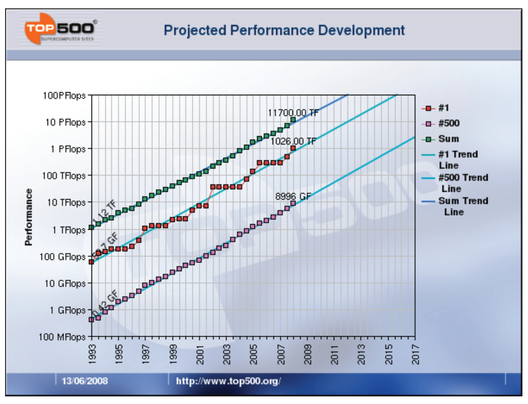
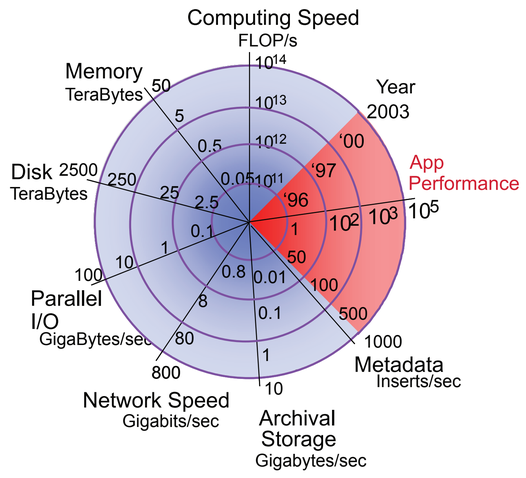
dBug: Systematic Evaluation of Distributed Systems

Jiri Simsa, Garth Gibson, Randy Bryant

Problem Setup

- HPC computing speed grows 2X per year
- Disk bandwidth grows only 20% per year
- Random access rate grows only 7% per year
- As a result, parallel FSs grow in:
  - disks, parallelism, prefetching, delaying
- Implementing and stabilizing more complex code at HPC scales is harder each year





Background: Proving Correctness

- Model checking tools in use:
  - MaceMC, SLAM, HAVOC, Terminator, SPIN, Slayer, ...
- No one-fits-all tool
- Typical limitations:
  - Limited range of properties / language constructs
  - Manual effort to annotate / specify / verify required
  - Proves correctness under assumptions

Background: Bug Finding

- Concrete / Symbolic execution tools in use:
  - MoDist, KLEE, eXplode, DART, VeriSoft, ...
- Execute real code in a test harness
- Typical limitations:
  - Under-constraining environment
  - Limited ability to setup test cases

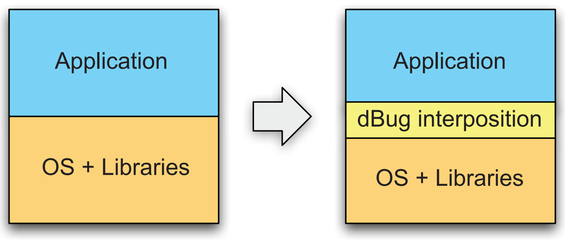
Distributed System Verification Challenges

- Number of behaviors grows exponentially with number of concurrent events
- Non-determinism arises due to interactions with environment
- Conflicting requirements, such as performance versus consistency, imply high complexity
- (Formal) Specifications of distributed systems inaccurate, outdated, or non-existent

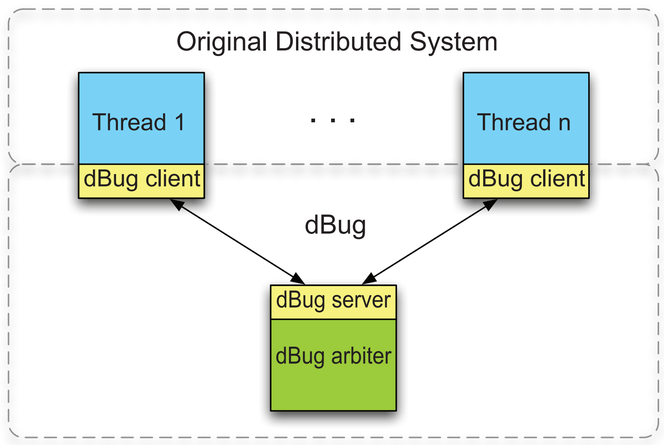
Case Studies: PVFS and FAWN

- Our Goal: systematic evaluation of distributed systems
- Initial cases: PVFS and FAWN-based distributed key-value store
- Integration of both systems with dBug ~ 100 LOC
- Additional annotations used for:
  - Accelerating exploration of dBug
  - Reducing the number of behaviors needed to explore
- Verified 10 workloads; found known & unknown bugs
  - E.g., a FAWN bug was violating strong consistency guarantee

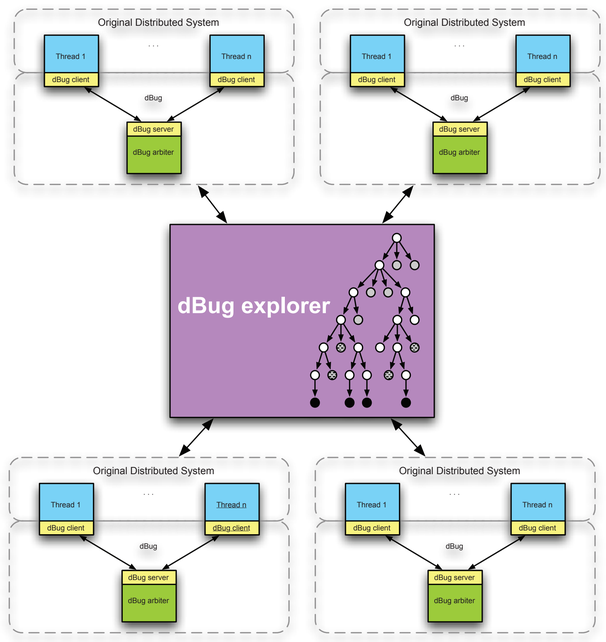
dBug Approach: Design & Implementation



Avoid annotation of source code by interposing on popular library interfaces



Use centralized arbiter to control execution order of concurrent events



Through arbiter scheduling, systematically explore different execution orders

Carnegie Mellon University

pdsi

Parallel Data Laboratory

PDLR10