FIX Antenna Java Failover Extension 1.0

Release Notes

Table of Contents

1 Overview

The purpose of this document is to provide an overview of POC results of FIX Engines Failover project. This document will be updated in course of the extension development with further implementation, supporting and maintenance related notes.

1.1 Problem introduction

In practice of FIX protocol-based system, all systems treat TCP/IP disconnects as implicit loss of FIX Session and expect clients to send a logon message upon re-connect. When logging in, there are two options — a re-connecting session may send the next outgoing sequence number or it may ask server to reset the sequence (to 1). In first case, the server side may send a logon acknowledgement if sequence is greater or equal to what it expected, or close (or even reject) the session if the received sequence number is less than expected. Additionally, if the sequence was greater than expected, server will issue a re-transmission. Client session monitors the sequence of the server as well, and needs to request a re-transmission if it detects a gap (received sequence is greater than expected). In the second case, if the server supports sequence reset, both in and out sequences are reset to 1 and no messages are recovered.

But also there are possible specifics of the venue FIX clients connect to. For instance, the client TCP connection should be switched to a failover FIX server. If a failover switch happens, usually a FIX server side fails to keep the sequence number intact, and clients are left no choice but reset the sequence number.

Also there is problem in FIX practice of FIX protocol-based system to deliver asynchronously created FIX messages to FIX client which is temporary in disconnected state. For instance, other components of a user application just handled an order execution but FIX client appeared in disconnected state at the moment, so user app can't send this just generated mandatory message. **Figure 1** below is FIX Message Sequence Diagram which describes FIX protocol connectivity and mentioned above decision making.

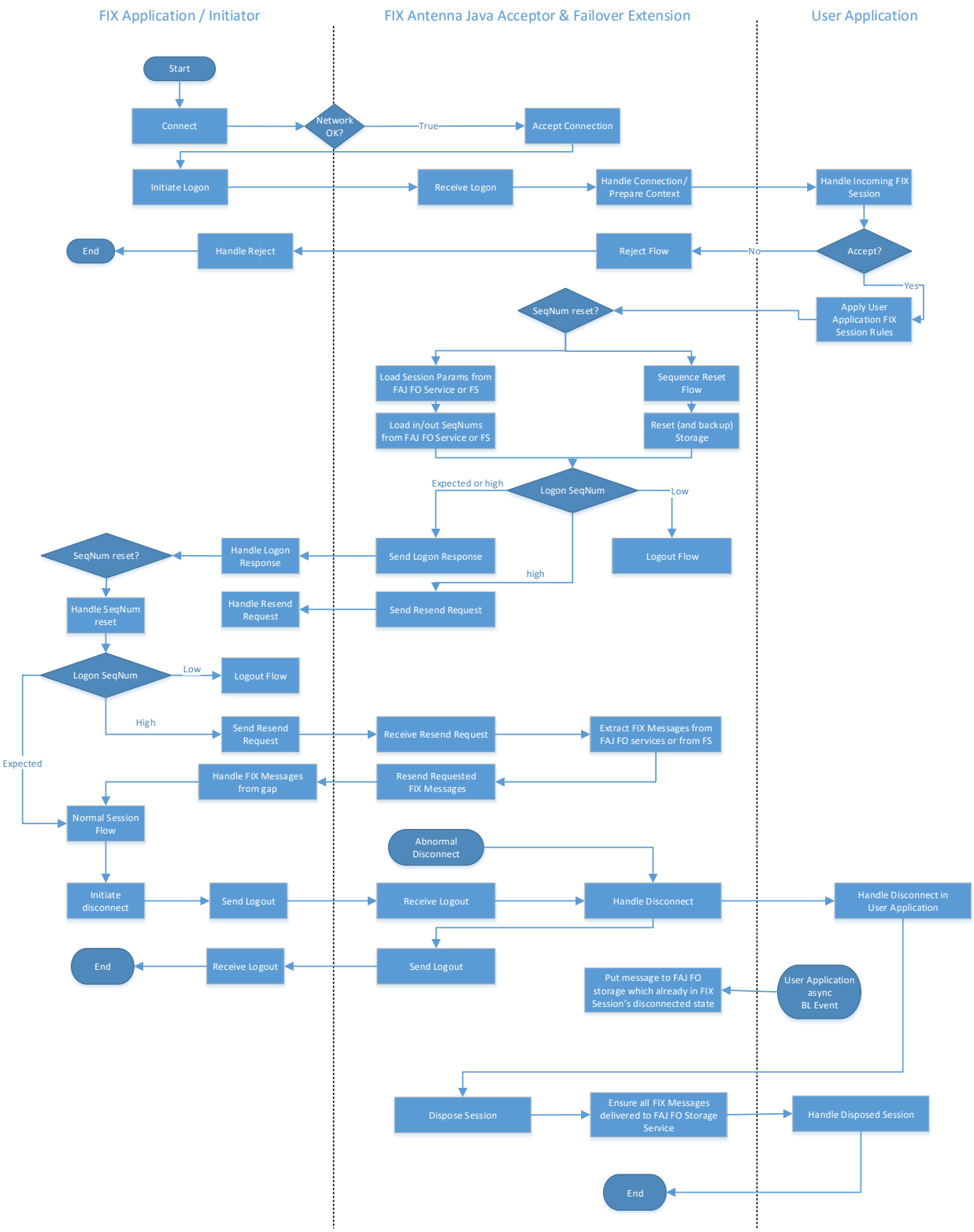


Figure 1 – FIX protocol sequence diagram

1.2 Solution Overview

B2BITS FIX Antenna Java (FAJ) is a FIX engine used by B2BITS customers on both initiator and acceptor sides. FAJ has an abstract layer to implement pluggable standard and specific storages. This way customers are able to enable required FIX Session persistence. For instance, FilesystemStorage stores and loads FIX Session properties, messages and sequences to/from file system. In case of enabled persistent (e.g. mentioned file system based persistence), it's possible to re-login and continue FIX Session at the same host. However, to address problem descried in the beginning, it was also decided to add distributed FIX Storage Service as a standalone component. Also it was decided to add specific storage implementation, which extends logic of usual local persistence with required distributed extension. This way it's possible to create a specific setup in a customer application environment, where FIX messages are stored not only locally, but also broadcasted to shared FIX Storages Services via UDP. This is enabling ability to restore FIX session state already on another host (via provided JAX-RS client API), including ability to resend required range of FIX messages. Also now it's even possible to restore messages directly from FIX Storage Service, without FIX protocol resend-related use cases.

OpenHFT Chronicle-Queue instances are to be used as FIX messages logs in FIX Storage Service. RealLogic Aeron are to be used to implement delivery of FIX messages to FIX Storage Service from FIX application instances, based on FAJ with enabled Failover Extension.

Figure 2 is a high level diagram pointing out main application blocks. The next sections will elaborate on all of the components mentioned below.

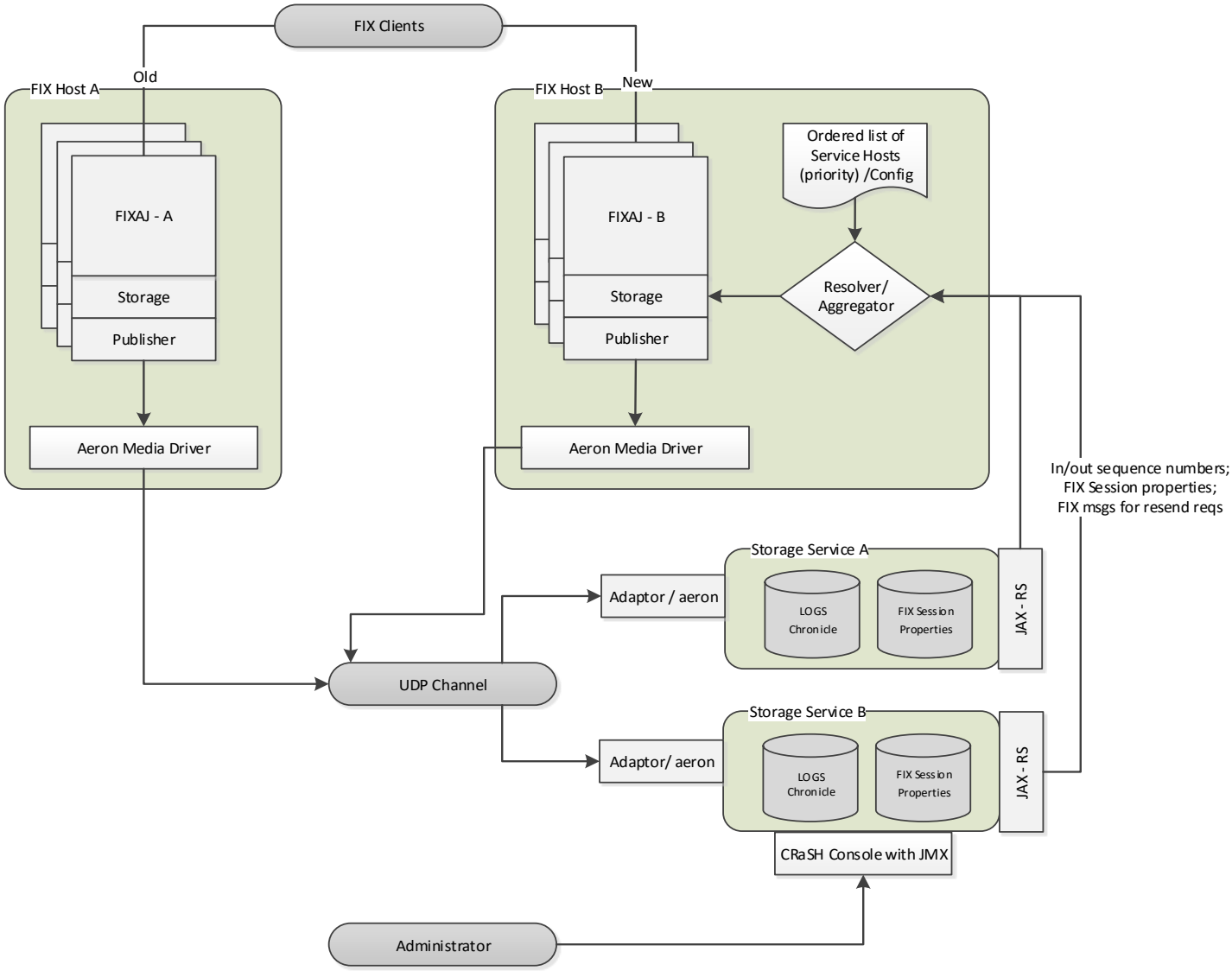


Figure 2 – High Level Solution Overview

2 Components and Extension structure

2.1 Extension overview

FIX Antenna Java Failover extension consists of number of sub modules.

2.1.1 Storage Service (fajfo-storage-service.jar)

Storage Service module contains implementation of distributed storage node based on Aeron and Chronicle. Storage Service also exposes jersey based REST web service allowing access to session's data store.

Packages overview (prefix: *com.epam.fixengine.failover.storage.service*):

- *storage* – storage related interfaces (storage, cursor, factory, manager, remover)
- *storage.remote* – REST service with configuration and interface
- *storage.chronicle* – Chronicle based implementation of the above storage interfaces
- *storage.nfsdb* – NFSdb based implementation of the above storage interfaces
- *storage.config* – implementation specific configurations

2.1.2 Storage Service Client (fajfo-storage-srv-client.jar)

Module defines number of client interfaces along with Aeron based implementation.

It doesn't have any business logic (filtering, data representation etc) on its own and basically needed as an abstraction layer between FIXAJ extensions and storage service layer (see 2.1.1).

From client module perspective any potential client just has to implement the following interfaces:

- *com.epam.fixengine.failover.storage.service.client.Publisher*
- *com.epam.fixengine.failover.storage.service.client.StorageService*

2.1.3 FIX Antenna Java Storage and Session implementations (fajfo-storage.jar)

Module defines some extra strategies/services deployed on top of FIXAJ:

- Cluster Message Storage – implementation of FIXAJ *MessageStorage* interface calling internally Storage Service Client through previously defined interfaces (see 2.1.2)
- FOFIXSession implementation and related factory which provided with ability to safely shutdown FIX Session in FO enabled setup, also ability to "send" FIX messages in already disconnected FIX Session (FIX message is to be added to FIX Storage Service and later can be loaded from it during FIX resend scenarios or directly)

2.1.4 Example "hello world" FIX-bases application (fajfo-example.zip)

"Hello World" type of example which allows to run an FIX Storage Service, run two instances of example FIX Servers (*com.epam.fixengine.failover.example.SwitchableFIXServer*, instances do not have shared file based persistence), and finally execute *com.epam.fixengine.failover.example.SimpleSwitchableClient*,

which will exchange with first FIX Server and then continue the same FIX Session against second FIX Server.

2.1.4.1 How to run example

Unpack fajfo-example-&version&.zip, go to bin directory:

- Run startStorage to start FIX Storage service
- Run startServer1 and then start Server2 to start two instances of SwitchableFIXServer instances, e.g. *startServer1 9992* and *startServer2 9993*
- Finally run an example, e.g. *startExample localhost 9992 localhost 9993*

2.2 Core frameworks/libraries overview

2.2.1 Chronicle

Chronicle is a Java project focused on building a persisted low latency messaging framework for high performance and critical applications (see https://github.com/OpenHFT/Chronicle-Queue)

The following features were considered as relevant from project perspective:

- Off heap storage means no extra pressure (almost) on Hotspot GC
- Selective replication is possible
- High performance and indexed storage support in single write, multiple readers mode.

2.2.2 Aeron

Efficient reliable unicast and multicast message transport (see https://github.com/real-logic/Aeron).

Aeron built around the concept of media driver so effectively any other protocols (apart from already supported) might be made available.

FXFO solution will be running a standalone version of Media driver per host. IPC wise Aeron client is connected to Media driver via memory mapped files (ring buffer data structure).

3 Solution Implementation Notes

This section elaborates on further implementation details of Aeron/Chronicle solution as it's seen in FAJ FO construction phase.

3.1 Overview

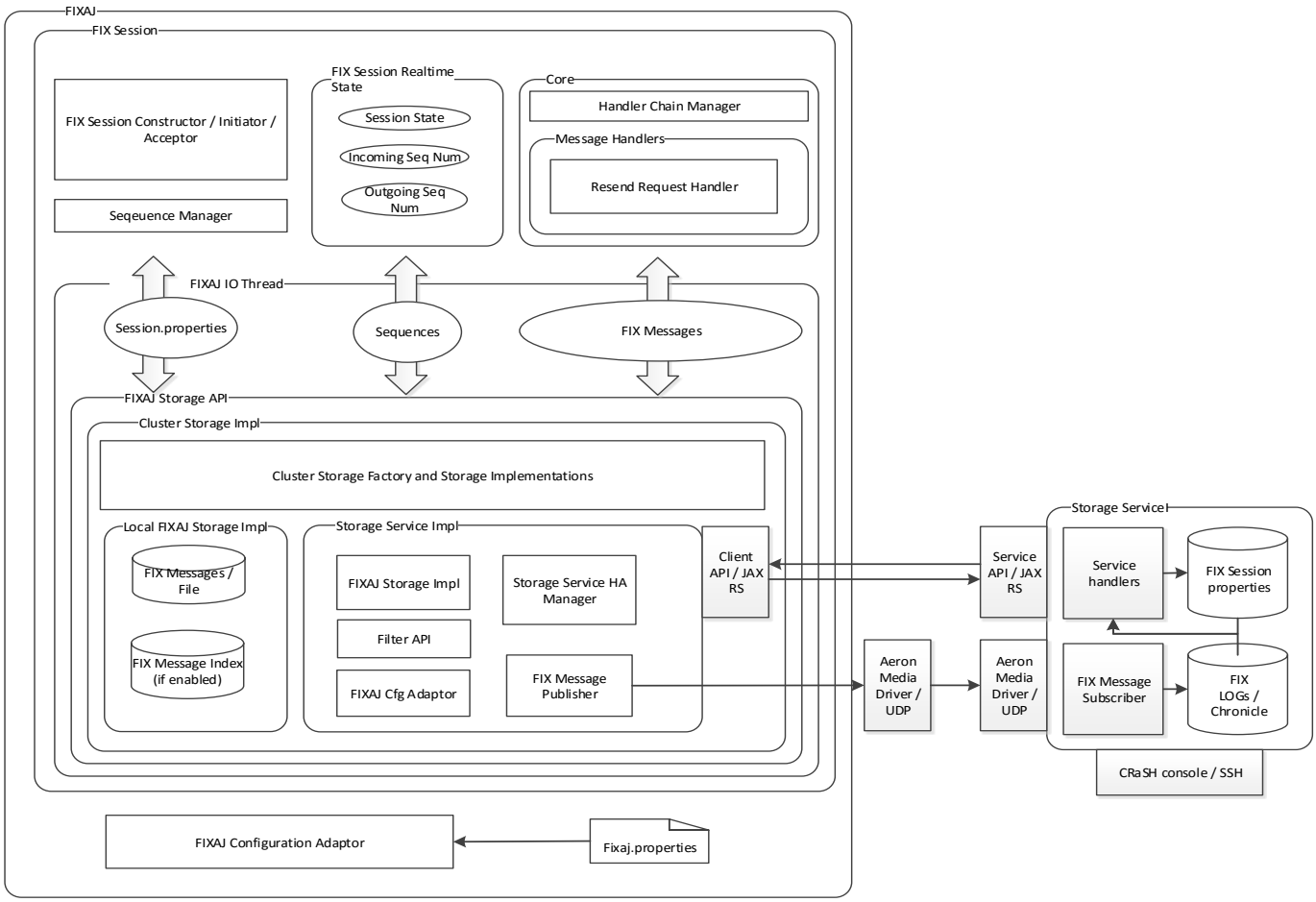


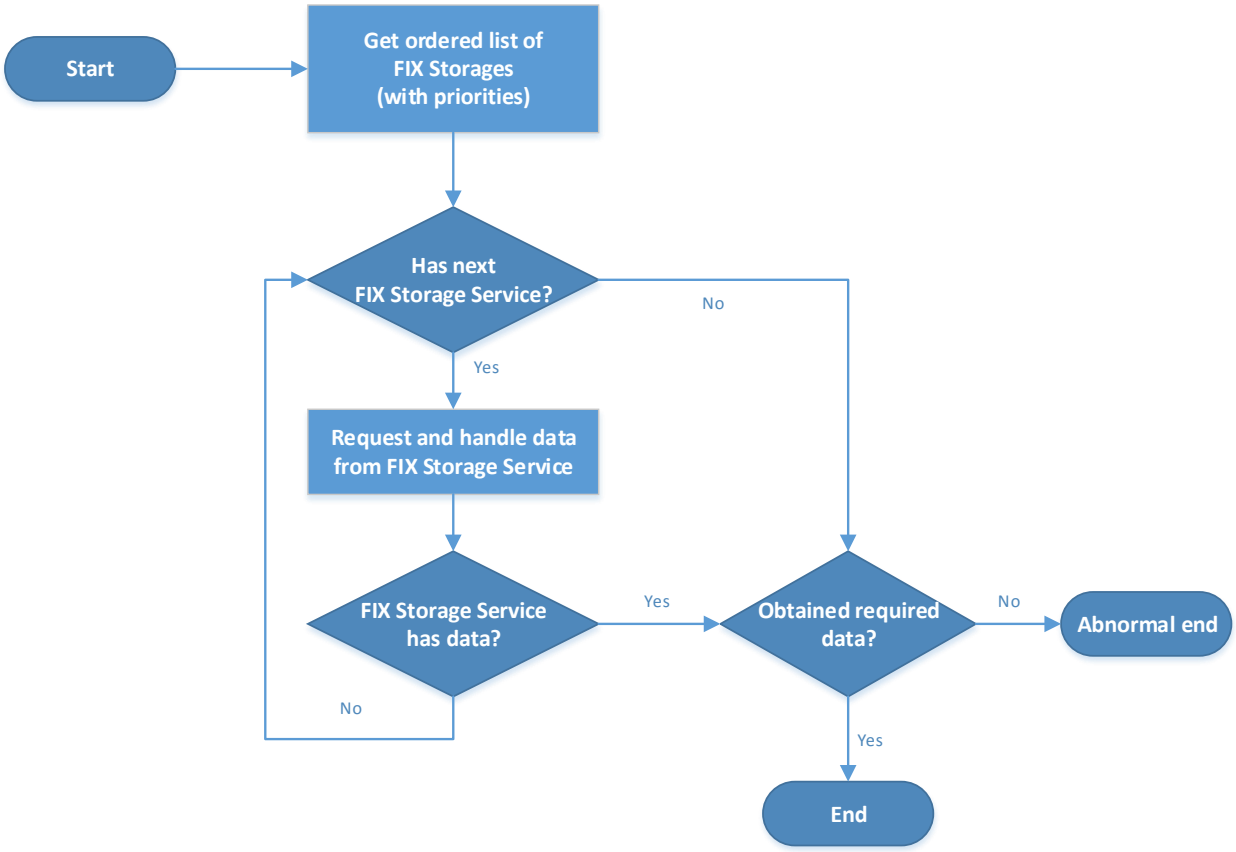
Figure 2 –Chronicle/Aeron solution overview

1. There are new cluster-based Storage Factory and Storage implementations on FIXAJ side.
2. Cluster-based Storage Factory and Storages create a configured instance of local persistence (available in FIXAJ) and instance of Storage Service Client(s)
3. Cluster-based storage, in turn, creates a publisher (Aeron based) to stream all of the incoming/outbound FIX messages in a Ring Buffer instance available through memory mapped file and Aeron media driver
    a. Filter API (as part of Client API) implements filtering of FIX messages

4. Aeron's Media Driver is responsible for sending FIX messages to FIX Storage Service(s) via configured FIX Message Channel. It's standalone instance, which handles all FIXAJ instances and FIX Session with enabled cluster storage per host
5. FIX Storage Service(s) are subscribed to configured FIX Message Channel to listen FIX Messages (via embedded Aeron's Media Driver)
6. FIX Storage Service(s) uses Storage API (and corresponding implementation) to log all of the incoming FIX message. Default implementation backed by Indexed Chronical Queue.
7. Chronicle log files organized per FIX Session at FIX Storage Service;
8. There are 3 possible type or records in log file:
    a. FIX message
    b. Filtered message indication
    c. Unknown value (used to indicate a gap in sequence numbers caused by data unavailability)
9. REST Web service exposes the following methods/functions:
    a. load/store FIX Session properties;
    b. get incoming/outbound sequence numbers for a FIX Session
    c. get a range of FIX Messages (for resend handling)
10. Requests for FIX Session state are sent to all configured FIX Storage Services. FIX Storage Service Client API in context of cluster-based persistence will make a decision on data is valid and perform data aggregations if necessary.

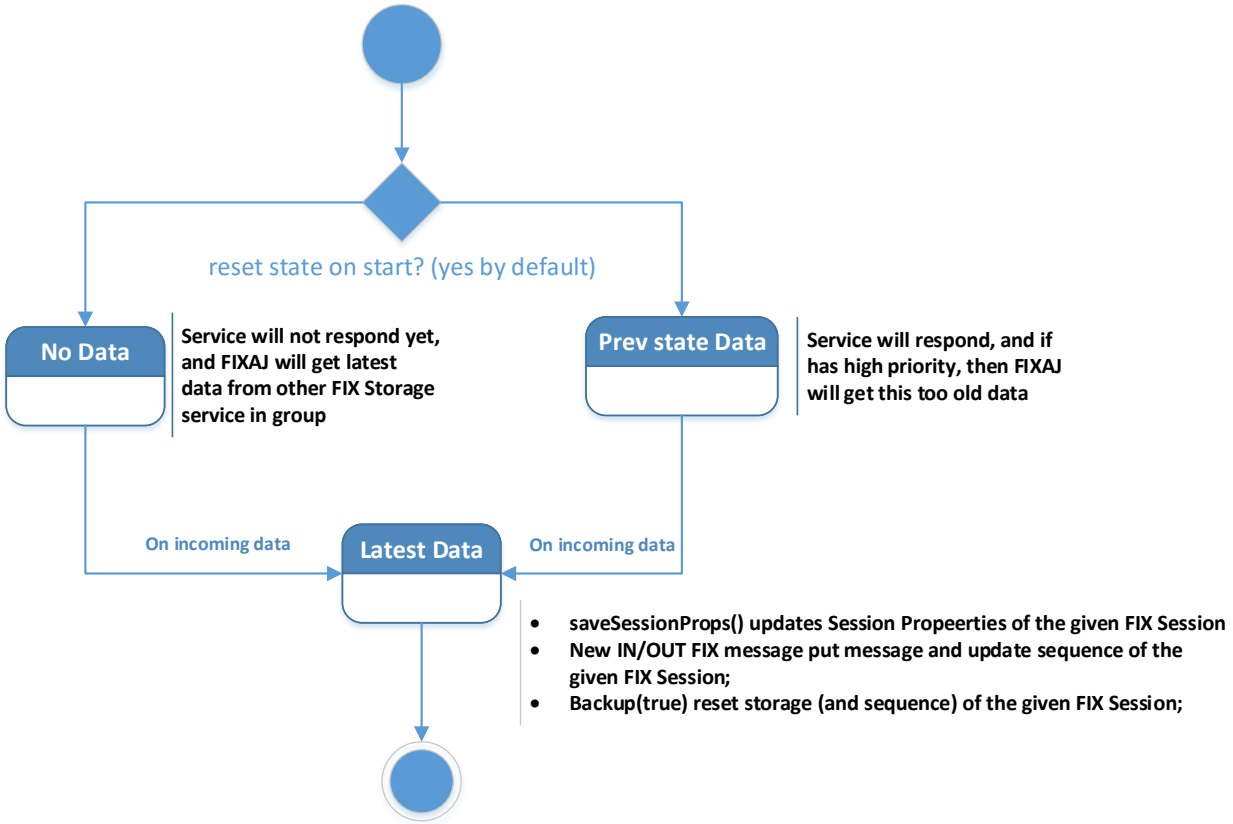
3.2 Logic overview in FIX Antenna Java Failover

3.2.1 REST API result aggregation in FIX Antenna Storage



- In order to get required data from FIX Storage, cluster-based storage in FIXAJ starts from the first FIX Storage Service in the configured list of FIX Storage Services (ordered by priority);
- If current instance of FIX Storage Service does not provide with expected data, then cluster-based storage tries to get data from next FIX Storage Service in the configured list of FIX Storage Services (if available);
- If all FIX Storage Services do not provide with required data, then FIXAJ storage tries to get data from local file-bases storage (if configured);
- If all sources do not provide with required data, then it is exceptional case with has to be handles properly according to FIX Session business logic;

3.2.2 FIX Storage Service states



- Just started FIX Storage Service does not have any data (be default it backup previous data in FIX Storages);
- If FIX Storage Service in "No data" state receives RS request for data from any FIXAJ instance, then it does not provide response with expected data; FIXAJ will get the expected data from another FIX Storage Service (according to priority);
- If FIX Storage Service received Session Properties of the given FIX Session at least once, then this FIX Storage Service is ready to provide any FIXAJ with Session Properties of the given FIX Session;
- If FIX Storage Service received FIX Message of the given FIX Session at least once (or special replacement of FIX Message), then this FIX Storage Service is ready to provide any FIXAJ Storage with sequence number of the given FIX Session;
- If high priority FIX Storage Service does not provide the required range of FIX Messages, then cluster-based storage will try to get the messages from other FIX Storage Service(s);

4 Non-Functional

Majority of design decisions made for Failover project are focused on making sure that overall FIXAJ performance in distributed mode stays as close to local mode as only possible. We do not expect more than 20% performance deviation from the baseline based on the most recent testing results.

4.1 Manageability

JMX can be implemented to expose current storage/session state. Should be design and implemented according to a customer application framework.

Storage(s) administration is also possible with CRaSH (http://www.crashub.org/) based console. FAJ FO commands are implemented to expose current storage/session state. Standard commands from the CRaSH release can be used to expose current state of JVM instances;

4.2 Reliability

It's recommended to have three storage nodes per data center running with one primary and two active backups. Deployment in UBS environment will give more information on the current setup and number of nodes might be adjusted.

4.3 FIX Antenna Java latency impact

Testing environment setup:

- Fix client (FIX AJ with Aeron based outgoing messages persistence) sending continues stream of prepared messages
- Fix server (FIX AJ with original binaries) accepts all the incoming messages
- 3 seconds warmup, 5 iterations, 5 seconds measurement time

Baseline metrics a taken in similar test environment but FIX client uses unmodified FIXAJ binaries with just local (file based) persistence.

Possible Aeron publisher impact to FIXAJ latency is (for avg message size 100 bytes):

p(50,0000) = 1,810 us/op

p(90,0000) = 1,812 us/op

p(95,0000) = 2,112 us/op

p(99,0000) = 4,224 us/op;

for avg message size 200 bytes):

p(50,0000) = 2,112 us/op

p(90,0000) = 2,112 us/op

p(95,0000) = 2,716 us/op

p(99,0000) = 7,544 us/op

5 Configuration Notes

5.1 FIX Storage

-Dstorage.config.file=[DIR] – set configuration file path for FIX Storage Service. Configuration file can have all properties (storage, communication) defined in CLI. Default configuration file path is *storage.conf*;

5.1.1 Storage:

- *-Dstorage.keepPrevState=[bool]* – move all previous created storage files to backup and start in clear state (true) or continue with previous state (false);
- *-Dstorage.path=[DIR]* – set base folder for storage; default value: *System.getProperty("java.io.tmpdir")*;
- *-Dstorage.backup.path=[DIR]* – set base folder path for backup data; default value: *System.getProperty("java.io.tmpdir") + File.separator + "backup";*

5.1.2 Communication:

- *-Dchannel.module=[name]* – set multicast communication provider - AERON or FAST_CAST; default value AERON;
- *-Dchannel.multicast.addr=[ip:port]* – set multicast channel address to subscribe to FIX Messages; default value: 225.0.0.1:5000;
- *-Dchannel.interface.addr=[ip]* – set multicast interface to defined address; default value: 127.0.0.1;
- *-Dhttp.port=[number]* – set port for Storage RESTful API; Default value: 9999;
- *-Dcrash.service.enabled=[bool]* – start CRaSH service. By default, it is disabled;

5.2 FIX Antenna Java

- *storageFactory* – set storage factory; must be com.epam.fixengine.failover.cluster.storage.MultiStorageFactory to enable failover features in FIXAJ;
- *foPublisherAeronEmbeddedMediaDriver* – set Aeron's Media Driver mode; default value: true;
- *foPublisherModule* – set multicast communication provider - AERON or FAST_CAST; default value AERON;
- *foStorageServiceUrl* – set connection URL to FIX Storage RESTful API; default value: http://localhost:9999/sessions;
- *foPublisherMulticastAddr* – set multicast address for FIX message publisher; default value: *aeron:udp?group=225.0.0.1:5000*;
- *foPublisherInterfaceAddr* – set interface address for FIX message publisher; default value: 127.0.0.1;
- *foStorageFactoryClassesList* – set storages factory used by MultiStorageFactory -- default value: class= com.epam.fixengine.failover.cluster.storage.FilesystemsStorageFactory,type=PRIMARY,serviceQueue= true;class=com.epam.fixengine.failover.cluster.storage.ClusterStorageFactory;

- *foMessageFilterClass* – set class name of FIX message filter implementation (implements com.epam.fixengine.failover.cluster.storage.*filter.FXFOMessageFilter*); default value: " com.epam.fixengine.failover.cluster.storage.filter.StandardFXFOMessageFilter";
- *foMessageFilterTagsList* – set tag list of FIX message types which should be filtered and not sent to FIX storage. Expected in case of default filter implementation: " com.epam.fixengine.failover.cluster.storage.filter.StandardFXFOMessageFilter";