FIXAJ Failover Solution

Overview

The purpose of this document is to provide an overview FIXAJ Failover solution.

Solution designed to provide a seamless failover within or outside of the data center (providing infrastructure support is available) for FIX sessions based on the already existing FIXAJ interfaces.

Solution is transparent from FIXAJ as a library perspective and only minimal changes to FIXAJ configuration are needed while the already existing FIXAJ related code is supposed to remain untouched.

Solution Overview

FIXAJ failover solution consists of the following main components:

1. fo-fixaj is an implementation of FIXAJ MessageStorage and MessageStorageFactory interfaces enabling nearly seamless integration with FIXAJ library.
2. fo-client is a module internal to fo-fixaj module supporting client side integration with fo-storage.
3. fo-storage is a standalone component acting as a server side from fo-client perspective facilitating capturing of session related information and data.

Figure 1 is a high-level diagram of main application blocks.

Figure 1 – High Level Solution Overview

Components and Project structure

Project overview

FO project consists of a number of sub modules. It's a maven-based project currently hosted on EPAM Git and built with Intellij TeamCity/Jenkins as CI.

Storage [/fo-storage]

Storage module contains implementation of a distributed storage node based on Aeron and Chronicle libraries.

Storage also exposes jersey based REST web service allowing remote access to a session data.

Packages overview (com.epam.fixengine.fo.messagestorage):

- storage – storage related interfaces (storage, cursors, factory, managers, replication)
- remote – REST service with configuration and interface
- chronicle – Chronicle based implementation of the above storage interfaces
- config – implementation specific configurations
- replication – session date replication services
- discovery – storage state streaming service (part of the storage discovery mechanism)

Storage implementation has no dependencies on FIXAJ library nor referencing any FIX protocol specific concepts and effectively can be used to store any sequenced data with some limitations and preconditions.

Storage Client (/fo-client)

Module implement a number of interfaces to support seamless integration of fo-fixaj with clustered storage solution.

It does not have any business logic (filtering, data representation etc) on its own and needed as an abstraction layer between FIXAJ extensions (see 2.1.3) and the transport layer.

Important classes:

- StorageClientProxy – storage client supporting failing over between available client
- StorageClientSet – clients set operating over the set of available clients
- FOClientContext – fo context initialised once storage state(s) received through discovery mechanism

FIXAJ Integration and Services [/fo-fixaj]

Module implements a set of strategies/services deployed on top of FIXAJ:

- ClusterMessageStorage – implementation of FIXAJ MessageStorage interface calling internally Storage Service Client through previously defined interfaces (see 2.1.2)
- ClusterStorageFactory – cluster oriented implantation of FIXAJ storage factory interface
- MultiStorageFactory – multi storage factory implementation of FIXAJ storage factory interface, with default settings run ClusterStorage as primary and FileSystemStorage as secondary
- MultiMessageStorage – storage operating on a combination of primary and secondary storage(s)
- ClusterInMemoryQueue – cluster backed implementation of FIXAJ in memory queue

Tests [/fo-tests]

Tests are using Cucumber BDD framework for integration and feature based testing and with help of the GridKit (ViCluster) library.

The following test scenarios are current supported:

- CLUSTER_EMU/bau.feature – basic scenarios
- CLUSTER_EMU/discovery.feature – discovery related scenarios
- CLUSTER_EMU/fallback.feature – scenarios covering fallback from primary to secondary etc.
- CLUSTER_EMU/queue.feature – scenarios testing cluster queue support
- CLUSTER_EMU/replication.feature - TBC
- CLUSTER_EMU/resubmit.feature – various resubmit scenarios with multiple storages
- REMOTE_STORAGE/bau-remote.feature

Please note that it is not available in distribution package and is not designed to be executed outside of EPAM network.

Core frameworks/libraries overview

Chronicle

Chronicle is a Java project focused on building a persisted low-latency messaging framework for high performance and mission 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
- High performance and indexed storage support in single write, multiple readers mode.

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.

Aeron transport functionally is used to delivery/exchange the following information:

- Session properties from client to storage
- Session messages from client to storage
- Storage state (discovery) from storage to clients and other storages
- Replication protocol (combination of multicast and unicast)

Solution Implementation Notes

This section elaborates on further implementation details of FIXAJ Failover solution.

Overview

Figure 2 –Chronicle/Aeron solution overview

1. Cluster based Storage Factory and Storage were implemented on FIXAJ side.
2. Cluster-based Storage factory creates a configured instance of local persistence (available in FIXAJ) and instance of storage client(s)
3. Cluster-based storage creates a publisher (aeron based) to stream all of the incoming/outbound FIX messages through a memory mapped file = aeron media driver
4. FIX Storage(s) are subscribed to configured message channels listening for incoming data (e.g. FIX Messages)
5. FIX Storage(s) uses Storage API (and corresponding implementation) to log all of the incoming messages. Indexed Chronical Queue backs the default implementation.
6. Chronicle log files organized per session and have a nontrivial logical structure to capture:
    a. Incoming messages
    b. Log sequence gaps
    c. Capture backfilled gaps through replication service
    d. Track filled gaps versus logged gaps
7. REST Web service exposes the following methods/functions:
    a. load session properties
    b. get incoming/outbound sequence numbers for a FIX Session
    c. get a range of session message (for resend handling)

Discovery

Storage clients (embedded into FIXAJ) are discovering remote storages in real time over the multicast channel.

Storage state distributed over multicast contains the following information:

- Storage id

- Storage address
- REST port

It means that extra storages can be added to the cluster group without changes in clients' configuration.

Discovery is essential mechanism for fo-clients and replication services run by fo-storages.

FO Storage Configuration

FO Storage configuration consists of the following properties configured either (or as a combination of) storage property file or VM options (-D).

Key	Default	Description
storage.id	N/A	Must be defined and unique within a configured cluster group. It is important for data replication and discovery services. E.g. A01, A02, A03 etc.
storage.path	./storage01.data	Main root folder for session data files. All data representing storage's current runtime state is stored here.
storage.backup.path	./storage01.data.backup	Path to back up session data folder
storage.backup	false	Enable to have session data files to be backed up on session reset.
storage.session.postfix	fo	Session data directory postfix, e.g. ini01-acc01.in.fo
storage.loadonstartup	true	Enable to have all storages at storage.path to be loaded on storage start up
storage.staterecovery	true	Enable to have session state to be recovered (sequence numbers etc.) on storage start up
rest.port	9991	Should be available / free, used by fo clients
multicast.address	224.0.3.1	UDP multicast address. Should be the same for all clients and storages (see fo.multicast.address in client configuration).
multicast.port		UDP multicast port. Should be the same for all clients and storages (see fo.multicast.port in client configuration).
multicast.interface		Network interface. Usually don't need to be changed unless have a dedicated multicast interface within/cross DCs.
data.streamId	1	Data stream ID
data.streams.num	1	Not used at the moment
state.streamId	50	Internal stream used for storage(s) state broadcasting. Usually don't need to be changed (see foStateStreamId in client configuration)
state.publish.interval	3000	In ms.
storage.localrolltime	23:00	Parameter drives Chronicle rolling.

FO Client Configuration

FO Client configuration consists of the following properties configured through either the FIXAJ properties file or VM options (-D).

Key	Default	Description
fo.multicast.address	227.0.3.1	UDP multicast address. Should be the same for all clients and storages.
fo.multicast.port	5001	UDP multicast port. Should be the same for all clients and storages.
fo.multicast.interface		Network interface. Usually don't need to be changed unless have a dedicated multicast interface within/cross DCs.

fo.multicast. stateStreamId	50	Internal stream used for storage(s) state broadcasting. Usually do not need to be changed.
fo.multicast. stateAvailabilityTimeoutMs	6000 (in ms)	If remote storage state are not available within the configured interval, MultiMessageStorageFactory will be falling back to a secondary factory.
fo. messageFilterClass	DefaultFOMessageFilter	Default message filter configured with foMessageFilterTagsList
foMessageFilterTagsList		
fo.fixaj. storageFactory .classes	class=com.epam.fixengine.fo.fixaj.cluster.storage.ClusterStorageFactory,type=PRIMARY,serviceQueue=true; class=com.epam.fixengine.storage.FilesystemStorageFactory, serviceQueue=true	Second layer of storage factories used by MultiStorageFactory (see above for details)
fo.fixaj. storageFactory. fallbackOnSecondarySessionParameters	true	Same as fo.fixaj.storage.fallbackOnSecondary but drives session parameters retrieval
fo.fixaj.storage. fallbackOnSecondary	true	If enabled allows fo-client to fall back on the secondary message storage while primary is not available (see MultiStorageFactory)
fo.fixaj. useInMemoryQueueOnRemoteRecoveryFailure	true	Keep in memory queue on session initialization if remote storages are not available
fo.remote. multiSourcing	true	If enabled allows fo-client reaching out to all the available storages (see discovery) to close gaps in the active client
fo.remote. incompleteResult	false	If enabled returns even incomplete message ranges with gaps. If disabled forcing fallback on secondary storage (depends on the settings) or throwing exception forcing FIXAJ to send reset.
fo.remote.batchSize	5000	REST client batch size per getMessage request
fo.publisher. notConnected. retryCount	11	Publisher retry count if aeron not connected (with scaling idle between retries)
fo.publisher. backPressure. retryCount	11	Publisher retry count if aeron returns back pressure status (with scaling idle between retries)
fo.publisher. dropOnBackPressure	false	Drop messages if networking/aeron layer is experiencing back pressure immediately w/o retry

Distribution package and setup

There are two packages in FIXAJ for distribution:

fo-fixaj-client-<version>-bin.zip

fo-storage-server-<version>-bin.zip

Each contains pom.xml with all the dependencies and jar file.

Note that these are not “uber” jars. It means that all the dependencies have to be resolved and included in application classpath according to the maven pom.

fo-storage start up

Main class: com.epam.fixengine.fo.messagestorage.FOStorage

Parameters:

-Dstorage.id=A01

-Dmulticast.address=224.0.3.1

-Dmulticast.port=5001

In order to run multiple storages make sure they have different storage.ids.

See FO Storage Configuration for extra parameters and description.

FIXAJ setup

FIXAJ configuration has to be adjusted with the following properties (either with –D or in engine.properties)

-DstorageFactory= com.epam.fixengine.fo.fixaj.cluster.storage.ClusterStorageFactory

-Dfo.multicast.address=224.0.3.1

-Dfo.multicast.port=5001

Note that multicast address/port for FIXAJ is coordinated with fo-storage parameters.

See FO Client Configuration for extra parameters and description.

For simple FIXAJ client / server please use samples provided within FIXAJ distribution package.

Limitations

1. Solution uses UDP multicast Aeron media driver meaning that UDB multicast must be allowed within datacenter and tunneled cross data centers where necessary.
2. Solution uses REST services for data retrieval on user configurable ports.
3. FO Storage doesn’t recover queue on storage restart regardless of storage.*backup* setting.
4. FO Storage doesn’t support data replication cross cluster of storages.
5. It’s not possible to configure both file based and cluster based persistence for queues.