File System Tracing, Replaying, Profiling, and Analysis on HEC Systems

Erez Zadok, Klaus Mueller Ethan L. Miller



Stony Brook University

www.cs.stonybrook.edu

UC Santa Cruz

www.cs.ucsc.edu

STONY BROOK
STATE UNIVERSITY OF NEW YORK

Overview

- Problem: I/O dominates performance
  - Where and why I/O time is spent?
- Existing technology:
  - Trace Capturing & Replaying
  - OS and file system profiling
- Expertise in:
  - File system benchmarking
  - Data analysis and visualization
  - Code instrumentation
  - Large storage clusters
- Project goals:
  - Help identify source of I/O times
  - Low overhead
  - Useful to small and large clusters

Tracefs & Replayfs

- Versatile
  - Fine grained selective tracing/replaying
    - e.g., focus on “hot spots”
  - replay at different speeds
- Efficient
  - low overhead selective tracing
  - compression
  - localized data transofrmations
  - replay as much as 2.5x *faster*
    - runs in kernel

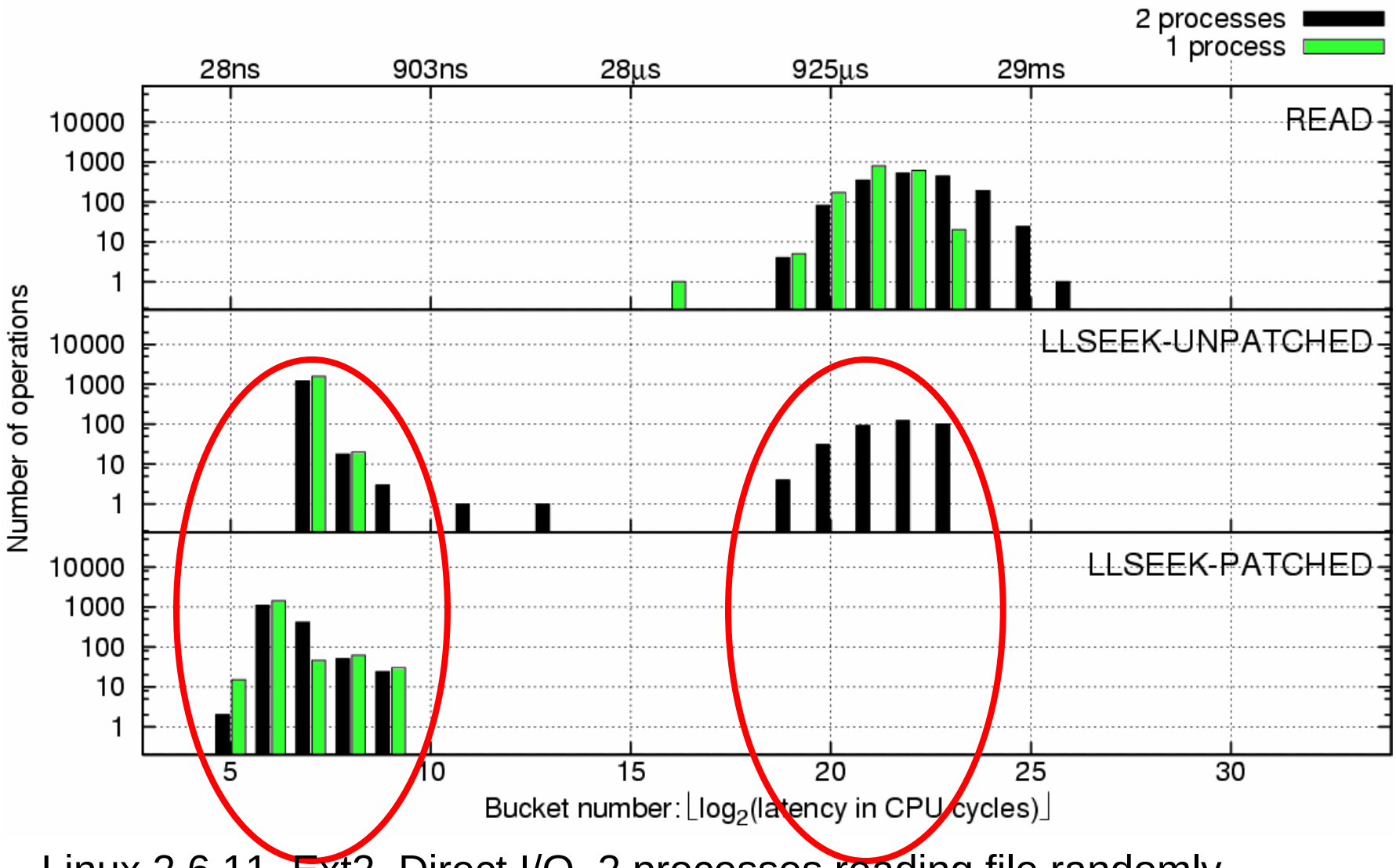
[FAST 2004, FAST 2005]

Profiling

- Standard profilers have high overheads and capture only CPU-related information.
- OSprof measures I/O latencies
  - plot in histograms with exponential buckets
- Low overhead: <4% CPU
- Small memory footprint
- Discovered many interesting performance problems
  - single-host machines
  - client-server settings (NFS/CIFS interactions)

[OSDI 2006]

Profiling Example



Linux 2.6.11, Ext2, Direct I/O, 2 processes reading file randomly

STONY BROOK STATE UNIVERSITY OF NEW YORK

Code Instrumentation

- GCC “plugin” architecture
- Works at various levels
  - CFG, GIMPLE, RTL, etc.
- Easily instrument code for any language
- Written plugins for
  - bounds checking
  - bracketing problems
    - malloc/free, lock/unlock, etc.
  - Monte Carlo based statistical verification

Visual Analytics

- Large clusters can produce a lot of trace and profile data
- Reduce data at source as much as possible
  - early data analysis
- Compare sets of profiles
  - Earth Mover's Distance, chi-suqared, etc.
- Visualize data interactively
  - Human visual system highly effective

Putting It All Together

1. Instrument code with profilers
   - distributed HEC applications
   - operating system components
   - other user-level services
2. Capture traces + profiles
3. Perform local data analysis/reduction
4. Log trace/profile data over time
5. Collect and visualize data to user
   - real time interaction or by replaying past logs
   - compare successive application runs
   - Correlate profiles with source code
   - Dynamic feedback/control of
     - instrumentation, trace/profile level

Help Us to Help You…

- Talk to
  - Ethan Miller
  - Klaus Mueller
  - Erez Zadok