Infusing Pub-Sub Storage with Transactions

Liana V. Rodriguez, John Bent, Tim Shaffer, Raju Rangaswami

Background

- Storage systems are complex and any new feature addition requires significant modifications to their codebase.
- Examples of frameworks that allow incorporating new features with little effort are the vnode API and stackable file systems.
- Other work such as ABLE[1] enables extensions at the block layer and encapsulates storage functionality in the form of a library.

[1] ”The Case for Active Block Layer Extensions” ACM Operating System Reviews, 2006

Properties of Extensible Storage Systems

	Storage access	Transactional guarantees	NO Read/Write amplification	Independent Scale and deployment	Ease of development
Integrated	✓	✓	✓	X	XXX
Interposed IO	✓	X	✓	✓*	X
Pub-Sub	X	X	X	✓	✓
FDMI	✓	✓	X	✓	✓

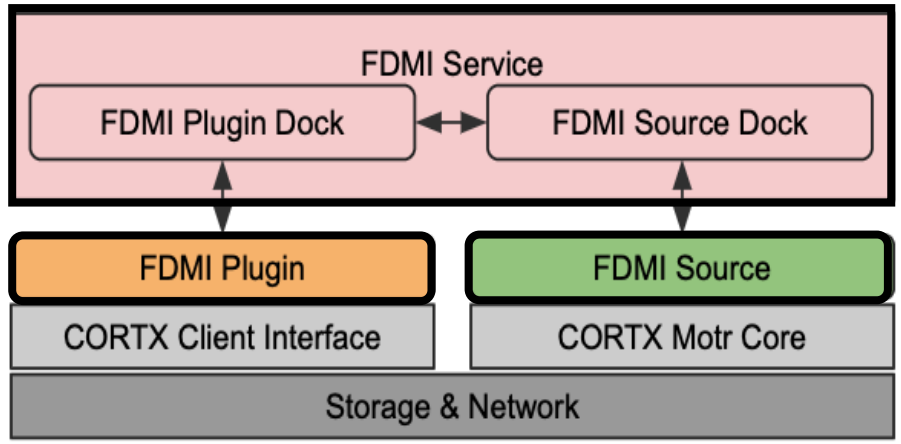
* Frameworks such as ABLE are tightly coupled and therefore do not scale independently.

Extensible Storage: FDMI in CORTX Motr

The **FDMI Source** is part of the storage server and is the only entity that manipulates the storage system.

The **FDMI Source Dock** communicates with FDMI source using source specific record functions.
The **FDMI Plugin Dock** interacts with each plugin and is responsible for registering and de-registering plugins.

The **FDMI Plugin** implements features we want to incorporate into the storage system.



Transactional Coupling

- FDMI plugins subscribe to client operations and potentially make storage system changes in response.
- Plugin actions often depend on the state of storage.
- Transactional coupling executes plugin operations atomically simultaneously with or upon completion of a client-initiated operation.
- Plugin actions stay consistent with the state of the storage system.

Plugin Classes

- **Class A**

  Plugins get notified of committed transactions reliably and do not add or modify anything in the source storage system.

  - *I/O profiling*

- **Class B**

  Plugins get notified of committed transactions, and in response can generate additional CORTX transactions that are guaranteed to commit if the plug-in runs successfully.

  - *Semantic Enhancer, Async Compression*

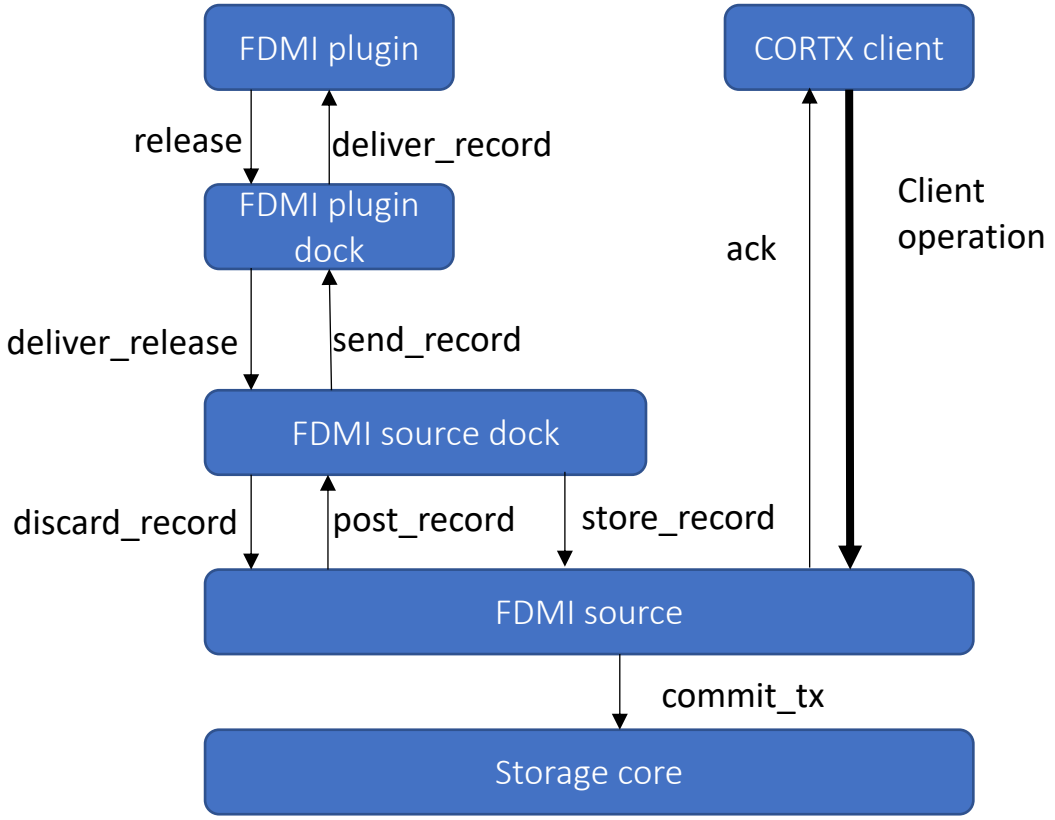
- **Class C**

  Plugins get notified of source transaction operations prior the source transaction commit so they can update and commit transactions collaboratively with the client.

  - *Dynamic Tiering, Caching, Inline Compression, Inline Encryption and Inline Deduplication*

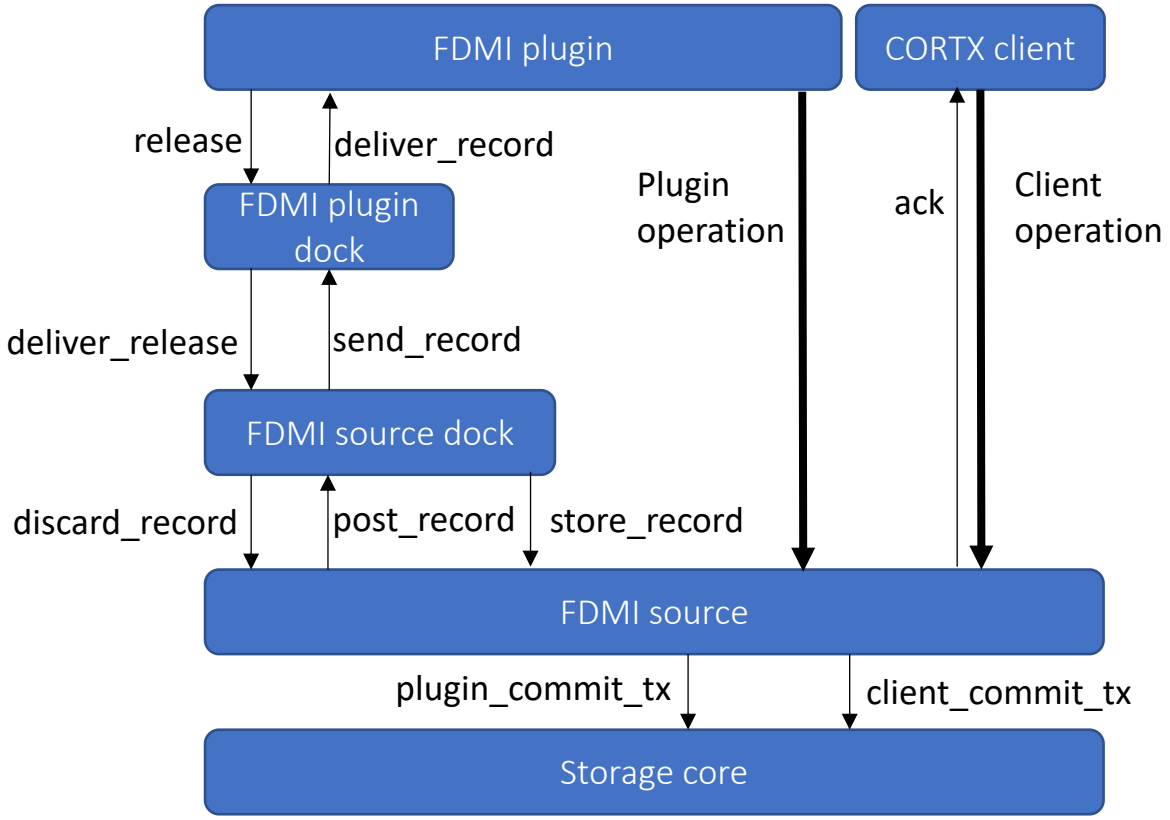
Class A plugin

- FDMI records are generated after client operations have been committed at the storage core.
- A special Release message is sent from plugin to source whenever records need to be discarded.



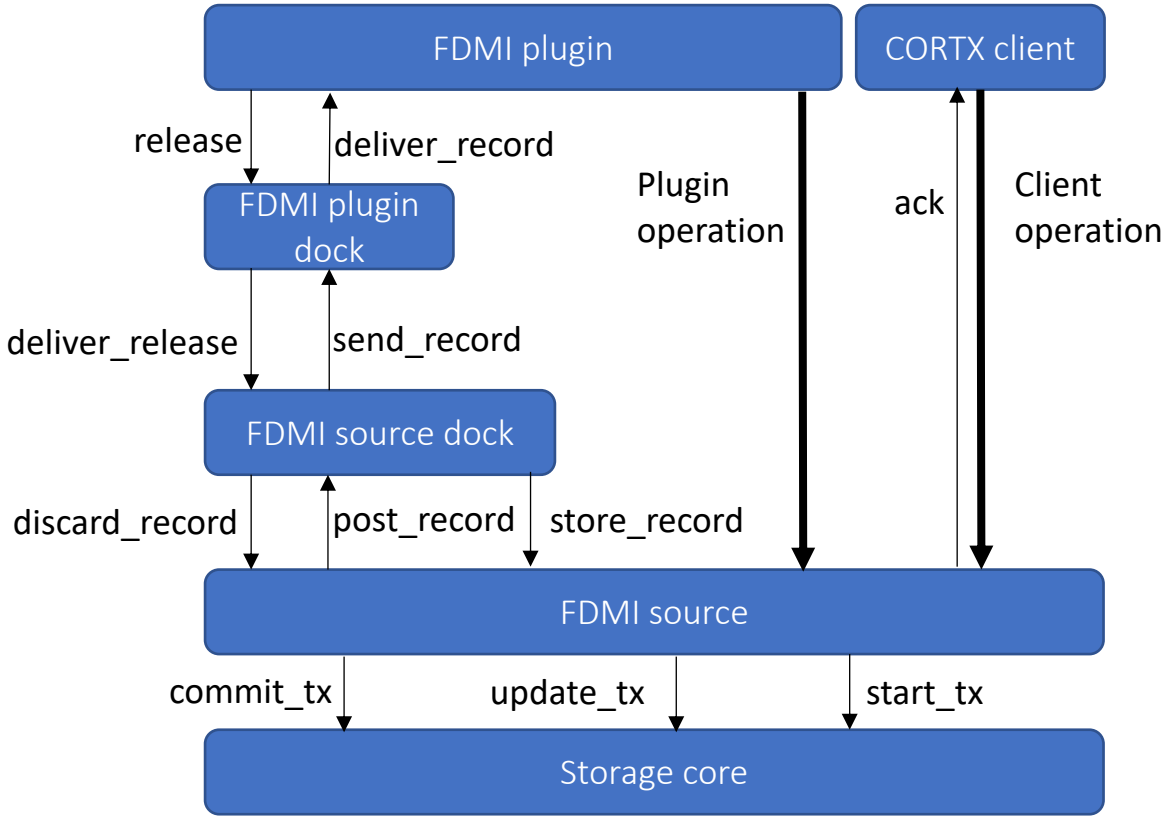
Class B plugin

- FDMI records are generated after client operations have been committed at the storage core.
- Plugin operations can be committed after initial client operations.
- A Release message is sent from plugin to source when the record can be discarded.



Class C plugin

- FDMI records are generated after client operations have been initiated at the storage core.
- Plugin-initiated operations are added to tje client-initiated transaction.
- The plugin's Release message discards the record at the source.



Categorizing Plugins

Plugin Class	Plugins	Integrated					Interposed					FDMI				
		S	T	N	E	D	S	T	N	E	D	S	T	N	E	D
A	I/O and System profiling												A			
B	Backup/Replication, Compression, Encryption, Semantic Enhancer, etc.												B,C			
C	Tiering, Caching, I/O offloading, etc.												C			

S	T	N	E	D
Storage access	Transactional	NO R/W amp.	Independent Scaling	Ease of development

Limitations and Discussion

- FDMI introduces overheads by adding multiple hops to the I/O path
- FDMI implementation can be optimized
  - Caching client I/O payloads at FDMI source
  - Offload computation to the FDMI source
- Generality evaluation using other storage systems (e.g., Ceph, Swift, and MinIO)

Conclusions

- FDMI is a pub-sub architecture that enables storage plugins with transactional guarantees.
- FDMI improves plugin development experience and makes plugins independently scalable.
- FDMI uses three plugin classes to address the needs of a wide variety of storage features.
- FDMI is a new design point for next-generation scalable and extensible storage systems.

Thank you!

SEAGATE

FIU | FLORIDA INTERNATIONAL UNIVERSITY

Examples of Plugins

Storage Plugins	Integrated					Interposed					FDMI				
	S	T	N	E	D	S	T	N	E	D	S	T	N	E	D
I/O profiling												A			
System profiling												A			
Backup/Replication												B,C			
Deduplication												B,C			
Encryption												B,C			
Compression												B,C			
Integrity Checker												B,C			
RAID Mirroring												B,C			
Semantic Enhancer												B,C			
Versioning												B,C			
Data Reorganization												B,C			
Tiering												C			
Caching												C			
I/O off-loading												C			
I/O shepherding												C			