Behavior-Based Problem Localization for Parallel File Systems

Michael P. Kasick

Rajeev Gandhi, Priya Narasimhan

Carnegie Mellon University

Electrical & Computer ENGINEERING

Problem Diagnosis Goals

- To leverage behavioral instrumentation sources to diagnose problems in an off-the-shelf file system
  - Sources: Instruction-pointer samples & function-call traces
  - Environmental performance problems: disk & network faults
  - Target file system: PVFS
- To develop methods applicable to existing deployments
  - Application transparency: avoid code-level instrumentation
  - Minimal overhead, training, and configuration
  - Support for arbitrary workloads: avoid models, SLOs, etc.

Motivation: Real Problem Anecdotes

- Problems motivated by PVFS developers' experiences
  - From Argonne's Blue Gene/P PVFS cluster

Motivation: Real Problem Anecdotes

- Problems motivated by PVFS developers' experiences
  - From Argonne's Blue Gene/P PVFS cluster
- "Limping-but-alive" server problems
  - No errors reported, can't identify faulty node with logs
  - Single faulty server impacts overall system performance

Motivation: Real Problem Anecdotes

- Problems motivated by PVFS developers' experiences
  - From Argonne's Blue Gene/P PVFS cluster
- "Limping-but-alive" server problems
  - No errors reported, can't identify faulty node with logs
  - Single faulty server impacts overall system performance
- Storage-related problems:
  - Accidental launch of rogue processes, decreases throughput
  - Buggy RAID controller issues patrol reads when not at idle

Motivation: Real Problem Anecdotes

- Problems motivated by PVFS developers' experiences
  - From Argonne's Blue Gene/P PVFS cluster
- "Limping-but-alive" server problems
  - No errors reported, can't identify faulty node with logs
  - Single faulty server impacts overall system performance
- Storage-related problems:
  - Accidental launch of rogue processes, decreases throughput
  - Buggy RAID controller issues patrol reads when not at idle
- Network-related problems:
  - Faulty-switch ports corrupt packets, fail CRC checks
  - Overloaded switches drop packets but pass diagnostic tests

Motivation: Behavioral Approach

- Previous work demonstrates performance-metric approach
  - Performance manifestations masked by normal deviations
  - Certain faults (e.g., network-hogs) not reliably diagnosed
- Performance problems also have behavioral manifestations
  - Overloaded servers act differently from normal servers
  - Behavioral manifestations may be more prominent

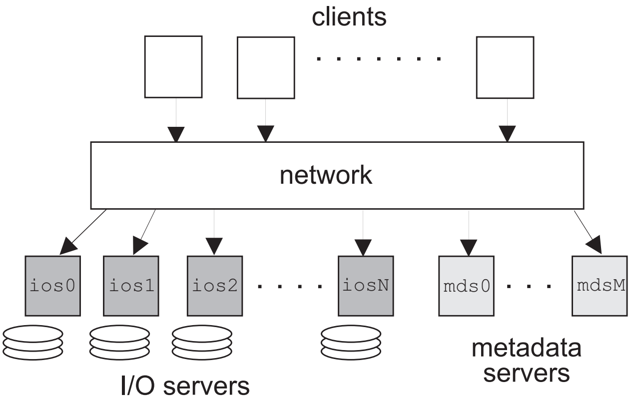
M. P. Kasick et al. Black-box problem diagnosis in parallel file systems. In *FAST*, San Jose, CA, Feb. 2010.

Outline

Parallel Virtual File System

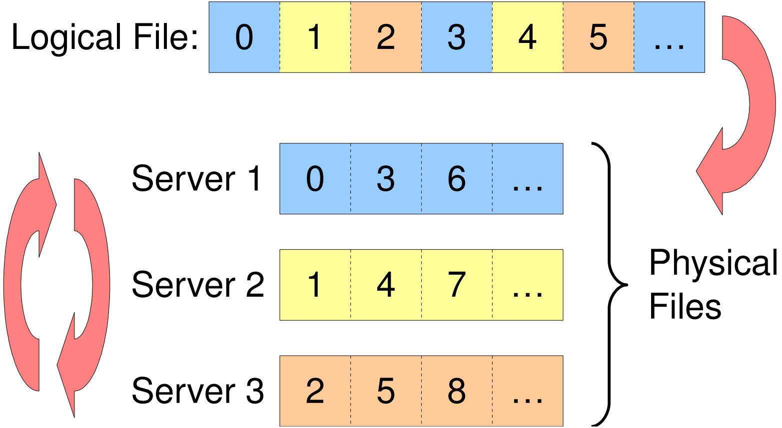
- Open source parallel file system
- Aims to support I/O-intensive applications
- Provides high-bandwidth, concurrent access
- Runs on a cluster of commodity computers

PVFS Architecture



- One or more I/O and metadata servers
- Clients communicate with every server
  - No server-server communication

PVFS Data Striping

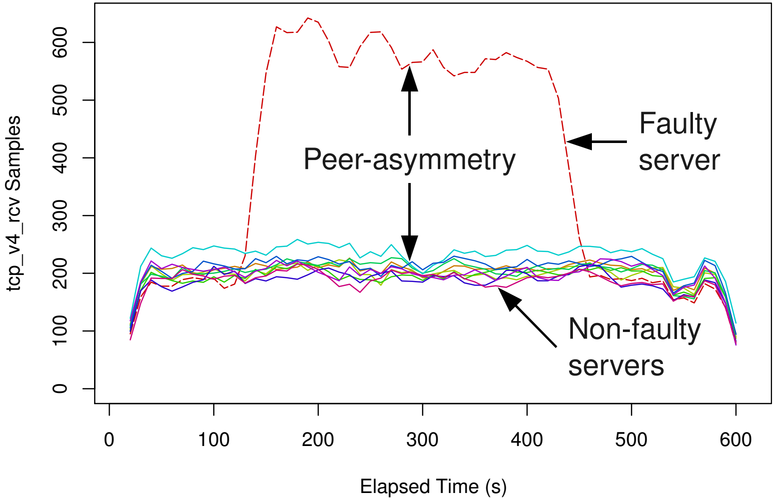


- Client stripes local file into 64 kB–1 MB chunks
- Writes to each I/O server in round-robin order

Parallel File Systems: Empirical Insights

- Server behavior is similar for most requests
  - Large I/O requests are striped across all servers
  - Small I/O requests, in aggregate, equally load all servers
- **Hypothesis:** Behavioral peer-similarity
  - Fault-free servers exhibit similar behavioral metrics
  - Faulty servers exhibit behavioral dissimilarities
  - Peer-comparison of metrics identifies faulty node

Example: Write-Network-Hog Fault



- Strongly motivates peer-comparison approach

Outline

System Model

- Fault Model:
  - Non-fail-stop problems
    - “Limping-but-alive” performance problems
  - Problems affecting storage & network resources
- Assumptions:
  - Hardware is homogeneous, identically configured
  - Workloads are non-pathological (balanced requests)
  - Majority of servers exhibit fault-free behavior

Instrumentation: Sample Profiling

- **Samples** of the CPU instruction pointer:
  - Determines program & function the CPU is executing
  - Statistical approximation of function execution times
  - Measures each function's computational demand
- OProfile: User- & kernel-space sample profiler
  - Samples via NMI every 100,000 unhalted CPU cycles
  - Profiles collected every 10 seconds on each server
  - Samples attributed to application, binary image, & function

Instrumentation: Function-Call Tracing

- Traces of function-call entries & exits:
  - Creates profiles of function-call count & execution time
    - **Count**: Number of times a particular function is called
    - **Time**: Wall-clock time spent executing or blocked in a syscall
  - Provides exact metrics, not approximations
- Custom instrumentation module:
  - Instruments PVFS at build-time, requires source code
  - Count & time profiles collected every second on each server
  - Traces PVFS daemon only, not kernel or other processes

Instrumentation Examples

- Sample profile example:

Application	Image	Function	Samples
pvfs2-server	vmlinux	tcp_recvmsg	658
vmlinux	vmlinux	sk_run_filter	808
vmlinux	vmlinux	tcp_rcv_established	686
vmlinux	vmlinux	tcp_v4_rcv	943

- Function-call trace example:

Function	Count	Time (s)
job_testcontext	58	1.04
dbpf_pwrite	9	0.75
dbpf_dspace_testcontext	118	0.99
dbpf_sync_db	11	0.33

Workloads

- ddw & ddr (`dd` write & read)
  - Use `dd` to write/read many GB to/from file
  - Large (order MB) I/O requests, saturating workload
- iozonew & iozoner (IOzone write & read)
  - Ran in either write/rewrite or read/reread mode
  - Large I/O requests, workload transitions, `fsync`
- postmark (PostMark)
  - Metadata-heavy, small reads/writes (single server)
  - Simulates email/news servers

Fault Types

- Susceptible resources:
  - Storage: Access contention
  - Network: Congestion, packet loss (faulty hardware)
- Manifestation mechanism:
  - Hog: Introduces new workload (visible behavior)
  - Busy/Loss: Alters existing workload

	Storage	Network
Hog	disk-hog	write-network-hog read-network-hog
Busy/Loss	disk-busy	receive-packet-loss send-packet-loss

Experiment Setup

- Cluster of 10 clients, 10 combined I/O & metadata servers
- Each client runs same workload for $\approx$600 s
- Faults injected on single server for 300 s
- All workload & fault combinations run 10 times

Outline

1. Introduction
2. Experimental Methods
3. Diagnostic Algorithm
4. Results
5. Conclusion

Diagnostic Algorithm

- Node Indictment
  - Analyzes sample, count, and time profiles across servers
  - Automatically identifies faulty servers
- Root-Cause Analysis
  - Identifies functions most affected by an anomaly
  - Enables manual inspection of faulty resources

Data Representation: Feature Vectors

- Metric profiles represented as feature vectors
  - Components correspond to profiled functions
  - Values consist of metric sums over a sliding window

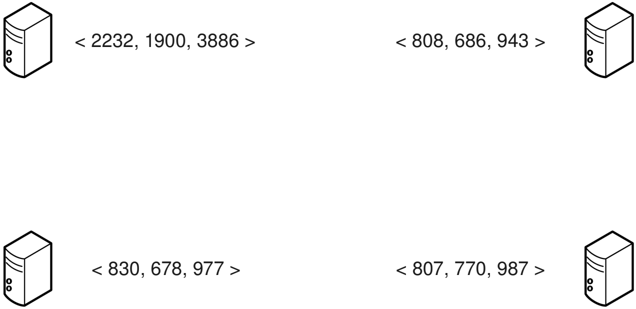
< ...	2232,	1900,	3886, ... >
	`sk_run_filter`	`tcp_rcv_established`	`tcp_v4_rcv`

Node Indictment

- Peer-compare feature vectors across servers

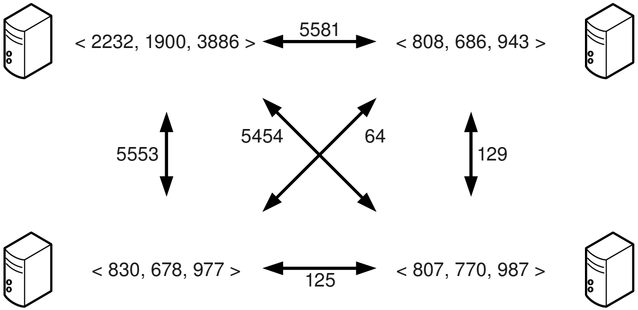
Node Indictment

- Peer-compare feature vectors across servers
  - Compute vectors for each server over a sliding window



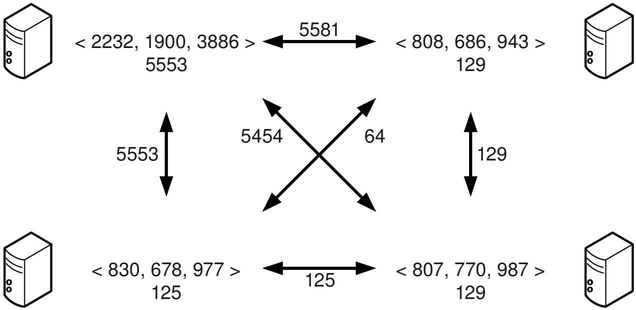
Node Indictment

- Peer-compare feature vectors across servers
  - Compute vectors for each server over a sliding window
  - Compute Manhattan distance for each server pair



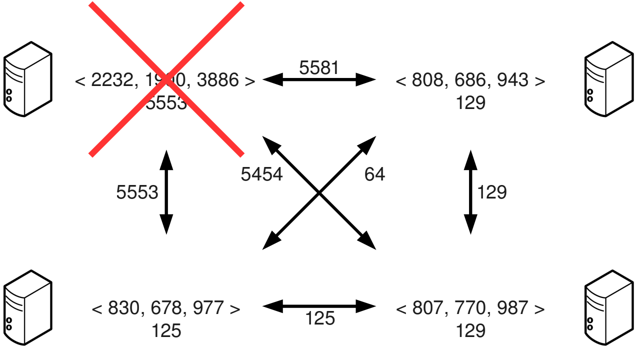
Node Indictment

- Peer-compare feature vectors across servers
  - Compute vectors for each server over a sliding window
  - Compute Manhattan distance for each server pair
  - Determine median pair-wise distance for each server



Node Indictment

- Peer-compare feature vectors across servers
  - Compute vectors for each server over a sliding window
  - Compute Manhattan distance for each server pair
  - Determine median pair-wise distance for each server
  - Flag server if its median distance exceeds threshold



Threshold Selection

- Fault-free training session (stress test)
  - Run ddw, ddr, (& postmark) under fault-free conditions
  - Find minimum threshold that eliminates all anomalies
- Node indictment uses per-server thresholds
  - Captures normal behavioral deviations of each server
  - Important to train on each cluster & file system
- Train on performance-stressing workloads only
  - Behavior deviates most when servers are saturated
  - Caveat: Ignores non-performance-related deviations

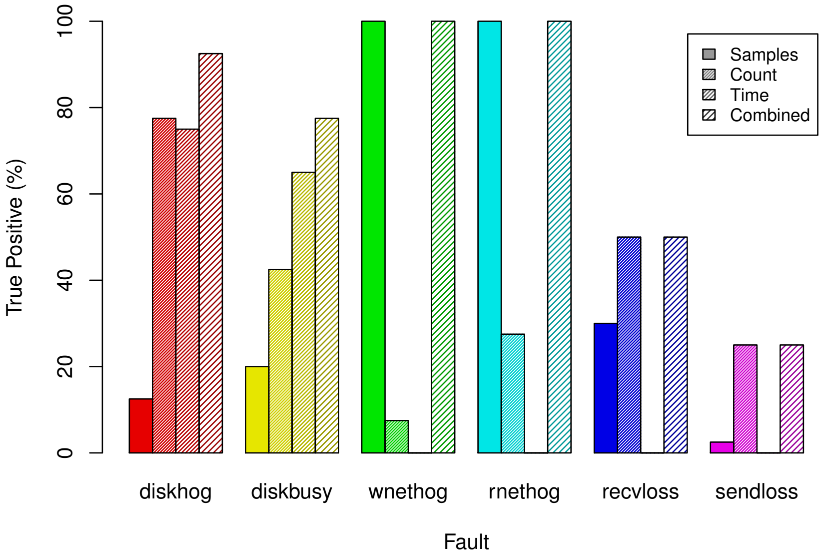
Root-Cause Analysis

- Identify the functions most affected by an anomalous metric
  - Compute component-wise distances to median-dist. node
  - Sum component-wise distances over all windows
  - Rank & present top-ten affected functions for inspection

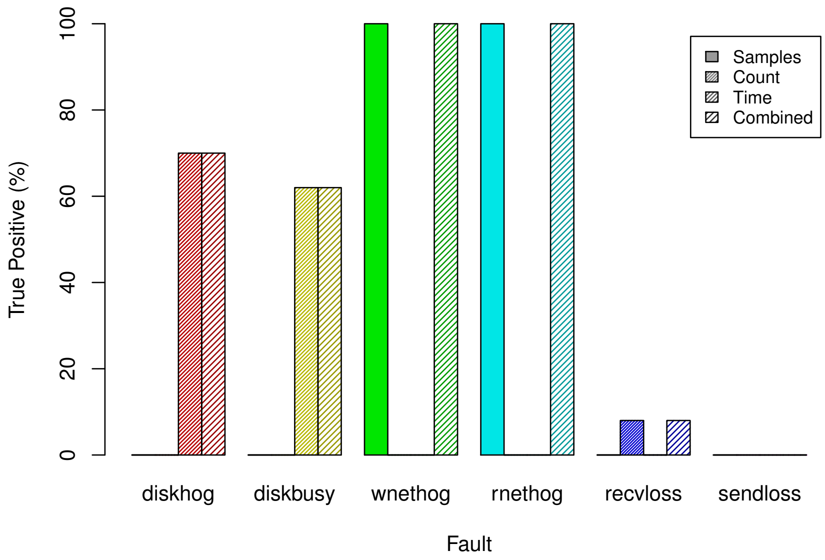
Application	Image	Function
socat	vmlinux	copy_user_generic_string
vmlinux	vmlinux	set_normalized_timespec
vmlinux	vmlinux	ktime_get_ts
socat	socat	/usr/bin/socat
tg3.ko	tg3.ko	tg3_poll
vmlinux	vmlinux	tcp_v4_rcv
vmlinux	vmlinux	__inet_lookup_established
vmlinux	vmlinux	sk_run_filter
vmlinux	vmlinux	tcp_rcv_established
vmlinux	vmlinux	kmem_cache_alloc_node

Outline

Results: Without Postmark



Results: With Postmark



Results Summary

- Each metric best discriminates different types of faults
  - **Samples**: network-hogs from kernel-level TCP computation
  - **Count**: receive-packet-loss from socket read calls
  - **Time**: disk-hog/disk-busy from blocked I/O syscalls
- **Count** attenuated by postmark's random & uneven requests
- False-positive rate $<$ 10% for all fault types
- Instrumentation overhead (increase in workload runtime)
  - $<$ 7% (98% conf.) for all sample profiling & large I/O tracing
  - $>$ 113% (98% conf.) for function-call tracing with postmark

Outline

Future Directions

- **Analysis:** Relevance of specific functions (postmark)
  - Weigh feature vectors by component-wise variance
  - Emphasizes functions affected least by random behavior
- **Instrumentation**: Kernel-level function-call tracing
  - To better observe kernel behavior (e.g., TCP retransmits)
  - Would diagnose send-packet-loss during read workloads
- **Overhead Reduction:** Selective call site instrumentation
  - Include sites determined relevant to prior observed faults
  - Exclude sites frequently called but determined less relevant

Summary

- Behavior-based approach to problem diagnosis in PVFS
  - Illustrates use of sample profiling & call tracing in diagnosis
  - Leverages peer-comparison to identify faulty nodes
  - Enables root-cause analysis by identifying affected functions
- Diagnosis method is applicable to existing deployments
  - Sample profiling is minimally invasive, low overhead
  - Call tracing prototype works well, may be further refined
  - Fault-free training with stress tests