - Policy Compiler determines status and performance requirements to detect constraint violation
    - i.e.: disk “health” : Dead/Alive
  - Policy Compiler constructs a View by selecting the health field of the database for each entry that corresponds to a disk

Example (Cont’)

- Define a trigger to detect when “health” field changes to dead
- Policy Compiler generates an adaptation code template
  - i.e.: a generic replication mechanism to
    - find the remaining copies
    - lock them in
    - copy them to the least utilized disk in the system
    - unlock the object
    - update the system index of new data location

Example (Contt)

- The Adaptation Code can be customized by the system designer if desired
  - i.e.: Store the 3rd replica on a disk not currently holding the object.

Related Work

- ISTORE’s adaptive and self-tuning properties reflect a general trend in the software community
- Researchers propose feedback driven adaptation for
  - extensible operating systems
  - databases
  - storage devices
Related Work

- ISTORE Differs from These Projects
  - A Hardware/Software combination vs. Software only
  - Server Adaptability is controlled by the application’s dynamic definition of views
- ISTORE’S Device Bricks are Generalizations of
  - Intelligent Disks
  - Intelligent Network Interfaces

Conclusion …..

By combining intelligent components with an extensible, reactive runtime system, the ISTORE architecture provides a powerful, flexible framework for building introspective servers.

Its modular intelligent hardware is adaptable, easily scaled, and reliable, while its runtime system simplifies and automates the creation of the application-specific monitoring and adaptation mechanisms that are essential to self-maintaining systems.