

WANdisco Distributed Agreement Engine Administration Guide

Table of Contents

<u>WANdisco Distributed Agreement Engine Administration Guide</u>	1
<u>1. Introduction</u>	2
<u>1.1. Definitions</u>	2
<u>2. Configuration File</u>	3
<u>2.1. Basic Syntax</u>	3
<u>3. Logging</u>	5
<u>3.1. Default File Handler</u>	5
<u>3.1.1. Log Directory</u>	5
<u>4. Required Elements</u>	6
<u>4.1. Provider Id</u>	6
<u>4.2. MAC Address</u>	6
<u>4.3. Member list</u>	6
<u>4.4. Agreement Manager</u>	7
<u>5. Preferences Reference</u>	9
<u>5.1. Transport</u>	9
<u>5.1.1. SOAP Transport</u>	9
<u>5.1.2. DConeNet Transport</u>	9
<u>5.1.3. In Process</u>	9
<u>5.1.4. TransportPolicy</u>	10
<u>5.2. Agreement</u>	10
<u>5.2.1. Timeouts</u>	10
<u>5.2.2. Activation</u>	11
<u>5.2.3. Optimizations</u>	11
<u>5.2.4. Backoff</u>	12
<u>5.3. Quorum</u>	12
<u>5.3.1. Quorum Change</u>	13
<u>5.4. Thread Pools</u>	13
<u>5.4.1. Agreement manager pool</u>	13
<u>5.4.2. Message Queue pool</u>	14
<u>5.4.3. Network IO Pool</u>	14
<u>5.5. Recovery and Persistence</u>	15
<u>5.5.1. Built-in Recovery Journal</u>	16
<u>5.5.2. JDBC based recovery</u>	18
<u>5.6. Distributed Garbage Collection</u>	19
<u>5.7. FileBasedProposal checksumming</u>	19
<u>5.8. Global Sequence</u>	19

WANdisco Distributed Agreement Engine Administration Guide

Version 1.5
Jan/2006

1. Introduction

DCone is at the heart of various WANdisco products like CVS Replicator. It provides the foundation for building a fault-tolerant distributed system.

This guide documents the various configuration parameters used by DCone. Considerable flexibility is provided in tuning DCone for various applications. Our products ship with reasonable defaults, but to get the most out of DCone it is imperative to look at this guide.

In addition to these configuration parameters, there are many programmatic APIs available to further tweak the behavior of DCone. These are used by application developers who are using our SDK.

1.1. Definitions

DCone

Distributed Coordination Engine.

GUID

Globally Unique Identifier. DCone uses 16 byte DCE UUIDs. We ship a utility called `guidgen` to create GUIDs easily.

Provider Id

This is the GUID for the current DCone node. It uniquely identifies a DCone node.

Agreement

Each step that the DCone replicated state machine executes is called an agreement.

Agreement Manager

Individual agreement steps of the replicated state machine are executed under the purview of one or more Agreement Managers.

Proposer

The node initiating a new *Agreement* step.

Acceptor

The recipient node for an incoming *Agreement* message initiated by a *Proposer*.

Learner

Any node that wants to learn the outcome of an *Agreement* step. Typically after a crash, a node becomes a *Learner* for all the agreements that occurred while it was down.

Quorum

Minimal number of nodes that are needed for reaching agreement on a proposal. These nodes need not be all up at the same time.

2. Configuration File

All configuration parameters are specified in a single file. The file uses XML syntax. Typically named `prefs.xml`. This can be easily over-ridden using the following system properties:

Preferences Root

-Dprefs.root=prefs-dir-name. Points to the directory which contains the prefs file. Default is current working directory.

Preferences file name

-Dprefs.file.name=myconfig.xml. Defaults to `prefs.xml`.

2.1. Basic Syntax

As noted above, we use an XML syntax to specify preferences(*prefs*). Root element is `Preferences`. There can only be one root element as required by XML. All other prefs are specified inside this root.

Look at the shipped `prefs.xml` file for sample syntax. Here is a snippet:

```
<?xml version="1.0" encoding="UTF-8"?>
<Preferences>

  <ProviderDescriptor>
    <DefaultProviderId>3bfbf219-2918-11d7-80c5-00065be02953</DefaultProviderId>
  </ProviderDescriptor>

  <MemberList>
    <Member name="3bfbf219-2918-11d7-80c5-00065be02953">
      <Profiles>
        <TransportPolicy>AlwaysDConeNet</TransportPolicy>
        <Transport>
          <DConeNet>
            <ListenerIP>localhost</ListenerIP>
            <ListenerPort>6020</ListenerPort>
          </DConeNet>
          <DConeFTP>
            <ListenerIP>localhost</ListenerIP>
            <ListenerPort>6021</ListenerPort>
          </DConeFTP>
        </Transport>
      </Profiles>
    </Member>
  </MemberList>

  <AgreementManagerList>
    <AgreementManager name="6e49ad70-10d9-11d9-af41-00065be02953">
      <DisplayName>cvs-am</DisplayName>
    </AgreementManager>
  </AgreementManagerList>

  <DConeNet>
    <UseNonBlockingIO>true</UseNonBlockingIO>
    <ReadStage>
      <!-- In a thread per conn model, this is really the max conns we can do -->
      <MaxThreads>5</MaxThreads>
    </ReadStage>
```

WANdisco Distributed Agreement Engine Administration Guide

```
<WriteStage>
  <!-- -1 means no timeout /-->
  <ThreadKeepAliveTimeOut>-1</ThreadKeepAliveTimeOut>
  <MaxConnectionsPerThread>1</MaxConnectionsPerThread>
  <MaxThreads>5</MaxThreads>
  <UseBlockingConnect>true</UseBlockingConnect>
</WriteStage>
</DConeNet>

<ThreadPool>
  <AgreementThreads>2</AgreementThreads>
</ThreadPool>
<MessageQueue>
  <MaxThreads>2</MaxThreads>
  <BufferSize>5000</BufferSize>
</MessageQueue>

<org>
  <nirala>
    <util>
      <guid>
        <MACAddress>00:06:5B:E0:29:53</MACAddress>
      </guid>
    </util>
  </nirala>
</org>

</Preferences>
```

3. Logging

DCone's logging facility is built on top of JDK1.4 style logging. The usual `log.properties` can be used to control the logging parameters. The system properties that govern how the `log.properties` file is located are:

Java property –

`-Djava.util.logging.config.file=file-name` If this is specified, it over-rides all other properties.

Preferences Root –

`-Dprefs.root=prefs-dir-name` If Java property is not specified, DCone will try to locate it in the `prefs` directory.

DCone enhances the default JDK logging facility to dump thread-ids, microsecond timestamps, and automatically attach a default file handler.

3.1. Default File Handler

The default file handler dumps the log records to files prefixed by `Main_Class_Name-{prefs_file_prefix}.log`.

Log rotation can be setup by specifying the standard Java properties in the `log.properties` file. These properties are:

Maximum log file size –

`java.util.logging.FileHandler.limit = {number of bytes}` If this is specified, it over-rides the default which is unlimited log file size.

Number of log files –

`java.util.logging.FileHandler.count = {count of files}` It specifies how many log files to cycle through, defaults to 1. Each log file will have a count appended to it.

3.1.1. Log Directory

Our products, such as CVS Replicator, dump their log records to `installDir/logs`.

For the users of our SDK, we provide a programmatic API to control the log file name and the log directory:

```
org.nirala.trace.Logger.initDefaultFileHandler(className, logDir);
```

4. Required Elements

Basic bootstrapping requires the presence of a few mandatory elements in the prefs file. These are documented below:

4.1. Provider Id

There are two ways to specify the provider id:

1. Hardwire a GUID directly (all our products like CVS Replicator use this form):

```
<ProviderDescriptor>
  <DefaultProviderId>3bfbf219-2918-11d7-80c5-00065be02953</DefaultProviderId>
</ProviderDescriptor>
```

2. Use Programmatic API:

```
<ProviderDescriptor>
  <DefaultProviderIdHookClass>A.Class.For.ProviderIdHook</DefaultProviderIdHookClass>
</ProviderDescriptor>
```

This class needs to implement the interface `org.nirala.util.DefaultProviderIdHook` that specifies:

```
public interface DefaultProviderIdHook {
    public String getIdString();
    public IGUID  getID();
}
```

4.2. MAC Address

The built-in GUID generator needs to bootstrap itself using an Ethernet MAC address. This enhances the uniqueness of the GUIDs and reduces the odds of the same GUID being generated by two different machines. Most of our sample prefs files ship with a default MAC Address. Please change it as soon as possible. The syntax is:

```
<org>
  <nirala>
    <util>
      <guid>
        <MACAddress>00:06:5B:E0:29:53</MACAddress>
      </guid>
    </util>
  </nirala>
</org>
```

You can easily determine your MAC address by running `ifconfig -a` on UNIX or `ipconfig /all` on Win32.

4.3. Member list

Since DCone is used in a distributed setting, to do anything useful, one needs to specify an initial *membership*. This is the initial set of members that are part of the distributed system. For example in case of CVS

Replicator this is the list of all the replicas. Using our group evolution feature it is possible to dynamically adjust this initial membership.

Here is an XML snippet:

```
<MemberList>
  <Member name="3bfbf219-2918-11d7-80c5-00065be02953">
    <Profiles>
      <TransportPolicy>AlwaysDConeNet`w</TransportPolicy>
      <Transport>
        <DConeNet>
          <ListenerIP>sanfrancisco-replicator</ListenerIP>
          <ListenerPort>6020</ListenerPort>
        </DConeNet>
      </Transport>
    </Profiles>
  </Member>

  <Member name="c9bcecl9-4301-11d9-b12d-080020b0d8b8">
    <Profiles>
      <TransportPolicy>AlwaysDConeNet</TransportPolicy>
      <Transport>
        <DConeNet>
          <ListenerIP>bangalore-replicator</ListenerIP>
          <ListenerPort>6020</ListenerPort>
        </DConeNet>
      </Transport>
    </Profiles>
  </Member>

</MemberList>
```

For each member (including the self node) we have specified:

- Member name using a GUID.
- For each member, the associated Profiles.
 - ◆ The `TransportPolciy` profile specifies the default transport this member prefers when a node needs to communicate with it.
 - ◆ Transport Endpoints for communication. In the above example, we have specified *DConeNet* binary protocol for communication as well as the listener IP address and port. Multiple transports can be specified, as we will see in the [transport prefs](#).

Please specify the `Member` element for each initial member of the distributed system.

4.4. Agreement Manager

A Distributed Agreement Manager accepts proposals from the application, and inserts them into a global sequence. This is the fundamental concept behind our SDK. For further information please read our [Introduction to DCone](#) guide.

Some applications, such as CVS Replicator, only use a single agreement manager. In general, applications can use as many agreement managers as they require. DCone is designed to scale to multiple agreement managers. For example a Distributed File System application may choose to use one agreement manager per directory.

WANdisco Distributed Agreement Engine Administration Guide

All the members that are part of a distributed system, must specify common agreement managers in their individual prefs file. It is also possible to specify agreement managers using our programmatic API.

Each agreement manager essentially represents a *replicated state machine*. Submitted proposals are turned into globally sequenced agreements by the agreement manager.

Syntax:

```
<AgreementManagerList>
  <AgreementManager name="6e49ad70-10d9-11d9-af41-00065be02953">
    <DisplayName>DFS-Dir1</DisplayName>
  </AgreementManager>
  . . . .
  <AgreementManager name="3e40ad70-10d9-a1d9-ef45-00065be02953">
    <DisplayName>DFS-Dir2</DisplayName>
  </AgreementManager>

</AgreementManagerList>
```

The `AgreementManagerList` is the root element of this tree. It contains multiple `AgreementManager` instances, each identified using a GUID. An optional `DisplayName` can be associated for documentation.

Some products, like CVS Replicator, already have an agreement manager specified in the shipped prefs.xml file. There is typically no need to change that, unless you start deploying multiple instances of the CVS Replicator product.

5. Preferences Reference

5.1. Transport

DCone supports multiple pluggable transports for its messaging. It comes bundled with :

- SOAP/XML Transport
- DConeNet Binary Transport
- In-process Transport

Transports are specified within the *MemberList/Member* element for a given member.

```
<MemberList>
  <Member name="3bfbf219-2918-11d7-80c5-00065be02953">
    <Profiles>
      <TransportPolicy>..
      <Transport>..
    ...
```

5.1.1. SOAP Transport

Typically used by Web Services. DCone uses Apache AXIS by default as the SOAP transport provider. To use it, just specify the SOAP/Http endpoint *URL*. E.g.

```
...
<Transport>
  <SOAPTransport>http://localhost:7005/dcone/services/dconeEndPoint</SOAPTransport>
</Transport>
...
```

5.1.2. DConeNet Transport

This is the preferred transport. It uses an optimized binary encoding, and is significantly less verbose than SOAP. To use it, specify the host-name/IP address and a TCP Port. The *TimeToConnect* element specifies the maximum timeout in milli-seconds that a thread waits when connecting to the end-point. Default is 500ms. For very high-latency connections (example from USA to China) it can be increased to avoid spurious timeouts while waiting to connect.

```
.....
  <Transport>
    <DConeNet>
      <ListenerIP>sanfrancisco-replicator</ListenerIP>
      <ListenerPort>6020</ListenerPort>
      <TimeToConnect>750</TimeToConnect>
    </DConeNet>
  </Transport>
```

5.1.3. In Process

DCone supports in-process activation of member nodes. Messages are delivered directly without even going to the loopback address. Inprocess is always preferred over other transports unless the *TransportPolicy*

provides an over-ride.

```
<Transport>
  <InProgress>true</InProgress>
</Transport>
```

5.1.4. TransportPolicy

If multiple transports are available for a given member, the DCone node can be told to prefer one over the other using `TransportPolicy`. The valid values are `AlwaysDConeNet` or `AlwaysInProgress` or `AlwaysSOAP`. If the `TransportPolicy` element is missing, DCone defaults to `AlwaysDConeNet`, if DConeNet profile is available.

5.2. Agreement

Each step that the DCone replicated state machine executes is called an *Agreement*. The following sections documents the prefs that impact the Agreement module.

The various tunables are specified under the XML element `Synod`.

```
<Synod>
  ...
</Synod>
```

Some of the less commonly used one are specified under `org/nirala/synod` tree:

```
<org>
  <nirala>
    <synod>
    ....
```

5.2.1. Timeouts

The timeouts are specified as following:

```
<Synod>
  <AcceptorTimeout>20000</AcceptorTimeout>
  <AggregatorTimeout>20000</AggregatorTimeout>
  <learnerTimeout>20000</learnerTimeout>
</Synod>
```

AcceptorTimeout

Amount of time in milli-seconds, after which an acceptor will timeout and initiate a *learning phase* to learn the outcome of the agreement for which it is an acceptor. Defaults to 600 milli-secondseconds.

AggregatorTimeout

Amount of time in milli-seconds, a proposer waits to receive replies for it's agreement messages.

Timeout triggers counter-measures to complete the agreement. Defaults to 200s.

learnerTimeout

Amount of time in milli-seconds, after which an acceptor tries to initiate a learning protocol to see if someone else on the network has learned the outcome for a given agreement. Defaults to 60s.

5.2.2. Activation

To keep the memory footprint of a DCone node low, agreements that are done (globally agreed upon) are deactivated from memory and swapped out onto a persistent datastore on disk. Various activation policies control when the deactivation occurs. Default is deactivate right away/immediately.

Deactivation policy is specified via :

```
<Synod>
  <ActivationManager>activation-policy</ActivationManager>
  ...
```

The *activation-policy* can be set to:

- *org.nirala.synod.activation.Simple* (default) : immediate deactivation
- *org.nirala.synod.activation.Disable* : never deactivate. Useful for debugging.
- *org.nirala.synod.activation.Delayed* : Deactivate after a delay (default 10s).

If using delayed deactivation, a delay in milli-seconds can be specified:

```
<Synod>
  <ActivationManager>org.nirala.synod.activation.Delayed</ActivationManager>
  <DeactivationDelay>2000</DeactivationDelay>
```

It makes sense to use delayed deactivation if remote nodes go down frequently and want to re-learn finished agreements from this node. Delayed deactivation avoids frequent swapping from disk to memory for such scenarios.

5.2.3. Optimizations

These are patent-pending optimizations that can make order of magnitude difference in performance under appropriate conditions.

```
<Synod>
  <Optimizations>
    <UseFastPropose>true</UseFastPropose>
    <UseWeakReservation>true</UseWeakReservation>
    <UseDistinguishedRoundNumbers>true</UseDistinguishedRoundNumbers>
    <PresumeConsensus>true</PresumeConsensus>
  </Optimizations>
</Synod>
```

UseFastPropose

Defaults to `true`. Optimizes the normal case by cutting down on an extra round of messages.

UseWeakReservation

Defaults to `false`. If you have an extremely high volume of transactions and load can be evenly distributed across DCone nodes, turning on this optimization can improve performance tremendously.

UseDistinguishedRoundNumbers

Defaults to `true`. If you have lot of contention from multiple DCone nodes, this option can prevent false aborts.

PresumeConsensus

WANdisco Distributed Agreement Engine Administration Guide

Defaults to `false`. Use it in conjunction with `UnanimousQuorum` and `UseFastPropose` set to off. Optimizes the failure case, by sensing agreement early on.

5.2.4. Backoff

The DConc uses an exponential backoff algorithm similar to *CSMA/CD for non-switched ethernets*, to ensure rapid convergence when multiple competing nodes are all trying to act as proposer for the same agreement. The following options can be used to tweak the behavior of the backoff algorithms:

```
<org>
  <nirala>
    <synod>
      <heuristics>
        <friendliness>1.0f</friendliness>
        <!-- A single multiplier for all the backoff tunables
          fixedComponentOfInitialValue *= friendliness;
          groupSizeMultiplier *= friendliness;
          upperBoundMultiple *= friendliness;
          gracePeriodMultiplier *= friendliness;
          estimatedRoundTripDelayMultiplier *= friendliness;
        /-->
      </heuristics>
    </synod>
```

You can either specify a `friendliness` multiplier or modify individual tunables.

5.3. Quorum

Current crop of high availability solutions typically require blocking for response from all the member nodes. Unlike them, DConc only needs a quorum of nodes to vote on an agreement before declaring it as done or committed.

In addition to supporting declarative quorum policies in prefs file, developers can easily plug-in custom quorum implementations.

```
<Quorum>
  <!--
    Default is
      org.nirala.group.quorum.UnanimousResponseQuorum -->
  <DefaultClass>org.nirala.group.quorum.MajorityResponseQuorum</DefaultClass>
  <DistinguishedNode>3bfbf219-2918-11d7-80c5-00065be02953</DistinguishedNode>
</Quorum>
```

Preferences file based policies are:

1. *UnanimousResponseQuorum* (default) – As the name suggests, will wait for responses from all nodes.
2. *MajorityResponseQuorum* – Only requires 51% of nodes to respond. For example in a 5 node scenario, will require only 2 remote nodes to send a response (presuming self node always sends a response). In case of even number of nodes (for example, 4) use `DistinguishedNode` as the tie-breaker.
3. *DistinguishedNode* – Optional. Useful when there are an even number of nodes or singleton quorum is used. 50% of the nodes constitute a majority quorum if and only if they include the distinguished node.

4. *SingletonResponseQuorum* – This quorum trades off performance for availability. The *distinguished node* becomes effectively the required quorum as well as a single point of failure. If a transaction is committed at the distinguished node it can zip through very fast as responses from remote nodes are not needed. However the remote nodes always need a response from the *distinguished node*.

5.3.1. Quorum Change

DCone supports the notion of dynamically changing the *distinguished node* associated with *SingletonResponseQuorum* or *MajorityResponseQuorum with even number of node*. This can be done via the web administration interface or a schedule can be specified in the `prefs.xml`. The schedule is applied on a daily basis or 24 hours. For instance,

```
<AgreementManagerList>
  <AgreementManager name="52ec6735-ce20-11d9-8e57-00065be02953">
    <DisplayName>cvs-am</DisplayName>
    <Quorum>
      <Schedule>
        <at name="12:00:23 AM">
          <DistinguishedNode>fb7723de-ce1e-11d9-ae57-00065be02953</DistinguishedNode>
        </at>
        <at name="12:20 AM">
          <DistinguishedNode>659a768d-ce1f-11d9-aeec-00065be02953</DistinguishedNode>
        </at>
        <at name="12:40 AM">
          <DistinguishedNode>3fae40f3-ce20-11d9-8e6a-00065be02953</DistinguishedNode>
        </at>
      </Schedule>
    </Quorum>
  </AgreementManager>
</AgreementManagerList>
```

This schedule will result in *distinguished node* being changed to `fb7723de-ce1e-11d9-ae57-00065be02953` every morning at `12:00:23 AM` local time. The time syntax is – *hh:mm:ss AM/PM* – seconds are optional.

5.4. Thread Pools

DCone uses several thread pools that can be independently tuned. These are:

1. Agreement manager pool
2. Message Queue pool
3. Network IO pool

5.4.1. Agreement manager pool

Controls the concurrency of the *Agreement Manager* stage. Agreement manager can launch agreements in parallel.

```
<ThreadPool>
  <AgreementThreads>10</AgreementThreads>
  <AgreementBufferSize>1000</AgreementBufferSize>
</ThreadPool>
```

The `AgreementBufferSize` is the size of the queue of waiters, when all the agreement threads are busy. If the queue becomes full, application will see throttling via proposal submit calls blocking till a slot in the queue or a thread becomes available.

5.4.2. Message Queue pool

Incoming messages from DCone nodes are deposited to a message queue stage. Its concurrency can be controlled via:

```
<MessageQueue>
  <MaxThreads>5</MaxThreads>
  <BufferSize>5000</BufferSize>
  <DepositTimeout>1000</DepositTimeout>
</MessageQueue>
```

`MaxThreads` specifies the maximum number of threads this stage will ever use. `BufferSize` is the size of the queue of waiters. `DepositTimeout` in milli-seconds specifies the timeout at which the message is discarded. Remember – DCone is tolerant to message loss, so this just helps throttle an over-loaded server.

5.4.3. Network IO Pool

This applies to the DConeNet transport. DConeNet supports blocking as well as non-blocking network IO model.

For both the IO models, DCone supports independent tuning of TCP socket reader and writer stages.

Blocking IO

With an efficient thread scheduler like Linux NPTL(Native Posix Thread Library), blocking IO model can be faster! On a Linux kernel with NPTL, we have seen it scale nicely to over 4000 persistent connections. It is the default model. To use it set `UseNonBlockingIO` false or don't specify the option.

```
<DConeNet>
  <!-- This translates to thread per connection model for both
        network reader and writers /-->
  <UseNonBlockingIO>>false</UseNonBlockingIO>
  <ReadStage>
    <!-- In a thread per conn model, this is really the max
          conns we can do -->
    <MaxThreads>5</MaxThreads>
  </ReadStage>
  <WriteStage>
    <!-- -1 means no timeout /
    <ThreadKeepAliveTimeOut>-1</ThreadKeepAliveTimeOut-->
    <!-- In a thread per conn model, this is really the max
          conns we can do -->
    <MaxThreads>5</MaxThreads>
  </WriteStage>
</DConeNet>
```

With blocking IO model, DCone uses a thread-per-connection strategy. `MaxThreads` specifies the maximum number of connections in the read/write stage. To control the lifetime of IO threads, you can set a `ThreadKeepAliveTimeOut` timeout.

Non-Blocking IO

For very large fan-in/fan-out or long latency connections, non-blocking IO model may give better performance. It certainly scales more gracefully. It uses `poll`, `/dev/poll` etc underneath.

```
<DConeNet>
  <!-- Default is blocking IO -->
  <UseNonBlockingIO>true</UseNonBlockingIO>
  <ConnectionKeepAliveTime>1800000</ConnectionKeepAliveTime>
  <ReadStage>
    <ReactorKeepAliveTimeOut>300</ReactorKeepAliveTimeOut>
    <MaxThreads>5</MaxThreads>
  </ReadStage>
  <WriteStage>
    <!-- -1 means no timeout /-->
    <ThreadKeepAliveTimeOut>1000</ThreadKeepAliveTimeOut>
    <MaxConnectionsPerThread>6</MaxConnectionsPerThread>
    <MaxThreads>5</MaxThreads>
    <!-- default is 100 /-->
    <MaxWriteMessagesOutstanding>50000</MaxWriteMessagesOutstanding>
    <UseBlockingConnect>>false</UseBlockingConnect>
  </WriteStage>
```

`ConnectionKeepAliveTime` specifies the inactivity timeout for an idle persistent connection. If the persistent connection has not seen any read or write activity in the specified time interval, it is closed. This may cause the connectivity agent to create a new connection if the endpoint is left without a single connection. This fact can be used to deal with buggy NAT/port forwarding devices that reset connections without sending a TCP level reset to endpoints.

`ReactorKeepAliveTimeOut` specifies the idle time for a read reactor. With non-blocking IO, reactive IO is used for reads. Each reactor will handle multiple connections. `MaxThreads` really means the maximum number of read reactors.

`MaxThreads` for the write stage specifies the maximum number of write reactors that can be active at the same time. `MaxWriteMessagesOutstanding` specifies the threshold at which a new write reactor will be created. For a long latency WAN, `UseBlockingConnect` set to false will ensure TCP connection establishment happens in the background with blocking.

5.5. Recovery and Persistence

DCone comes bundled with a built-in recovery journal and an object database. It is also possible to configure it to use an external JDBC data source. Internally, DCone maintains two repositories – *proposal* and *agreement* repositories.

Common Recovery options are:

```
<Recovery>
  <isEnabled>true</isEnabled>
  <!-- resetRepositories if true will nuke existing repositories -->
  <resetRepositories>true</resetRepositories>
  <trackApplicationStatus>true</trackApplicationStatus>
  <useJdbc>true</useJdbc>
  <proposalRepositoryType>JDBC</proposalRepositoryType>
  <agreementRepositoryType>RecoveryJournal</agreementRepositoryType>
```

WANdisco Distributed Agreement Engine Administration Guide

```
<!-- db vendor-neutral config goes here -->
<JdbcStore>
  <ds-user></ds-user>
  <ds-password></ds-password>
  <ds-url>/mysql/synod</ds-url>
</JdbcStore>
</Recovery>
```

To enable or disable, set `isEnabled` to `true` or `false`. Warning: a node can not recover from failure if this setting is `false`. `resetRepositories` lets you reset all the underlying persistent repositories. Warning: if set to `true`, all recovery context of a node will be deleted at startup time, so, in effect, a node can not recover from failure. `trackApplicationStatus` is used by applications to track completion of an agreement step from application's view point. DCone will re-deliver the agreement proposal, if the application status bit indicates *not done*. This is typically used in conjunction with our programmatic API to set the bit from within the application.

`useJdbc` if set to `true` will cause JDBC data sources to be registered with DCone's JNDI provider.

`agreementRepositoryType` and `proposalRepositoryType` can be set to `JDBC` or `RecoveryJournal`.

If a JDBC based recovery repository is being used, JDBC database vendor-neutral configuration can be specified with the `Recovery` element as:

```
<Recovery>
....
  <!-- db vendor-neutral config goes here -->
  <JdbcStore>
    <ds-user></ds-user>
    <ds-password></ds-password>
    <ds-url></ds-url>
  </JdbcStore>
</Recovery>
```

The usual JDBC data source `user/password` and URL can be specified here.

5.5.1. Built-in Recovery Journal

This is considerably faster and lighter-weight than using JDBC based recovery. Since disk IO has considerable impact on performance, we provide several options to tune the recovery journal and maximize disk throughput.

```
<Recovery>
  <isEnabled>true</isEnabled>
  <useJdbc>false</useJdbc>
  <proposalRepositoryType>RecoveryJournal</proposalRepositoryType>
  <agreementRepositoryType>RecoveryJournal</agreementRepositoryType>
</Recovery>

<RecoveryJournal>
  <UseSynchThread>..</UseSynchThread>
  <UseNIO>..</UseNIO>
  <JournalDir>..</JournalDir>
  <flushMethod>..</flushMethod>
  <AlignBlock>..</AlignBlock>
  <BlockSize>..</BlockSize>
```

WANdisco Distributed Agreement Engine Administration Guide

```
<BucketSize>..</BucketSize>
<MaxJournalFileSize>...</MaxJournalFileSize>
<DiskMonEnable>..</DiskMonEnable>
<DiskMonInterval>..</DiskMonInterval>
<DiskMonCriticalLevel>..</DiskMonCriticalLevel>
<DiskMonWarningLevel>..</DiskMonWarningLevel>
</RecoveryJournal>
```

The built-in recovery journal can be configured with the following prefs:

`UseSynchThread` by default is `true`. If there is concurrency within the DCone stage, turning this option can improve disk throughput considerably. It allows our persistence mechanism to club multiple small writes into fewer bigger writes. If the application is completely sequential then turning this option off (`false`) and tuning other prefs may be a better idea.

`UseNIO` if set to `true`, lets you take advantage of other IO options like `flushMethod`, `AlignBlock`, `mmap` IO for reads.

`flushMethod` can be set to `rws` or `rwd` or `fsync` or `fdatasync`. The `rws` and `rwd` options can only be used with `UseSynchThread` set to `false`.

The `rws` and `rwd` option basically map to POSIX `O_SYNC` or `O_DSYNC` options respectively. What that means it use synchronous IO with or without metadata sync.

The `fsync` and `fdatasync` options have the usual POSIX semantics. They can only be used with `UseSynchThread` and `UseNIO` set to `true`.

If using `UseNIO` option, considerable performance improvement can be obtained via setting `AlignBlock` to `true`. Default is `false` as it results in very large journal files. Default block size is 512 bytes. It can be tuned via `BlockSize` specified in bytes.

The `JournalDir` specifies the path to a directory that will contain the DCone journals. Default is to use the value specified by system property, `java.io.tmpdir`.

The `BucketSize` option specifies how many objects per bucket. Default is 10000 objects. Setting it to a large value will increase memory footprint, smaller value could lead to more disk IO.

The default `MaxJournalFileSize` is 500Mbytes. It specifies the threshold at which journal files are rotated.

The `DiskMonEnable` option is `true` by default, it turns on disk monitoring for free space. The disk monitoring interval can be specified via `DiskMonInterval`, defaults to 15 minutes. Interval can be specified in milli-seconds or using our abbreviated syntax – `{interval}s|m|h`, for example `600s`. When the disk usage reaches a warning level, web dashboard will show a red-alert, if an email is specified at startup, an email alert is also generated. The default warning level is 75% disk used. When the disk usage reaches a critical level, defaults to 95% full, the DCone process is shutdown to avoid any corruption issue when disk is full. The levels can be adjusted using `DiskMonCriticalLevel` and `DiskMonWarningLevel`. The number specified corresponds to percentage of disk in use.

Distributed Garbage Collection Related Options

For individual repositories, the following options are provided:

```
<RecoveryJournal>
  <AgreementRepository>
    <!-- default is 20000 -->
    <DeactivationCushion>0</DeactivationCushion>
    <!-- default is 3 minutes -->
    <DeactivationInterval>2000</DeactivationInterval>
  </AgreementRepository>

  <ProposalRepository>
    <!-- default is 70000 -->
    <GCCushion>0</GCCushion>
    <!-- default is 10 minutes -->
    <GCInterval>5000</GCInterval>
  </ProposalRepository>
</RecoveryJournal>
```

The agreement repository caches information (concluded agreement instances) needed to help other replicas recover from failures. If this information is not cached, it can still be read from disk.

`DeactivationCushion` specifies the minimum number of concluded agreement instances cached. `DeactivationInterval` specifies how often the concluded agreement instances are identified for removal from the cache.

The proposal repository can be garbage collected locally without needing distributed coordination. The cushion and interval settings have the same semantics as above.

Performance Tips

1. For maximum performance setup the recovery journal on an separate disk device. If the file system supports journaling, turn it on. For example, on Linux, `ext 3` will give you better performance than `ext 2`.
2. Locate the filesystem journal on a separate device from the filesystem itself.
3. Specify disk block size (4K for example). Doing block size IO is faster (10x) on Linux for example, but results in significantly higher disk usage.
4. Play with various sync methods (`fdatasync`, `fsync` etc)

5.5.2. JDBC based recovery

To use JDBC based repositories for recovery, set `useJDBC` to `true`. Then specify the database vendor specific configuration as below:

```
<Recovery>
  <isEnabled>true</isEnabled>
  <useJdbc>false</useJdbc>
  ....
  <JdbcStore>
    <ds-user></ds-user>
    <ds-password></ds-password>
    <ds-url>/mysql/synod</ds-url>

    <maxConnectionPoolSize>10</maxConnectionPoolSize>
    <connectionWaitTime>-1</connectionWaitTime>
    <connectionKeepAliveTime>10000</connectionKeepAliveTime>
  </JdbcStore>
```

```

</Recovery>

<JdbcConfig>
  <!-- db Vendor specific config goes here -->
  <MySQL>
    <!-- Enter the URL to the Synod Recovery Database -->
    <URL>jdbc:mysql://localhost:3306/synod</URL>
  </MySQL>
</JdbcConfig>

```

Just specify the URL as required by the vendor of the JDBC driver. Make sure the jdbc driver is on the classpath of DCone.

DCone has a built-in implementation of a JDBC connection pool. It is on by default with a `maxConnectionPoolSize` of 100. You can tune the pool prefs by setting `connectionWaitTime` to the time in milli-seconds to wait to grab a JDBC connection. The `connectionKeepAliveTime` specified in milli-seconds controls the idle time for a JDBC connection.

5.6. Distributed Garbage Collection

DCone implements a Distributed Garbage Collection protocol for cleaning up distributed state. The following options are available to tune it:

```

<DistributedGC>
  <enable>>true</enable>
  <!-- default is 2 hours -->
  <interval>10m</interval>
</DistributedGC>

```

It is enabled by default. The distributed garbage collection `interval` can be specified as a number with a suffix `s` or `m` or `h` to denote seconds, minutes or hours.

5.7. FileBasedProposal checksumming

DCone applications which use file-based proposals can enable check-sum validation of proposals as an optional safeguard against the possible corruption of the proposal in transit between DCone nodes.

```

<FileBasedProposal>
  <!-- default is false -->
  <enableCRC32>true</enableCRC32>
  <!-- default is false -->
  <enableMD5>true</enableMD5>
</FileBasedProposal>

```

These prefs are of no consequence to applications which do not use file-based proposals. Both CRC32 and MD5 checksums are disabled by default. There is usually no need to enable these preferences, unless corruption of file-based proposals in transit is suspected.

5.8. Global Sequence

The DCone has a patent-pending global sequence management system. It supports the following prefs:

```

<GlobalSequence>

```

WANdisco Distributed Agreement Engine Administration Guide

```
<HoleFillerTimeout>1m</HoleFillerTimeout>  
<DeliverInLocalSequence>true or false</DeliverInLocalSequence>  
</GlobalSequence>
```

`HoleFillerTimeout` is specified as a number with a suffix `s` or `m` or `h` to denote seconds, minutes or hours. It controls how frequently `DCone` proactively tries to learn the outcomes of missing agreements and, if needed, fills up the *holes* with `no-op` proposals. Holes can be created if, for example, a node goes down for some time and then restarts. In the meantime the overall distributed system may have moved ahead (if not using `Unanimous` quorum). The restarted node then needs to learn the missing agreements or holes.

`DeliverInLocalSequence` if set to `true`, ensures that events submitted for agreement at a local node obey the local submission sequence. If set to `false`, `DCone` still ensures a consistent global ordering but ignores the local sequence. The `CVS Replicator` product sets `DeliverInLocalSequence` to `false`, as there isn't any dependency between incoming local `CVS` requests.