dLABEL:
A Label-Based Data Representation & its Application on Big Data I/O Systems

Xian-He Sun
Illinois Institute of Technology

Keynote Speech at HPBD&IS International Conference, May 23, 2020
The Data Access Bottleneck

- Super-parallel Computers, Accelerators, Big Data applications
- Many data requests
- Many different data requests

The Memory Wall & I/O Wall

- Many small data requests
System Complicity: Hierarchy, Concurrent, Remote, Heterogeneity

- Multi-core
- Multi-threading
- Multi-issue
- Multi-banked Cache
- Multi-level Cache
- Multi-channel
- Multi-rank
- Multi-bank

CPU

Cache 1
Cache 2
Cache 3

Memory

Out-of-order Execution
Speculative Execution
Runahead Execution

Pipelined Cache
Non-blocking Cache
Data Prefetching
Write buffer

Pipeline
Non-blocking
Prefetching
Write buffer

Input-Output (I/O) Parallel File System

Disks
<table>
<thead>
<tr>
<th>Feature</th>
<th>Data Requirement</th>
<th>HPC</th>
<th>Cloud/bigdata</th>
<th>Optimizations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data consistency</td>
<td>Data must be consistent between operations.</td>
<td>Strong, POSIX</td>
<td>Eventual, Immutable</td>
<td>Tunable consistency</td>
</tr>
<tr>
<td>Data access</td>
<td>Multiple processes must be able to operate on the same data concurrently.</td>
<td>Shared concurrent</td>
<td>Multiple replicas</td>
<td>Concurrent handlers, Complex locks, Collective access</td>
</tr>
<tr>
<td>Global namespace</td>
<td>Data identifiers must be resolved and recognizable in a global namespace that can be accessed from anywhere.</td>
<td>Hierarchical, Directory, Nesting</td>
<td>Flat</td>
<td>Namespace partitioning, Cache &amp; Client-side caching, Decoupling data-metadata, Connectors</td>
</tr>
<tr>
<td>Fault tolerance</td>
<td>Data must be protected against faults and errors.</td>
<td>Special hardware, check-pointing</td>
<td>Replication, Data partitioning</td>
<td>Erasure coding</td>
</tr>
<tr>
<td>Scale</td>
<td>Support for extreme scale and multi-tenancy</td>
<td>Few large jobs, Batch processing</td>
<td>Many small jobs, Iterative</td>
<td>Job scheduling, buffering, Scale-out</td>
</tr>
<tr>
<td>Locality</td>
<td>Jobs are spawned where data reside.</td>
<td>Remote memory &amp; storage</td>
<td>Node local</td>
<td>Data aggregations</td>
</tr>
<tr>
<td>Ease of use</td>
<td>Interface, user-friendliness and ease of deployment</td>
<td>High-level libraries</td>
<td>Simple data formats</td>
<td>Amazon S3, Openstack Swift</td>
</tr>
</tbody>
</table>
Challenges of a System Solution

- System Malleability
- Asynchronous Request
- Resource Heterogeneity
- Data Provisioning
- Storage Bridging
A out-of-box thinking

How retailer giants handle their online order today?
A simple analogy of shopping

• **Sending a gift (before)**
  - Drive to three different retailers
  - Purchase items independently
  - Decide what package to use
  - Decide which delivery provider
  - Decide options (priority, etc)

• **Accessing data (today)**
  - Three different data sources
  - Acquire data elements
  - Data representation (file, object, etc)
  - Storage device (SSD, HDD, etc)
  - Storage semantics
A simple analogy of online shopping

• Sending a gift via an online retailer
  • Add items to cart
  • Specify details (payment info, etc.)
  • Submit order
  • The gift is shipped from the warehouse

• Similar data management
  • Create labels for data
  • Define label attributes (feature, operation, etc.)
  • Push labels to queue
  • Data operation is carried at the data Warehouse
The Data Label Representation

- Location-independent abstraction expressing data intent.
- A tuple of one or more operations and a pointer to the physical data.
- Exclusive to each application.
- Immutable, independent of one another, and cannot be re-used.
- **Data Label structure** includes:
  - Type
  - Unique identifier
  - Source and destination
    - memory address, file path, server IP, network port
  - Function pointer (user-defined or pre-defined)
    - all functions are store in a shared program repository
  - Set of flags indicating label’s state
    - queued, scheduled, pending, cached, invalidated, etc.,
Data requests are transformed into a configurable unit, called a (data) **Label**.
- A label is a tuple of an operation and a pointer to the data.
- Resembles a shipping label following a Post Office package.

Labels are pushed to a distributed queue.
- Data or contents are pushed into a warehouse.
- A dispatcher distributes labels to the workers.
- Workers execute labels independently (i.e., fully decoupled).
The Key Question

Don’t have good ideas if you aren’t willing to be responsible for them.

Alan Perlis

Our Solution

LABIOS: A Label-Based I/O System
LABIOS: Label-Based I/O System

- Distributed, scalable, and adaptive storage solution
- Fully decoupled architecture
- Software defined storage (SDS)
- Energy-aware enabling power-capped I/O
- Reactive storage with tunable I/O performance
- Flexible API
- Intersection of HPC and BigData
Software Stack

- Can be used as:
  - Middleware I/O library
    - via LABIOS API
  - Full-stack storage solution
    - via I/O call interception
- LABIOS can unify multiple namespaces by connecting to external storage systems. (storage bridging)
High-level Architecture

- Two main ideas:
  1. Split the data, metadata, and instruction paths.
  2. Decouple storage servers from the application.
**LABIOS Client**

- **Objectives:**
  1. Performs system initialization per-application.
  2. Accepts application’s I/O requests.
  3. **Builds labels** based on the incoming I/O.

- **Modules:**
  1. Label manager
  2. Content manager
  3. Catalog manager

---

1/11/2021
LABIOS Core

- Manages the instruction, data, and metadata flow separately.
- Distributed data structures:
  - Label queue
  - Warehouse
- Modules:
  - Administrator
  - Label Dispatcher: Optimization and scheduling
LABIOS Server

- Manages workers (i.e., storage servers)
- Modules:
  - **Worker**
    - Logical entity who carries the work
  - Worker manager
• The storage server in LABIOS
• Responsibilities:
  • service its own queue
  • execute labels
  • calculate its own worker score and send it to the worker manager periodically
  • auto-suspend itself if there are no labels in its queue for a given time window
  • connect to external storage sources
• Weighting system expresses the scheduling policy
• Final score is a double precision between 0 and 1
  • Higher score -> better worker

\[
Score(\text{workerID}) = \sum_{n=1}^{5} \text{Weight}_j \times \text{Variable}_j
\]

<table>
<thead>
<tr>
<th>Priority</th>
<th>Availability</th>
<th>Capacity</th>
<th>Load</th>
<th>Speed</th>
<th>Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low latency</td>
<td>0.5</td>
<td>0</td>
<td>0.35</td>
<td>0.15</td>
<td>0</td>
</tr>
<tr>
<td>Energy savings</td>
<td>0</td>
<td>0.15</td>
<td>0.2</td>
<td>0.15</td>
<td>0.5</td>
</tr>
<tr>
<td>High bandwidth</td>
<td>0</td>
<td>0.15</td>
<td>0.15</td>
<td>0.70</td>
<td>0</td>
</tr>
</tbody>
</table>
LABIOS in depth

- Label Scheduling
- Deployment Models
- LABIOS API example
• Data distribution via scheduling policies:
  1. Round Robin
     • similar with PFS
  2. Random Select
     • randomly select workers
  3. Constraint-based (i.e., heuristically)
     • Priorities based on higher weight in the worker score
       • Availability, load, capacity, performance
  4. MinMax
     • Minimize energy consumption while maximizing performance
       • Subject to load and capacity
       • NP-hard combinatorial optimization
         • Multidimensional knapsack algorithm
LABIOS can:
1. replace existing distributed storage solutions
2. be used as I/O accelerator to one or more underlying storage subsystems

Machine model in use (motivated by the recent machines Summit in ORNL or Cori on LBNL):
- Compute nodes equipped with a large amount of RAM
- Local NVMe devices in each compute node
- An I/O forwarding layer
- A shared burst buffer installation based on SSD equipped nodes, and
- A remote PFS installation based on HDDs
LABIOS as I/O accelerator (in compute nodes)

**Pros**
- Fast distributed cache
  - For temporary I/O
  - On top of external sources
- Hadoop workloads with node local I/O

**Cons**
- Overheads by using compute cores to run its services
- I/O traffic mixed with compute network
LABIOS as I/O forwarder (in ION)

- **Pros**
  - Asynchronous I/O
    - Non-blocking data movement
  - Connect to external storage

- **Cons**
  - Subject to I/O forwarding layer
  - Limited scalability
Pros
- Fast scratch space
- Data sharing between applications
- In-situ visualization and analysis

Cons
- Requires additional resources (e.g., buffers)
  - Storage
  - Network
LABIOS as Remote Distributed Storage

Pros

- Scalability
- Better resource utilization
- Higher flexibility

Cons

- Increased deployment complexity
- Requires systems admins
• All experiments on bare metal on Chameleon:
  • 64 client nodes
  • 8 burst buffer nodes
  • 32 storage servers
• Cluster OS: CentOS 7.1
• PFS: OrangeFS 2.9.6
• Workloads:
  • CM1 simulation
  • HACC simulation
  • Montage application
  • K-means clustering
Storage Malleability

• Metric: Total I/O time in sec
• 4096 labels of 1MB each
• Vary the ratio of active – suspended workers
• Worker activation in 3 sec on average
• Worker allocation techniques
  • Static: labels only on active workers
  • Elastic: labels to all workers (even on suspended paying the penalty of activation)
• When small % of workers are active, elastic boosts performance
• When enough workers are active, activation latency hurts performance
I/O Asynchronicity

- Metric: Overall execution time in sec
- Support of both sync – async modes
- Label paradigm fits (naturally) in async
- CM1 simulation scaled up to 3072 processes with 16 time steps
- Each process writes 32MB of I/O
  - 100GB per step for the 3072 case
- Sync mode competitive with PFS baseline
- Async mode overlaps label execution with computations
  - 16x boost in I/O performance
  - 40% reduction in execution time
Resource Heterogeneity

- Metric: Overall execution time in sec
- HACC simulation scaled up to 3072 processes with 16 time steps
- Update-heavy workload
  - Each process updates 32MB of I/O
  - Checkpoint in burst buffers
  - Final flush of last checkpoint data to PFS
- 6x improvement in I/O performance
- Flushing in the background from workers
Data Provisioning

- Metric: Overall execution time in sec
- Montage application
  - Multiple executables that share data
- 50GB of intermediate results in temporary files in PFS
- LABIOS shares data via the warehouse (i.e., in-memory)
  - Label destination is analysis compute nodes
- Performance acceleration
  - No temporary files are created in remote storage
  - Simulation and analysis can be pipelined
- **17x boost** in I/O performance
- **65% reduction** in execution time
• Metric: Overall execution time in sec
• Two modes for LABIOS:
  • Node-local I/O (similar to HDFS)
  • Remote external I/O (similar to HPC)
• Map processes read 32MB each and then write them back to storage
• Reduce processes read 32MB each
• Shuffle sends 32MB through network
• Hadoop-memory optimized version
  • No disk I/O for intermediate results
• LABIOS employs collective I/O to perform data aggregations
• LABIOS successfully integrates MapReduce with HPC
LABIOS: A Successful Example of the Data Label Approach

LABIOS provides storage flexibility, versatility, and agility due to a new data model, the data labels and its decoupled data-centric architecture; provide a new way to support:

- Storage malleability
- Asynchronous I/O
- Resource Heterogeneity
- Data Provisioning
- Storage Bridging

LABIOS can boost I/O performance on certain workloads by up to 17x and reduce overall execution time by 40-60%.

Its potential is not fully explored
Take Home Questions:

- What is the next of LABIOS?
- What is the next of Data Label?

The Key Contribution

Data Label has separated the control flow with the data flow

Future Work

- Use LABIOS for data scheduling
  - HPC center
  - Data center
  - Cloud environment
- Establish a LABIOS compliant file system
- Extend LABIOS to a general software-defined IO system/storage
- Extend dLABEL to memory systems
Thank you
Any questions?


We would like to thank
A Kougka, H. Devarajan, K. Bateman, J. Cernuda, F. Ku, L. Logan, ....

Find more at:
www.cs.iit.edu/~scs
www.akougkas.com/research/labios