Practical Persistent Volumes
Features for supporting storage frameworks

Anindya Sinha and Yan Xu
Agenda

• Shared Persistent Volumes
  • Overview
  • Share and Unshare call flows
  • Resources abstraction
  • Allocator and sorter
  • Considerations for Frameworks
  • Current Status and Future Work
  • Demo

• Permissions for persistent volumes
  • The Problem
  • Ownership for persistent volumes
  • Access by tasks
Agenda

• Shared Persistent Volumes
  • Overview
  • Share and Unshare call flows
  • Resources abstraction
  • Allocator and sorter
  • Considerations for Frameworks
  • Current Status and Future Work
  • Demo

• Permissions for persistent volumes
  • The Problem
  • Ownership for persistent volumes
  • Access by tasks
Regular Resources

Scheduler

Master

Task 1
Task 2
Task 3
Task 4
Task 5

Agent

x1Ks

x5

Not a contribution
Shared Resources

Scheduler x5
Master x5
Agent
Task 1
Task 2
Task 3
Task 4
Task 5
x1Ks

Not a contribution
A Typical Use Case

Database Tasks

Backup Tasks

Physical Storage

Not a contribution
Shared Resources: What and Why?

- Regular resources are available to a single task only at a given point of time.
- Shared resources can be assigned to **multiple tasks simultaneously**.
- Shared resources associated with an identity.
- Switch between shared and unshared modes for resources.
- Frameworks opt in for shared resources.
- Shared resources enabled for persistent volumes.
- Tasks get access to shared volumes with appropriate access ("read-only" vs "read-write").
Alternatives considered

- Out of band storage
- Combining tasks
- Tasks in the same container/executor
- Sub containers in a pod

- Ease of use
- Decoupling
- Flexibility

Introduced and generalized concept of shared resources across various resource types.
Potential resource types

• Persistent volumes
• Non-persistent volumes
• Non-local storage
• Custom resource types
Create Shared Volume

Framework

OFFER
- cpus: 24
- mem: 128G
- disk: 1024G

Master

OFFER
- cpus: 24
- mem: 128G
- disk: 1024G
Create Shared Volume

Framework

- OFFER
  - cpus: 24
  - mem: 128G
  - disk: 1024G

Master

- RESERVE
  - disk(db): 768G

RESERVE
disk(db): 768G
Create Shared Volume

Framework

OFFER
- cpus: 24
- mem: 128G
- disk: 1024G

RESERVE
- disk(db): 768G

Master

OFFER
- cpus: 24
- mem: 128G
- disk: 256G
- disk(db): 768G

OFFER
- cpus: 24
- mem: 128G
- disk: 256G
- disk(db): 768G
Create Shared Volume

Framework

OFFER
cpus: 24
mem: 128G
disk: 1024G

RESERVE
disk(db): 768G

Master

OFFER
cpus: 24
mem: 128G
disk: 256G
disk(db): 768G

CREATE
disk(db)[id:path]
<SHARED>: 768G

CREATE
disk(db)[id:path]
<SHARED>: 768G
Create Shared Volume

Framework

- OFFER
cpus: 24
mem: 128G
disk: 1024G

- RESERVE
disk(db): 768G

- OFFER
cpus: 24
mem: 128G
disk: 256G
disk(db): 768G

- CREATE
disk(db)[id:path] <SHARED>: 768G

Master

- OFFER
cpus: 24
mem: 128G
disk: 256G
disk(db)[id:path] <SHARED>: 768G

Not a contribution
Launch Tasks on Shared Volume

Framework

LAUNCH

cpus: 2
mem: 16G
disk(db)[id:path]
<SHARED>: 768G

Task1 Launch

Master

LAUNCH

cpus: 2
mem: 16G
disk(db)[id:path]
<SHARED>: 768G
Launch Tasks on Shared Volume

Framework

LAUNCH
- cpus: 2
- mem: 16G
- disk(db)[id:path]
- <SHARED>: 768G

Master

OFFER
- cpus: 22
- mem: 112G
- disk: 256G
- disk(db)[id:path]
- <SHARED>: 768G

Task1 Launch

Shared volume still offered
Launch Tasks on Shared Volume

Framework

- LAUNCH
cpus: 4
mem: 8G
disk: 16G
disk(db)[id:path] <SHARED>: 768G

Master

- LAUNCH
cpus: 2
mem: 16G
disk: 256G
disk(db)[id:path] <SHARED>: 768G

- OFFER
cpus: 22
mem: 16G
disk: 16G
disk(db)[id:path] <SHARED>: 768G

- LAUNCH
cpus: 4
mem: 8G
disk: 16G
disk(db)[id:path] <SHARED>: 768G

Task1 Launch

Task2 Launch

Shared volume still offered

Task1 Launch

Shared volume still offered

Not a contribution
Launch Tasks on Shared Volume

Framework

Task 1 Launch

LAUNCH
- cpus: 2
- mem: 104G
- disk(db)[id:path]<SHARED>: 768G

Task 2 Launch

LAUNCH
- cpus: 4
- mem: 8G
- disk: 16G
  disk(db)[id:path]<SHARED>: 768G

Master

OFFER
- cpus: 18
- mem: 104G
- disk: 240G
  disk(db)[id:path]<SHARED>: 768G

Shared volume still offered

Shared volume offered. Again...

Not a contribution
Destroy Shared Volume

Framework

OFFER

- cpus: 20
- mem: 120G
- disk: 240G
- disk(db)[id:path]
- <SHARED>: 768G

Task 1 Terminated

Master

OFFER

- cpus: 20
- mem: 120G
- disk: 240G
- disk(db)[id:path]
- <SHARED>: 768G
Destroy Shared Volume

Framework

OFFER
cpus: 20
mem: 120G
disk: 240G
disk(db)[id:path]
<SHARED>: 768G

Task 1 Terminated

Master

DESTROY
disk(db)[id:path]
<SHARED>: 768G
Destroy Shared Volume

Task 1 Terminated

OFFER
cpus: 20
mem: 120G
disk: 240G
disk(db)[id:path]
<SHARED>: 768G

DESTROY
disk(db)[id:path]
<SHARED>: 768G

OFFER
cpus: 20
mem: 120G
disk: 240G
disk(db)[id:path]
<SHARED>: 768G

OFFER
cpus: 20
mem: 120G
disk: 240G
disk(db)[id:path]
<SHARED>: 768G

Not a contribution
Destroy Shared Volume

Framework

OFFER
- cpus: 20
- mem: 120G
- disk: 240G
disk(db)[id:path]
- <SHARED>: 768G

Task 1 Terminated

DESTROY
disk(db)[id:path]
- <SHARED>: 768G

Master

OFFER
- cpus: 20
- mem: 120G
- disk: 240G
disk(db)[id:path]
- <SHARED>: 768G

Task 2 Terminated

OFFER
- cpus: 24
- mem: 128G
- disk: 256G
disk(db)[id:path]
- <SHARED>: 768G

OFFER
- cpus: 24
- mem: 128G
- disk: 256G
disk(db)[id:path]
- <SHARED>: 768G

Not a contribution
Destroy Shared Volume

NOT A CONTRIBUTION

Task 1 Terminated

OFFER
- cpus: 20
- mem: 120G
- disk: 240G
- disk(db)[id:path]
  <SHARED>: 768G

DESTROY
- disk(db)[id:path]
  <SHARED>: 768G

OFFER
- cpus: 20
- mem: 120G
- disk: 240G
- disk(db)[id:path]
  <SHARED>: 768G

Task 2 Terminated

OFFER
- cpus: 24
- mem: 128G
- disk: 256G
- disk(db)[id:path]
  <SHARED>: 768G

DESTROY
- disk(db)[id:path]
  <SHARED>: 768G
Destroy Shared Volume

Framework

<table>
<thead>
<tr>
<th>CPUS</th>
<th>Mem</th>
<th>Disk</th>
<th>Disk (db)</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>28G</td>
<td>256G</td>
<td>768G</td>
</tr>
</tbody>
</table>

Master

<table>
<thead>
<tr>
<th>CPUS</th>
<th>Mem</th>
<th>Disk</th>
<th>Disk (db)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>20G</td>
<td>240G</td>
<td>768G</td>
</tr>
</tbody>
</table>

OFFER

<table>
<thead>
<tr>
<th>CPUS</th>
<th>Mem</th>
<th>Disk</th>
<th>Disk (db)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>20G</td>
<td>240G</td>
<td>768G</td>
</tr>
</tbody>
</table>

DESTROY

<table>
<thead>
<tr>
<th>Disk (db) [id:path]</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHARED: 768G</td>
</tr>
</tbody>
</table>

OFFER

<table>
<thead>
<tr>
<th>CPUS</th>
<th>Mem</th>
<th>Disk</th>
<th>Disk (db)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>20G</td>
<td>240G</td>
<td>768G</td>
</tr>
</tbody>
</table>

Task1 Terminated

Task2 Terminated

OFFER

<table>
<thead>
<tr>
<th>CPUS</th>
<th>Mem</th>
<th>Disk</th>
<th>Disk (db)</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>28G</td>
<td>256G</td>
<td>768G</td>
</tr>
</tbody>
</table>

DESTROY

<table>
<thead>
<tr>
<th>Disk (db) [id:path]</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHARED: 768G</td>
</tr>
</tbody>
</table>

OFFER

<table>
<thead>
<tr>
<th>CPUS</th>
<th>Mem</th>
<th>Disk</th>
<th>Disk (db)</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>28G</td>
<td>256G</td>
<td>768G</td>
</tr>
</tbody>
</table>
Share and Unshare Existing Volume
Share and Unshare Existing Volume

Framework

Master

OFFER
- cpus: 24
- mem: 128G
- disk: 256G
- disk(db)[id:path]: 768G

SHARE
disk(db)[id:path]: 768G

disk(db)[id:path]: 768G

SHARE
- disk(db)[id:path]: 768G
Share and Unshare Existing Volume

Framework

Master

OFFER
cpus: 24
mem: 128G
disk: 256G
disk(db)[id:path]: 768G

SHARE
disk(db)[id:path]: 768G

OFFER
cpus: 24
mem: 128G
disk: 256G
disk(db)[id:path]
<SHARED>: 768G

Volume now shared

OFFER
cpus: 24
mem: 128G
disk: 256G
disk(db)[id:path]
<SHARED>: 768G
Share and Unshare Existing Volume
Share and Unshare Existing Volume

Framework

Master

OFFER
- cpus: 24
- mem: 128G
- disk: 256G
disk(db)[id:path]: 768G

SHARE
disk(db)[id:path]: 768G

OFFER
cpus: 24
mem: 128G
disk: 256G
disk(db)[id:path]
<SHARED>: 768G

Volume now shared

UNSHARE
disk(db)[id:path]
<SHARED>: 768G

OFFER
cpus: 24
mem: 128G
disk: 256G
disk(db)[id:path]: 768G

Volume now unshared

OFFER
cpus: 24
mem: 128G
disk: 256G
disk(db)[id:path]: 768G

Volume now shared
Eligibility for DESTROY and UNSHARE Operations

- Shared Resource not in use by any running task.
- Shared Resource not requested by any pending task.
- Pending offers rescinded.
Resources Abstraction

- Shared count not exposed in Resource protobuf.
- Internal container for Resource protobuf.
- Reference counts for shared resources.
- Not exposed to callers directly.
- Clients do not need to manage shared counts.
- Operators and helper methods handle shared counts implicitly.

```cpp
class Resources
{
private:
    // An internal abstraction to facilitate managing shared resources.
    class Resource_
    {
    public:
        Resource_(const Resource& _resource);
        operator const Resource&() const { return resource; } 
        bool isShared() const { return sharedCount.isSome(); } 
        Option<Error> validate() const; 
        bool isEmpty() const; 
        bool contains(const Resource_& that) const;
        
        Resource_& operator+=(const Resource_& that);  
        Resource_& operator-=(const Resource_& that); 
        bool operator==(const Resource_& that) const; 
        bool operator!=(const Resource_& that) const;
    
    private:
        Resource resource; 
        Option<int> sharedCount; 
    
    
    
};

std::vector<Resource_> resources;

public:
    ... 
    size_t count(const Resource& that) const; 
};
```
DRF Sorter

- Based on sorting on role, and then based on frameworks within the same role.
- Shared resource allocated to multiple frameworks simultaneously.
- Options considered for fairness of shared resources:
  - Option 1: Shared resources not considered.
  - Option 2: Distribute shared resources based on utilization.
  - Option 3: Distribute shared resources evenly among allocated frameworks.
  - Option 4: Equal share of shared resource assigned to all allocated frameworks.
DRF Sorter

• Based on sorting on role, and then based on frameworks within the same role.
• Shared resource allocated to multiple frameworks simultaneously.
• Options considered for fairness of shared resources:
  • Option 1: Shared resources not considered.
  • Option 2: Distribute shared resources based on utilization.
  • Option 3: Distribute shared resources evenly among allocated frameworks.
  • **Option 4: Equal share of shared resource assigned to all allocated frameworks.**
Shared Resources across frameworks

• F1 and F2 not charged for shared resource.
• F3 charged fully for shared resource.
Shared Resources across frameworks

- F1 not charged for shared resource.
- F2 and F3 each charged fully for shared resource.

F1 (opt-out)
F2 (OFFER pending; no running tasks)
F3 (few running tasks)
Shared Resources across frameworks

- F1 not charged for shared resource.
- F2 and F3 each charged fully for shared resource.
Considerations for Frameworks

- High likelihood of a shared resource becoming a dominant resource.
- Avoid opt in for \texttt{SHARED\_RESOURCES} if not needed.
- \texttt{DECLINE} shared resources at the earliest; or get charged for them.
- Frameworks orchestrate \texttt{CREATE} and \texttt{DESTROY} for shared volumes.
How frameworks use shared volumes?

- **SharedInfo** in **Resource**:

  ```
  message Resource {
    ...
    // Allow the resource to be shared across tasks.
    message SharedInfo {}  // If this is set, resources are shared among multiple tasks.
    // Currently, this is enabled for persistent volumes only.
    optional SharedInfo shared = 10;
  }
  ```

- **Framework capability**:
  - Opt in capability **SHARED_RESOURCES**.
  - Frameworks need to register with this capability to receive shared resources in offers.

- **Additional Offer Operations**:
  - **SHARE**
  ```
  message Share {
    repeated Resource resources = 1;
  }
  ```
  - **UNSHARE**
  ```
  message Unshare {
    repeated Resource resources = 1;
  }
  ```
Offer Cycle

• Shared Resources are candidate in offers of every offer cycle.
• Shared Resource added as allocated resource when offers are sent out.
• Shared resource remains allocated to a framework till:
  • Offer declined; or accepted without launching tasks using this shared resource
  • No running tasks from this framework using this shared resource.
Offer Cycle

- Offer ACCEPTed and tasks launched using shared resource:
  - Shared resource remains allocated, and is not returned back to the allocator.

- Offer ACCEPTed but no task launched using shared resource, or Offer DECLINEd:
  - No resultant change in allocated shared resources.
  - Shared Resource returned back to allocator.

- Number of copies of shared resources allocated represents the number of tasks.
Summary

• Current Status:
  • Sharing of resources by maintaining shared count internally.
  • Sharing only enabled for persistent volumes.
  • Tasks access shared persistent volumes as “read-only” or “read-write”.
  • Opt in capability for frameworks.

• Potential Enhancements:
  • Switching sharedness of resources via SHARE and UNSHARE offer operations.
  • Enable sharedness of other resource types.
  • Limit number of consumers of shared resources.
  • Operator endpoints for share and unshare operations.
Agenda

• Shared Persistent Volumes
  • Overview
  • Share and Unshare call flows
  • Resources abstraction
  • Allocator and sorter
  • Considerations for Frameworks
  • Current Status and Future Work
  • Demo

• Permissions for persistent volumes
  • The Problem
  • Ownership for persistent volumes
  • Access by tasks
Permissions for Persistent Volumes

• Current Behavior:
  • CREATE of persistent volume sets ownership of the volume based on the slave process.
  • On task launch, the ownership of the volume moves to the task.

• Problem:
  • Tasks are consumers of persistent volume.
  • Tasks “own” or “adhere” to persistent volumes.
  • Shared persistent volumes have multiple tasks accessing the volume simultaneously.

• Solution:
  • Persistent volumes should possess discrete ownership.
  • Tasks allowed access to the volume only if they have compatible ownership.
Ownership of Persistent Volumes

- Associate a persistent volume with its owner (i.e. user).

```plaintext
message DiskInfo {
  message Persistence {
    required string id = 1;
    optional string principal = 2;
    optional string user = 3;
  }

  optional Persistence persistence = 1;
  optional Volume volume = 2;
  optional Source source = 3;
}
```

- Create persistent volume on the agent:
  - Ownership set to the **user**; or inherit the ownership from the slave process.
  - Allow multiple readers and multiple writers simultaneously.
Task Access to Persistent Volumes

• Track current consumers of persistent volume.
• Tasks access persistent volume only if it has compatible ownership as the volume.
• For the 1st consumer:
  • If ownership set, allow task to access the volume if it has compatible ownership.
  • If ownership not set, ownership of the volume is set based on ownership of the task.
• For subsequent consumers:
  • Ownership of the volume is already set.
  • Allow task to access the volume if it has compatible ownership.
References

• Shared Persistent Volumes:
  • MESOS-3421: Epic for shared resources
  • Design document for shared resources

• Ownership of Persistent Volumes:
  • MESOS-4893: Allow setting permissions and access control of persistent volumes
  • Design document for permissions of persistent volumes
Thank You