Task-Based I/O System

Anthony Kougkas
akougkas@hawk.iit.edu
# Background

## Resource Utilization
- Periodic I/O creates idleness between phases
- Over-provisioning due to ignorance or malicious intent
- Exclusive access reservations
- 24/7 always on

**Under - Over resource utilization**

## Interface/API
- Tightly-coupled to certain APIs
- Bound to comply with deployed underlying storage
- Vendor-specific solutions
- Expensive connectors

**Isolation, less flexibility and lower productivity.**

## Performance/Energy
- Predefined static storage resources
- No support for power-cap I/O
- No support for tunable performance features

**No adaptive concurrency Resource heterogeneity**
Motivation

**Challenge 1: Resource Utilization**
How to efficiently utilize I/O resources?
- Performance boost
- System efficiency
- Monetary benefits

**Challenge 2: Interface/API**
How to support a wide range of I/O interfaces?
- Productivity
- Flexibility
- Compatibility

**Challenge 3: Performance/Energy**
How to balance energy - performance?
- Energy savings
- Superior scalability
- Increased control
Introducing TABIOS
An Adaptive, Elastic, Energy-Aware, Distributed Task-based I/O System

- Applications create I/O tasks: **DataTasks**
- A **DataTask** is a placeholder of an I/O job: {operation + pointer to data}
- DataTasks are pushed in a distributed queue
- Workers execute DataTasks independently
Objectives

- **Storage malleability,**
  - resources can grow/shrink based on the workload.
- **Asynchronous I/O,**
  - operating with mixed media and various configurable storage options.
- **Resource heterogeneity,**
  - supporting a variety of storage resources under the same platform.
- **Data provisioning,**
  - enabling in-situ data analytics and process-to-process data migration.
- **Storage consolidation,**
  - supporting a diverse set of conflicting I/O workloads on a single platform.
**TABIOS Architecture**

- **Agile**
  - Adaptive to the environment
  - Fully decoupled architecture

- **Software Defined Storage**
  - Offloading computation to servers
  - Data-centric architecture

- **Energy-aware**
  - Power-cap I/O
  - Elastic I/O resources

- **Reactive**
  - Tunable I/O performance - Concurrency control
  - Guaranteed Storage QoS based on job size

- **Flexible**
  - POSIX, MPI-IO, HDF5, REST/Swift, Hadoop
  - Lustre, HDFS,Object Stores
Scheduling Policies

1. Random selection of worker
2. Round Robin
3. Load (queue size)
4. Capacity (size in GB)
5. Latency (access in ms)
6. Locality-aware (file location)
```cpp
#include <tabios.hpp>
...
Client client = InitClient(ip, port, connConfig);
std::string path = "pvfs2://data/integers.dat";
std::size_t pos = 0, size = 200MB;
DataTaskSrc src = new DataTaskSrc(path, pos, size);
DataTaskType type = SDS_IN_SITU; //complex type
DataTaskFlag flags = CACHE, | MPI_IO; //keep in cache
std::function<int(vector<int>)> fn = FindMedian;
DataTask datatask = client.CreateDataTask(type, src, fn, flags);
Status status = client.IPublishDataTask(datatask);
... //perform other computations
client.WaitDataTask(&status);
int median = std::static_cast<int>(status.data);
```
Anatomy of Operations

Write:
- Write Data: 29%
- Get Data: 17%
- Update MDM: 17%
- Schedule: 13%
- Subscribe: 13%
- Publish: 9%
- Put Data: 11%
- Build Task: 2%

Read:
- Read Data: 27%
- Send Data: 20%
- Publish: 15%
- Subscribe: 12%
- Update MDM: 11%
- Get Data: 9%
- Query MDM: 8%
- Build Task: 6%
- Publish Task: 5%
- Subscribe: 4%
- Schedule: 3%

TABIOS Client
Datatask Scheduler
TABIOS Worker
Thank you

Anthony Kougkas
akougkas@hawk.iit.edu
www.akougkas.com