GemFire Multi-site (WAN) Architecture

Introduction

The GemFire Multi-site (WAN) Architecture provides a way to connect to and asynchronously distribute events between disparate GemFire distributed systems. The main purpose of this architecture is to keep the data in two or more distributed systems consistent with each other. Each distributed system is referred to as a site, and each site is typically connected by a metropolitan area network (MAN) for disaster recovery or wide area network (WAN) for follow-the-sun processing. Events being distributed from one site to another are stored in queues on the local site and asynchronously batched and distributed to the remote site. The queues can be persistent, conflated, overflowed and/or highly available.

Terms

The following Multi-site (WAN) Architecture specific terms are used in this document.

A **Site** is one GemFire distributed system in the Multi-site (WAN) Architecture.

The **Gateway Event** (or **Event**) is a representation of a cache event that is sent from one **Site** to another.

The **Gateway Sender** (or **Sender**) represents the sending side of the Multi-site (WAN) Architecture. It converts local cache events into **Events** and queues them for distribution to remote **Sites**.

The **Gateway Receiver** (or **Receiver**) represents the receiving side of the Multi-site (WAN) Architecture. It receives batches of **Events** from remote **Sites** and applies them to the local **Site**.

The **Gateway Queue** (or **Queue**) represents the queue of **Events**.

A **Gateway Event Dispatcher** (or **Event Dispatcher**) peeks batches of **Events** from the **Queue**, filters, batches and distributes them to the **Receiver** on the remote **Site**.

An **Ack Reader** waits for and processes acknowledgements from remote **Sites**.

Discovery

Locators in each Site discover locators in remote Sites using the remote-locators property. When a locator starts, it attempts to exchange locator information with any remote locators defined by the remote locators property. The local locator sends a request to all remote locators containing its distributed system id, host and port among other information. Each remote locator responds with the hosts and ports of all known locators running in its Site. Each local locator maintains a mapping between a remote distributed system id and a set of all remote locators for that id.

Locator Startup Example

To start locators in three Sites that will discover each other, execute commands like below.

NY Site

```
gfsh start locator --name=locator_ny --locators=ny_host[10331] --port=10331 --J=\'-Dgemfire.remote-
locators=ln_host[10331],tk_host[10331]\'
```

LN Site

```
gfsh start locator --name=locator_ln --locators=ln_host[10331] --port=10331 --J=\'-Dgemfire.remote-
locators=ny_host[10331],tk_host[10331]\'
```

TK Site

```
gfsh start locator --name=locator_tk --locators=tk_host[10331] --port=10331 --J=\'-Dgemfire.remote-
locators=ny_host[10331],ln_host[10331]\'
```

Configuration notes:

1. The escaped single quotes around the `remote-locators` property are because of the comma.
2. If these commands are being tested on the same host, then in addition to the locator ports, the following ports will also need to be unique and configured on the command line:

- `jmx-manager-http-port` (configured like `--J=-Dgemfire.jmx-manager-http-port=8081`)
- `jmx-manager-port` (configured like `--J=-Dgemfire.jmx-manager-port=1091`)

The figure below shows the locators connected to each other across the WAN network.

blocked URL

Configuration

Senders and Receivers each have a number of available configuration options. Many of these are described below. For additional details on these and other configuration options, please see the documentation.

Sender

There are a number of Sender configuration options. The ones described in this document are listed below.

- remote distributed system id
- parallel
- high availability (HA)
- overflow / persistence
- dispatcher threads
- batch size and time interval
- conflation
- socket buffer size
- filters

Remote Distributed System Id

The remote distributed system id is the id of the distributed system to which this Sender is connecting.

Parallel

The Sender can either be serial or parallel.

Serial

Serial Senders send batches of Events between Sites from one primary member. A serial Sender comprises one primary and any number of secondary members. Each Event is maintained in all the members so that if the primary fails, one of the secondaries becomes primary and takes over where the previous primary left off.

When the serial Senders start, they all attempt to obtain the same distributed lock. The one that gets the lock becomes the primary. The others become secondaries. All the secondaries continue to try for the lock in case the primary fails.

Serial Senders should be used in these cases:

- when per-thread ordering of events is important. Events will be queued and distributed in the order they occurred per thread.
- when distributing events on replicated regions to remote Sites.

Parallel

Parallel Senders send batches of Events between Sites from many primary members. Parallel Senders spread the processing of Queues among all the nodes defining the partitioned region(s) that are attached to them.

Each parallel Sender runs in the members where the partitioned region data is stored and only processes events that are local to that member. Primary Senders process updates to primary data buckets; secondary Senders process updates to secondary data buckets. Both primary and secondary Senders deliver Events to the Queues. Only the primary Senders send batches of Events to Receivers and process acknowledgements.

There are a number of advantages to this topology including:

- more parallelism since each primary Sender is sending Events simultaneously.
- fewer local network hops since all replication is handled by the data region. See below for additional details.

Parallel Senders should be used when the data is partitioned and per-thread ordering of events is not important. Per-key ordering of events is always maintained.

High Availability (HA)

Queues can be highly available. Their entire contents are replicated in a separate member.

Serial

For serial Senders, Queue HA is achieved by configuring identical serial Senders in multiple members. The Queue is replicated between the members.

Parallel

For parallel Senders, Queue HA is achieved by configuring redundant-copies on the partitioned region(s) attached to the Sender. Since a parallel Sender is a special co-located partitioned region, it automatically uses the redundant copies setting of its attached partitioned region(s). The primary Senders are spread out among the members containing primary buckets for the data region.

Overflow / Persistence

Queues are always configured with LIFO overflow to disk that triggers when the Queue reaches a configurable memory threshold. They can also optionally be configured with disk persistence.

Dispatcher Threads

The dispatcher threads are the number of threads simultaneously distributing batches of Events to remote Sites. This option provides a way to parallelize batch distribution within a JVM. The default is 5 dispatcher threads. In the serial case, the key space is partitioned among the dispatchers so that one dispatcher is always responsible for updates to the same key(s). Similarly, in the parallel case, the buckets are partitioned among the dispatchers so that one dispatcher is always responsible for updates in the same bucket(s).

Batch Size and Time Interval

Batch size and time interval are the two ways to control when a batch of Events is distributed to the remote Site. Batch size refers to the number of events read from the Queue. Time interval refers to the amount of time reading from the Queue. The batch is sent whenever the batch size reaches the configured size or the time interval elapses whichever comes first. Setting the batch size or time interval low will decrease the latency since batches will be sent more frequently. Smaller batches may make it harder to fill the network bandwidth, though.

Conflation

Configure conflation when only the latest update to a particular key is necessary to distribute from one Site to another. Events are conflated both as they are added to the Queue and also after a batch has been peeked by the Event Dispatcher from the Queue. For queue conflation, the Event is added to the Queue, and if a previous value for the same key already exists on the Queue, then it is removed. If any duplicate key events get through queue conflation (which is possible if they are read from the Queue quickly), then the batch can still contain duplicates Events for the same key. Batch conflation removes these duplicates and only keeps the latest Event for each key.

Socket Buffer Size

The socket buffer size can be configured to set the buffer size of the Sender's socket. It is 32768 bytes by default. It generally should be set equal to or greater than the size of a batch of Events, so often the default is too small. It should be set identically to the one in the Receiver. One thing to note is that the socket buffer size can only be set to a maximum of the operating system's maximum socket buffer size. A warning will be logged if the configured size is greater than the operating system’s maximum.

Event Filters

One or more `GatewayEventFilters` can be added to any Sender. A `GatewayEventFilter` provides callbacks that can be used prevent queuing and /or transmission of Events. The callbacks can also be used log Events as they pass through the Queue.

A `GatewayEventFilter` provides these callbacks:

- The `public boolean beforeEnqueue(GatewayQueueEvent event)` callback can be used to prevent an Event from being enqueued.
- The `public boolean beforeTransmit(GatewayQueueEvent event)` callback can be used to prevent an Event from being transmitted to the remote Site.
- The `public void afterAcknowledgement(GatewayQueueEvent event)` callback can be used by the application to verify that an Event has been processed and acknowledged by the remote Site.

Receiver

There are a number of Receiver configuration options. The ones described in this document are listed below.

- start port and end port
- socket buffer size

Start Port and End Port

By default the Receiver's listen port is selected from an available port in the range 5000 (inclusive) to 5500 (exclusive). This port range can be changed by setting the start port and end port values. These options are generally used when there is a firewall which only allows specific ports need to be opened.

Socket Buffer Size

The socket buffer size can be configured to set the buffer size of the Receiver's socket. It is 32768 bytes by default. It generally should be equal to or greater than the size of a batch of Events, so often the default is too small. It should be set identically to the one in the Sender. One thing to note is that the socket buffer size can only be set to a maximum of the operating system's maximum socket buffer size. A warning will be logged if the configured size is greater than the operating system's maximum.

Configuration Examples

Both XML and gfsh configuration examples for the three Sites are described below.

XML Configuration

XML configuration for the three Sites would look like below.

NY Site

```
<gateway-sender id="ln" parallel="true" remote-distributed-system-id="2"/>
<gateway-sender id="tk" parallel="true" remote-distributed-system-id="3"/>
<gateway-receiver/>
<region name="data" refid="PARTITION">
  <region-attributes gateway-sender-ids="ln,tk"/>
</region>
```

LN Site

```
<gateway-sender id="ny" parallel="true" remote-distributed-system-id="1"/>
<gateway-sender id="tk" parallel="true" remote-distributed-system-id="3"/>
<gateway-receiver/>
<region name="data" refid="PARTITION">
  <region-attributes gateway-sender-ids="ny,tk"/>
</region>
```

TK site

```
<gateway-sender id="ny" parallel="true" remote-distributed-system-id="1"/>
<gateway-sender id="ln" parallel="true" remote-distributed-system-id="2"/>
<gateway-receiver/>
<region name="data" refid="PARTITION">
  <region-attributes gateway-sender-ids="ny,ln"/>
</region>
```

Gfsh Configuration

Gfsh scripts for the three Sites would look like below.

Note: Creating Senders and Receivers using gfsh results in their configuration being persisted by the cluster configuration service. See GemFire documentation for additional details regarding the cluster configuration service.

NY Site

```
connect --locator=ny_host[10331]
create gateway-sender --id=ln --remote-distributed-system-id=2 --parallel=true
create gateway-sender --id=tk --remote-distributed-system-id=3 --parallel=true
create gateway-receiver
alter region --name=data --gateway-sender-id=ln,tk
```

LN Site

```
connect --locator=ln_host[10331]
create gateway-sender --id=ny --remote-distributed-system-id=1 --parallel=true
create gateway-sender --id=tk --remote-distributed-system-id=3 --parallel=true
create gateway-receiver
alter region --name=data --gateway-sender-id=ny,tk
```

TK Site

```
connect --locator=tk_host[10331]
create gateway-sender --id=ny --remote-distributed-system-id=1 --parallel=true
create gateway-sender --id=ln --remote-distributed-system-id=2 --parallel=true
create gateway-receiver
alter region --name=data --gateway-sender-id=ny,ln
```

Supported Topologies

The GemFire Multi-site Architecture supports several different topologies. See the GemFire documentation for additional details regarding supported and unsupported configurations.

Star Pattern

The star pattern architecture is the most common. In this architecture, every Site knows about every other Site. An Event occurring in one Site is distributed by that Site to every other Site.

Serial Pattern

The serial pattern architecture is also used in a number of use cases. In this architecture, each Site knows about one other Site. An Event occurring in one Site is distributed by that Site to its configured remote Site which in turn distributes that event to its configured remote Site until the event has been distributed to all Sites.

Hub and Spoke Pattern

The hub and spoke architecture is also supported. In this architecture, each spoke Site knows about a central hub Site which in turn knows about all spoke Sites. An Event occurring in one spoke Site is distributed to the central hub Site which in turn distributes it to all other spoke Sites.

Site Consistency

By default, entry consistency is maintained between two Sites using each Event's timestamp. If updates to the same entry are done simultaneously, the resulting Events carry the timestamps to the remote Sites. Each Site applies the Event with later timestamp. This behavior can be overridden using a `GatewayConflictResolver`. The `public void onEvent(TimestampedEntryEvent event, GatewayConflictHelper helper)` callback is invoked when an Event is received from a different Site than the one that last modified the entry. The callback provides:

- a `TimestampedEntryEvent` containing the existing and proposed values as well as their timestamps and distributed system ids.
- a `GatewayConflictHelper` that can be used to disallow an event or change the event's value.

It is very important that this callback does the same thing in each Site; otherwise the Sites may not be consistent. See the GemFire documentation for additional details regarding Site consistency and `GatewayConflictResolvers`.

Under the Covers

Startup Messaging

When Senders and Receivers start, they exchange messages with their local locators.

Sender

When a Sender starts, it sends a request to its local locator containing its configured remote distributed system id. The local locator looks up the remote distributed system id in its mapping of remote locators and returns a response containing the set of all remote locators for that id. The Sender configured itself with a pool containing the remote locators.

Receiver

When a Receiver starts, it exchanges profile information with its local locators. The profile includes among other information, the Receiver's host and port. Each local locator maintains a list of all Receiver hosts and ports along with its current load (by default the number of connections to it). Whenever a request for a Receiver server is received, the locator returns the least-loaded Receiver from this list.

The figure below shows the Sender and Receiver startup messaging.

blocked URL

Event Lifecycle

The Event is generated on a server, but it starts with a client (or peer) cache operation.

- A client (or peer) does a cache operation (put/destroy)
- A server receives the cache operation and creates a cache event
- The server processes the cache event and delivers it to the local Sender(s) configured on the region being updated

- Each local Sender:
  - Generates an Event based on the cache event (including region name, key, value and callback argument, among other things)
  - Determines which remote Sites to should receive the Event
  - Updates the Event with the sending Site's distributed system id and each remote Site's distributed system id. New Events are sent to all configured remote Sites. For existing Events, the distributed system id of the sending Site and recipient Sites are used to determine which Sites have yet to receive it. Recipient Sites are updated accordingly as the Event is distributed between Sites.
  - Invokes any configured `GatewayEventFilter beforeEnqueue` callback
  - If the callback returns false, additional processing on the Event abandoned
  - If the callback returns true:
    - in the parallel case, the Event is put into the Queue for delivery to the remote Site by both the primary and secondary Sender
    - in the serial case, the Event is put into the Queue for delivery to the remote Site by the primary Sender and stored in the in-memory data structure by the secondary Sender

- The primary Event Dispatcher:
  - Peeks Events from the Queue a builds a batch of Events. The Queue is peeked until either the batch has reached the configured size or the time interval has elapsed (whichever comes first)
  - Invokes any configured `GatewayEventFilter beforeTransmit` callback for each Event in the batch
  - If the callback returns false, the Event is removed from the batch
  - Conflates any remaining events in the batch if conflation is enabled
  - Sends the batch to the remote Site for processing

- The Ack Reader:
  - Waits for acknowledgements from the remote Site
  - Invokes any configured `GatewayEventFilter afterAcknowledgement` callback
  - Removes the Events from the Queue
  - Waits for the next acknowledgement

- The Batch Removal Task:
  - Periodically sends messages from the primary Sender(s) to the Secondary Sender(s) to clean the secondary Queue(s)

- The Receiver receives the batch of Events
- For each Event in the batch, it:
  - Performs the appropriate operation (create, update, destroy) on the local Site (which causes the Sender cycle to start again)

The figure below shows Events flowing through a Queue.

blocked URL

Gateway Connectivity

The Sender is a client to the remote Receiver. Two fully-connected Sites maintain a two-way client/server architecture. The Sender's pool creates and maintains TCP connections to the remote Receiver like any other client pool. When the Sender needs a connection to a remote Receiver, the pool requests a Receiver host and port from one of its locators (which is a locator running in the remote Site). The remote Site locator sends a response containing the least-loaded Receiver host and port. See the documentation on Client/Server Configuration for additional details on pools.

The figure below shows Sender connectivity messaging.

blocked URL

Exception Handling

A few different kinds of exceptions can occur during the distribution of a batch of Events. These are:

- Connectivity exceptions between a Sender and a Receiver
- Processing exceptions while processing Events on a remote Site

Connectivity Exceptions

If a Sender cannot connect to a Receiver, Events remain in the Queue until the connection can be established. The Event Dispatcher continually attempts to establish the connection (by default, it tries every 1 second). As soon as the connection is established, batches of Events are sent normally. While the connection is down, the size of the Queue can continue to increase. If the size reaches the configured maximum queue memory, Event values are overflowed to disk. This helps reduce the chance of out-of-memory errors.

Batch Processing Exceptions

if an exception occurs while an Event is being processed by the Receiver, an exception is created for that Event, and batch processing continues. Once processing of the batch has completed, an exception reply is returned back to the Sender. Warnings are logged for any Event that caused an exception, and the Events are removed from the Queue. This allows the flow of Events to continue instead of getting into a potential infinite loop of exceptions.

Failover

Failover processing is different depending on whether the Sender is serial or parallel.

Serial

When a primary serial Sender fails, one of the secondaries obtains the distributed lock and becomes primary. When this occurs, each Event in the Queue is removed from the internal data structure. Once every Event in the Queue has been removed from the structure, the remaining Events in the structure are added to the Queue . These are Events that have only been received in the secondary. The data structure is then cleared. Finally, every Event in the Q ueue is marked as possible duplicate and processing continues. Every operation that needs to put Events into the Queue is impacted blocked during failover processing.

Parallel

Parallel Sender failover is handled entirely by the data region. When a data region's buckets become primary, the primary Sender automatically starts processing them.

Event Replication Within a Site

For Sender HA, Events are maintained in both primary and secondary Queues. Events are maintained differently between serial and parallel Senders.

Serial

A serial Queue is implemented as a special replicated region. All enqueuing to the Queue is done in the primary member. Events are replicated between the primary and secondary Queues using normal replicated region event replication.

The secondary serial Sender defines a data structure containing Events received by the secondary but not yet added to the Queue by the primary. Events are added to this data structure when the secondary Sender receives them and removed when the primary Sender adds them to the Queue and replicates them to the secondary Sender.

Once batches have completed processing and acknowledgements have been received in the primary, Events are locally removed from the Queue. A batch removal task periodically sends a message from the primary Sender to the secondary Sender so that the secondary's Queue is maintained properly.

The figure below shows serial Event replication within a Site.

blocked URL

Parallel

A parallel Queue is implemented as a special co-located partitioned region. Multiple partitioned regions can use the same Sender as long as they are co-located. Non-co-located partitioned regions must use different Senders even if they are connected to the same remote Site.

The parallel Sender uses its attached data region's event replication instead of providing its own. All replication is handled by the data region event replication. The parallel Sender introduces no additional network traffic.

With parallel Senders, the primary Sender is in the same member as the primary data region buckets, and the secondary Sender is in the same member as the secondary (or redundant) data region buckets. When cache events are received by the data region, they are delivered to local (in-member) parallel Senders for processing. Primary cache events are delivered to primary Senders; secondary cache events are delivered to secondary Senders. No Events are replicated between members. The primary Senders are the only ones distributing batches of Events to remote Receivers and processing acknowledgements.

As with serial Senders, once batches have completed processing and acknowledgements have been received in the primaries, Events are locally removed from the Queues. Batch removal tasks periodically send messages from the primary Senders to the secondary Senders so that the secondary's Queues are maintained properly.

The figure below shows parallel Event replication within a Site.

blocked URL