PAIO

General, Portable I/O Optimizations With Minor Application Modifications

Ricardo Macedo, Yusuke Tanimura[★], Jason Haga[★], Vijay Chidambaram[▼], José Pereira, João Paulo

INESC TEC and University of Minho [★]AIST [▼]UT Austin and VMware Research

1 CHALLENGES AND MOTIVATION

EXISTING I/O OPTIMIZATIONS ARE IMPLEMENTED IN SUB-OPTIMAL MANNER:

⊗ Tightly coupled optimizations
- Optimizations are single-purposed
- Require deep understanding of the system's internal operation model
- Require profound system refactoring

✓ Decouple optimizations
- Disaggregated from the system's logic
- Generally applicable and portable

⊗ Rigid interfaces
- Existing interfaces are predefined and do not consider application-level logic
- Discard information that could be used to classify and differentiate requests with different levels of granularity

✓ Information propagation
- Propagate application-level information down the I/O stack

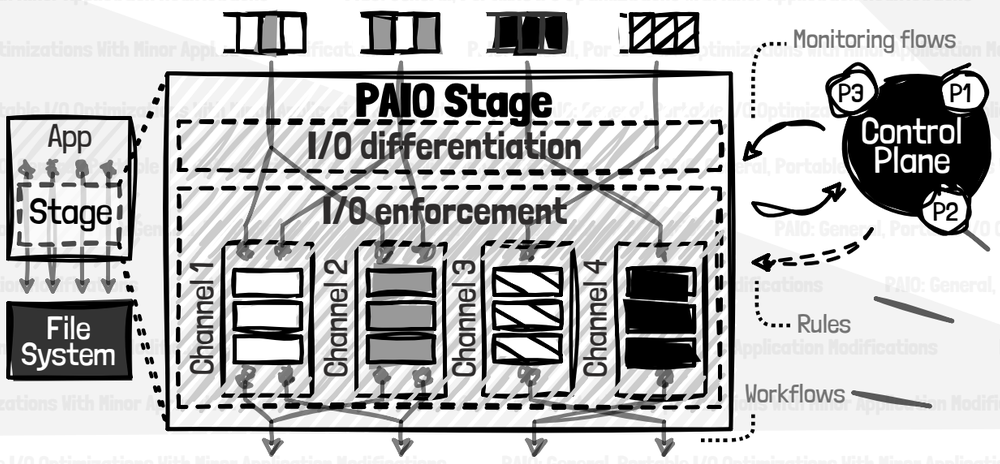
⊗ Kernel-level layers
- Impossible to extend without breaking APIs
- More restricted and error prone environment
- Ineffective under kernel-bypass stacks

✓ Actuate at user-level
- Dedicated user-level layer
- Promotes portability
- Eases information propagation

⊗ Partial visibility
- Optimizations act in isolation
- Oblivious to other systems, competing for shared resources
- Lack of coordination

✓ Global I/O control
- Operate in coordination
- Ensure holistic control of I/O workflows

2 PAIO IN A NUTSHELL



PAIO IS A USER-LEVEL FRAMEWORK THAT ENABLES BUILDING PORTABLE AND GENERALLY APPLICABLE STORAGE OPTIMIZATIONS

General applicability
- Stages are disaggregated from internal system logic, ensuring applicability across different I/O layers

Programmable building blocks
- Decoupled design that separates I/O mechanisms from policies
- Provides abstractions for building new storage optimizations

Fine-grained I/O control
- Classifies, differentiates, and enforces I/O requests with different levels of granularity

Stage coordination
- Exposes a control interface that enables the control plane to dynamically adapt stages to new policies and workloads

Low intrusiveness
- Porting I/O layers to use PAIO requires none to minor code changes

3 DIFFERENTIATION & ENFORCEMENT

Context propagation
- PAIO combines ideas from context propagation and applies them to ensure fine-grained control over I/O requests
- Context propagation enables a system to forward application-level information along its execution path

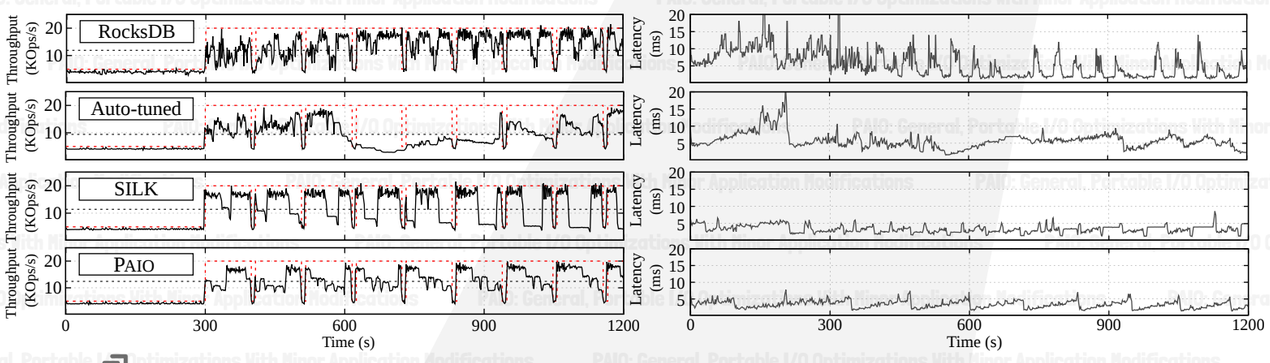
Channel and enforcement object-level I/O differentiation
- To classify requests, PAIO creates Context objects that contain the metadata that characterize a given I/O request
- Examples of such classifiers include the workflow-id, request type, size, and request context

Enforcement of I/O mechanisms over requests
- Provides building blocks for developing I/O mechanisms to be employed over I/O requests
- Noop implements a pass-through mechanism that copies the request's content to a Result object
- Dynamic Rate Limiter implements a token-bucket to control the rate and burstiness of I/O workflows

Transparent interception of I/O calls
- PAIO uses LD_PRELOAD to intercept calls to shared libraries (e.g., libc) and route them to the data plane stage
- Enables I/O layers to use PAIO without changing any line of code

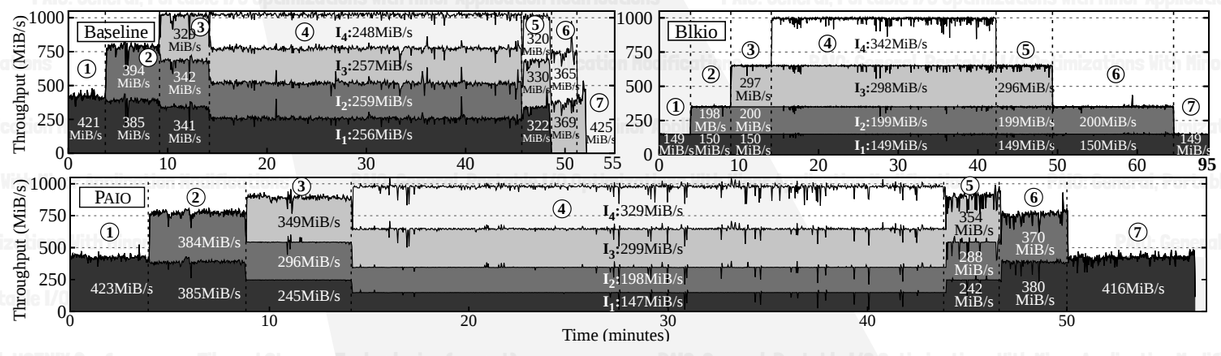
4 TAIL LATENCY CONTROL IN KVS

- We implemented a PAIO stage to prevent latency spikes in RocksDB
- The stage replicates SILK's I/O scheduler by (1) dynamically allocating bandwidth to internal operations, and (2) prioritizing flushes and low level compactions
- Integrating PAIO in RocksDB only required adding 85 LoC
- PAIO improves RocksDB's 99th percentile latency by 4x and enables similar control and performance as SILK



5 PER-APP. BANDWIDTH CONTROL

- We implemented a PAIO stage to achieve dynamic per-application bandwidth guarantees under shared-storage at the ABCI supercomputer
- Each stage controls the workflows' rate through a max-min fair share algorithm
- Integrating PAIO in TensorFlow did not required any code changes
- PAIO ensures that policies are met at all times, and whenever leftover bandwidth is available, PAIO shares it across active instances



FAST'22 INESCTEC AIST TEXAS vmware

ACKNOWLEDGMENTS. WE THANK AIST FOR PROVIDING ACCESS TO COMPUTATIONAL RESOURCES OF ABCI. THIS WORK WAS SUPPORTED BY THE PORTUGUESE FOUNDATION FOR SCIENCE AND TECHNOLOGY AND THE EUROPEAN REGIONAL DEVELOPMENT FUND, THROUGH THE PHD FELLOWSHIP SFRH/BD/146059/2019 AND PROJECTS BIGHPC (POCI-01-0247-FEDER-045924) AND PASTOR (UTA-EXPL/CA/0075/2019).

dsrhaslab/paio
rgmacedo@inesctec.pt
https://www.usenix.org/conference/fast22/presentation/macedo