Managing the File System from the Kernel

Shihabur R. Chowdhury*, Constantin M. Adam, Frederick Wu**, John Rofrano**, and Raouf Boutaba***

*David R. Cheriton School of Computer Science, University of Waterloo
**IBM TJ Watson Research Center

Presented By: Shihabur R. Chowdhury

File System issues

- The file systems get full
  - package files, backup files, “old” files
- Files get accidentally deleted
  - `/etc/resolv.conf` accidentally deleted; name resolution stops
- File permissions get accidentally changed
  - accidentally given global write permission to `/etc/hosts`
  - may cause security problems

File System Management: the traditional way

- File System issues are solved manually
  - Enter commands in the CLI
    - Error prone
    - Time consuming
    - Sometimes unproductive due to repetitive tickets
- Configuration management tools reduce the manual task
  - Less responsive
  - Detects an anomaly after they occur
  - Requires an infrastructure to work

File System Management: the traditional way (contd…)

- Why the file system currently cannot take care of itself ?
  - No knowledge of the file system usage requirements of applications
    - Which are the temporary files ?
    - Which are the required files ?
    - etc.
  - No policy based management interface available
- What if the file system took care of itself ?
  - Reduction in problem tickets
  - Reduction in management overhead

File System Management: The Extreme Automation Way

- Applications should be able to tell the file system about their requirements
  - `/etc/apache/httpd.conf` is a required file; don't delete it
  - Never allow the `/etc/apache/httpd.conf` to be world writable

File System Management: The Extreme Automation Way

- The system administrator should be specify policies; the file system should enforce it
  - There should be at least **10%** free space on **`/mnt/share`**
- In other words
  - Build the management capabilities within the file system itself
  - Provide interface to users and applications

Autonomic File System Management: Properties

- **Self – cleaning**
  - Should be able to clean the ***unnecessary*** files
  - Should be able to expand itself up to ***policy specified*** threshold if necessary
- **Self – protecting**
  - Should be able to prevent ***non policy compliant*** changes
- **Reactive and Responsive**
  - Detect the problem just before they occur
  - Transparently remediate problem reactively

Use Cases

- **Disk Cleanup**
  - Intercept ***write*** operations and detect disk full right before they occur
  - Try to clean up space by deleting files according to ***policies***
  - As a last step ***expand*** the file system
  - After the remediation pass the control back to the original system call

Use Cases

- **File Protection**
  - Allow applications and users to specify access mask for files
    - e.g. `/etc/resolv.conf` can never be world writable
    - Prevent non compliant permission changes in the first place
  - Allow applications and users to specify files ***as required***
    - Prevent accidental deletion of ***required*** files

Policies

- Four initial categories for files
  - ***Temporary*** – can e deleted immediately
  - ***Debug*** – can be deleted after a certain age
  - ***Audit*** – can be compressed to save space
  - ***Required*** – cannot be deleted at all
- Categories identify the deletion / compression candidates

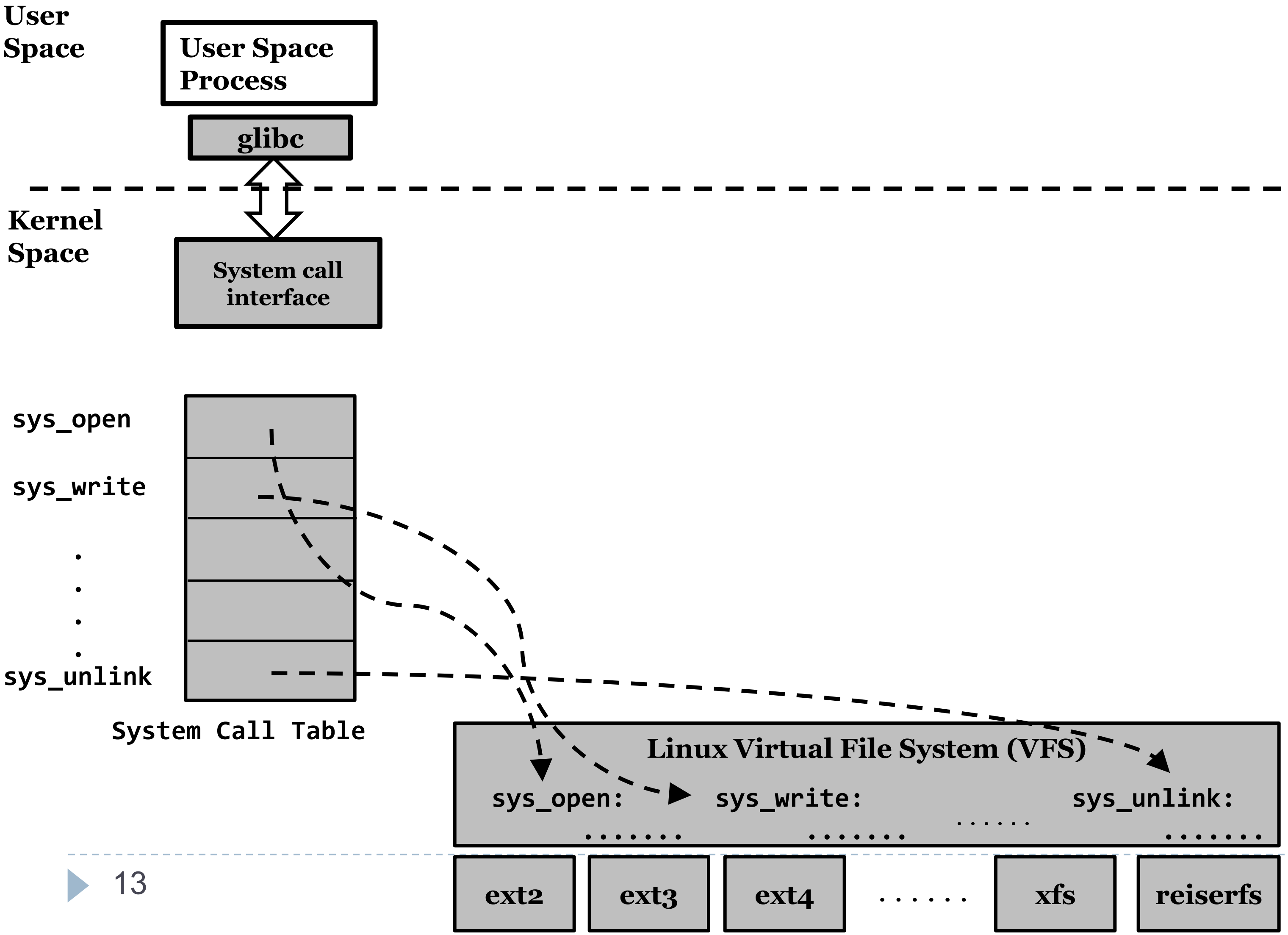
Policies

- ***System-wide policies***
  - Configure category parameters
    - e.g. maximum age of debug files
  - Currently stored in plain text configuration file
- ***Application / User policies***
  - Communicates filesystem usage requirement with the filesystem
  - A user-space API is provided

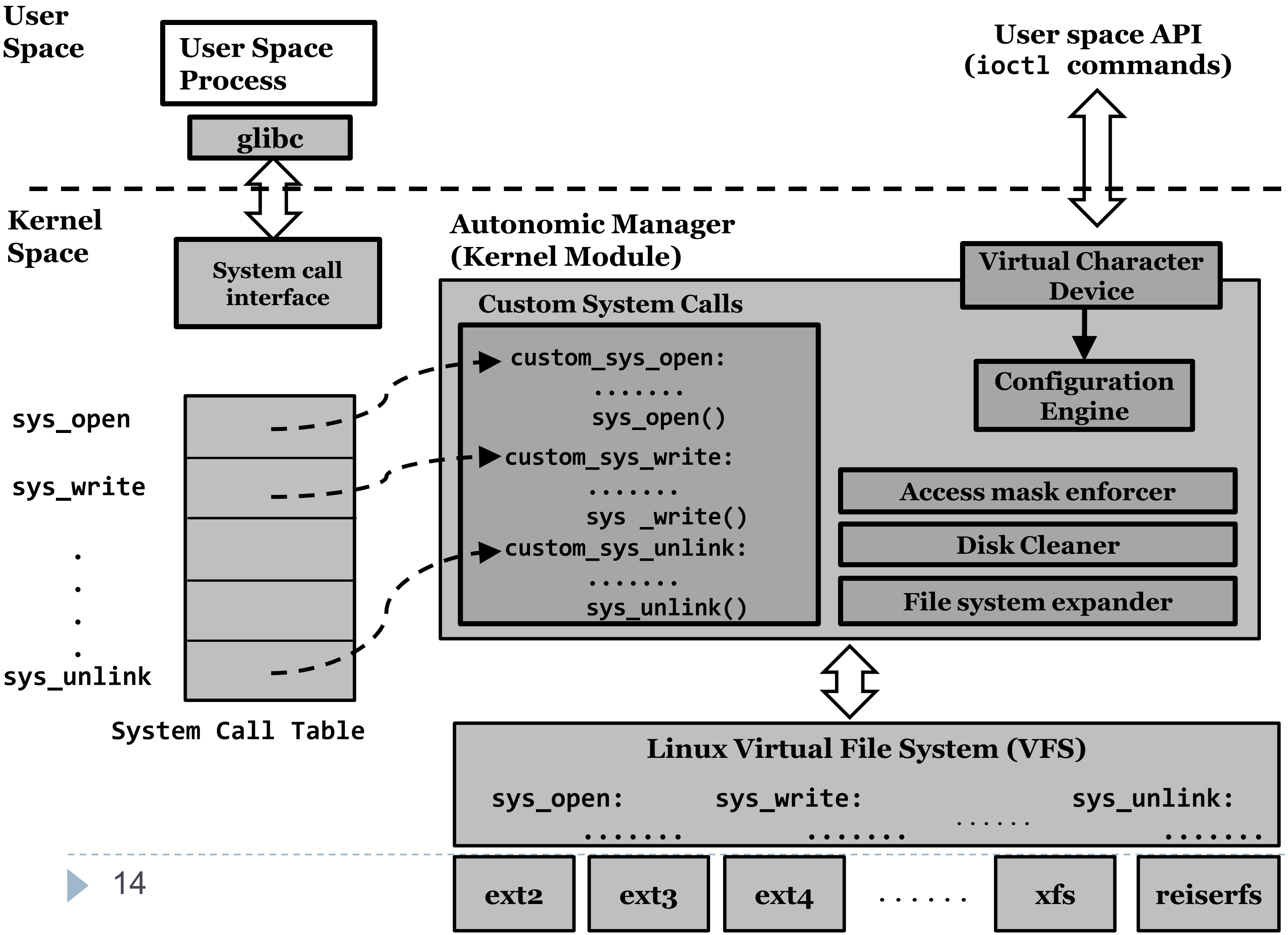
Interfaces

- ***File System Interface***
  - Access low level file system routines for reading its state and performing actions
- ***User-space Interface***
  - Understand the file system usage requirement of applications
  - Allow users and applications to specify their policies

File system: The current picture



System Architecture



Implementation: Proof of Concept

- Autonomic Manager implemented as a Loadable Kernel Module
    - It can intercept system calls
        - We have identified a set of system calls to intercept according to our need
    - Perform error condition checking
    - Perform remedial actions
        - Delete files according to application usage requirements
        - Expand the file system by spawning Logical Volume Management (LVM) processes

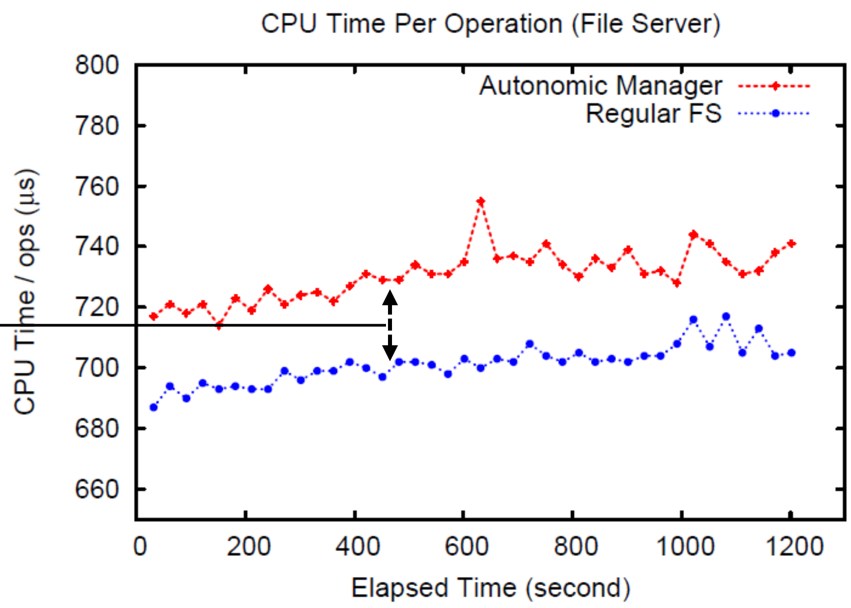
Implementation

- Use-space API
  - The kernel module registers a pseudo device
    - **`/dev/fs_interceptor`**
  - User programs can send control commands to the device
    - Using **`ioctl`** system call
  - The virtual device interprets the commands to configuration commands

Evaluation: Setup

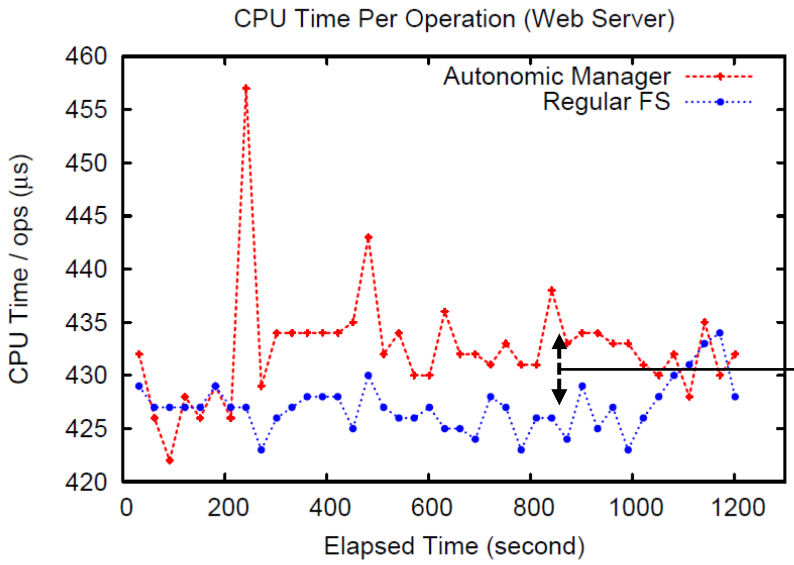
- System configuration
  - Ubuntu 13.04 virtual machine with 2 vCPUs, 3GB memory and 30GB disk
- Benchmarks
  - Filebench
    - File server (1:1 read and write) and Web server (10: 1 read and write) workload
    - Used to measure overhead
      - □ Impact on throughput
      - □ CPU time
  - Postmark
    - Used to demonstrate the effectiveness of self-cleaning property

Evaluation: CPU time


CPU Time Per Operation (File Server)

File server workload

< 7% extra CPU time


CPU Time Per Operation (Web Server)

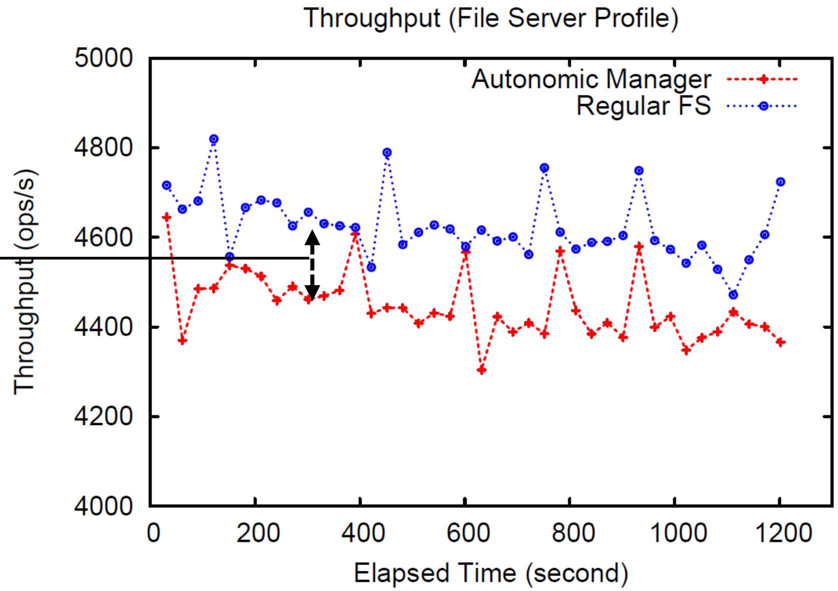
Web server workload

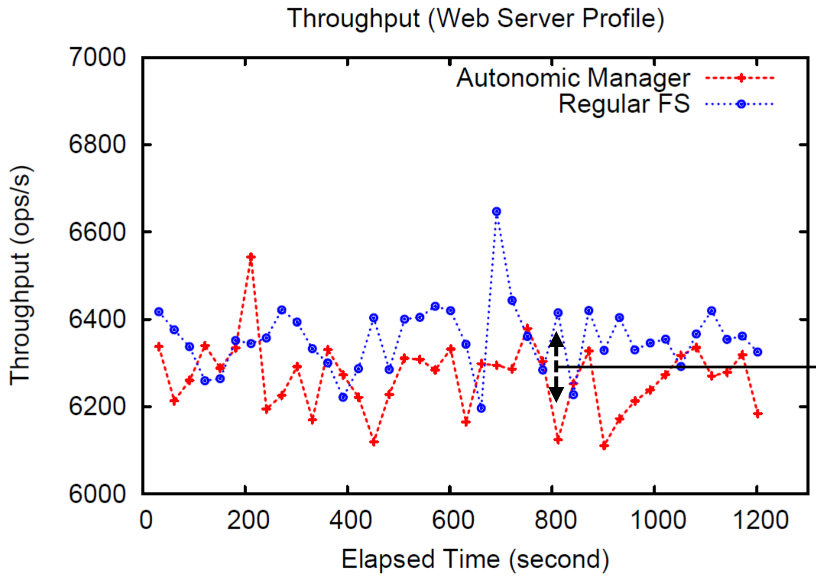
<5% extra CPU time

Evaluation: Throughput

File server workload

< 5% throughput reduction



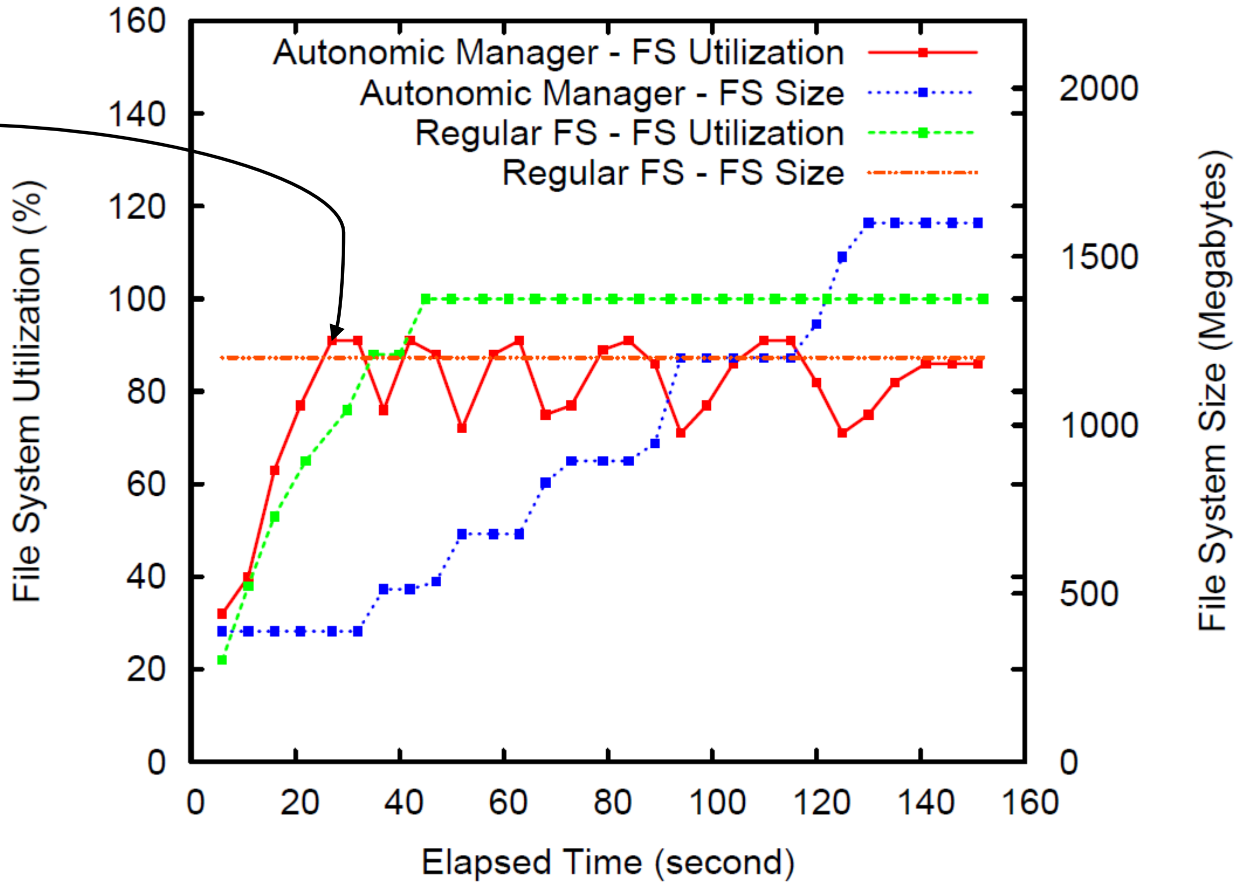


Web server workload

<3% throughput reduction

Evaluation: File System Utilization

Policy specified Utilization (<=90%) Is always maintained



Conclusion

- We need automaticity at the grass – root to make management easier and less error prone
- Autonomy can be at multiple levels
  - When the autonomic file system manager fails, it can notify a higher layer, which has a broader view of the system
  - When all layers fail to solve an issue, the human gets involved
- Autonomic management at the grass root level can be considered for other resources
  - CPU, Memory, Network Interface *etc.*

Questions

?

Related Works

- Autonomic Computing initiative by IBM
  - Monitoring agents monitors for non-compliant behavior
  - Plan an action according to learned environment and knowledge base
- Autonomic OS (AcOS) – *DAC '13*
  - Autonomic resource allocation
  - API for applications to express resource requirement
- Elastic Quota File system - *2002*
  - Allow users to exceed the quota by giving them some reclaimable elastic space
  - Most of the part built as user-space process
- NITIX
  - Self healing and managing filesystem
  - Acquired by IBM in 2008