Infinispan Reference Manual

Extended Statistics

Overview

This module adds a sets of attributes, which are useful to monitor the system's state and to profile transactions' costs. The collected attributes are divided in two groups: Profiling Statistics and Top-Key Collection.

Profiling Statistics

This group contains statistics relevant to the transactions' cost. Namely, it measures the local, remote and commit times of each transaction. To enable the collection of these statistics, you need to add the following interceptor in the configuration.

- Full Replication: `org.infinispan.distribution.wrappers.ReplCustomStatsInterceptor`
- Partial Replication: `org.infinispan.distribution.wrappers.DistCustomStatsInterceptor`

Top-Key Collection

This group contains statistics about the most accessed data, including, for example, the most contented key or the keys that originates more validation failures. To enable the collection of these statistics, you need to add the following interceptor in the configuration.

- Full Replication: `org.infinispan.stats.topK.StreamLibInterceptor`
- Partial Replication: `org.infinispan.stats.topK.DistributedStreamLibInterceptor`

Configuration

```
<global>
 ...
 <globalJmxStatistics
   enabled="true"
   jmxDomain="org.infinispan"/>
</global>
<default>
 ...
 <customInterceptors>
  <interceptor
    after="org.infinispan.interceptors.InvocationContextInterceptor"
    class="org.infinispan.distribution.wrappers.DistCustomStatsInterceptor"/>
  <interceptor
    before="org.infinispan.interceptors.NotificationInterceptor"
    class="org.infinispan.stats.topK.DistributedStreamLibInterceptor"/>
 </customInterceptors>
 ...
</default>
```

See the Statistics

The statistics are exposed via JMX. The ExtendedStatistics component is responsible for the Profiling Statistics; the StreamLibStatistics one for the Top-Key.

Available Statistics

Statistic Name	Type	Description
AbortRate	double	Abort Rate
AbortRateParam	double	Abort Rate per class
ApplicationContentionFactor	double	Application Contention Factor
ApplicationContentionFactorParam	double	Application Contention Factor per class
		Average aborted write transaction

AvgAbortedWriteTxDuration	long	Average aborted write transaction duration (in microseconds)
AvgClusteredGetCommandSize	long	Average clustered get command size (in bytes)
AvgClusteredGetCommandSizeParam	long	Average clustered get command size (in bytes) per class
AvgCommitAsync	long	Average asynchronous Commit duration (in microseconds)
AvgCommitAsyncParam	long	Average asynchronous Commit duration (in microseconds) per class
AvgCommitCommandSize	long	Average commit command size (in bytes)
AvgCommitCommandSizeParam	long	Average commit command size (in bytes) per class
AvgCommitRtt	long	Average Commit Round-Trip Time duration (in microseconds)
AvgCommitRttParam	long	Average Commit Round-Trip Time duration (in microseconds) per class
AvgCommitTime	long	Average local commit duration time (2nd phase only) (in microseconds)
AvgCommitTimeParam	long	Average local commit duration time (2nd phase only) (in microseconds) per class
AvgCompleteNotificationAsync	long	Average asynchronous Complete Notification duration (in microseconds)
AvgCompleteNotificationAsyncParam	long	Average asynchronous Complete Notification duration (in microseconds) per class
AvgGetsPerROTransaction	long	Average number of get operations per (local) read-only transaction
AvgGetsPerROTransactionParam	long	Average number of get operations per (local) read-only transaction per class
AvgGetsPerWrTransaction	long	Average number of get operations per (local) read-write transaction
AvgGetsPerWrTransactionParam	long	Average number of get operations per (local) read-write transaction per class
AvgLocalCommitTime	long	Average time it takes to execute the commit command locally (in microseconds)
AvgLocalCommitTimeParam	long	Average time it takes to execute the commit command locally (in microseconds) per class
AvgLocalGetTime	long	Average Local processing Get time (in microseconds)
AvgLocalGetTimeParam	long	Average Local processing Get time

AvgLocalGetTimeParam	long	(in microseconds) per class
AvgLocalLockHoldTime	long	Average lock local holding time (in microseconds)
AvgLocalLockHoldTimeParam	long	Average lock local holding time (in microseconds) per class
AvgLocalPrepareTime	long	Average time it takes to execute the prepare command locally (in microseconds)
AvgLocalPrepareTimeParam	long	Average time it takes to execute the prepare command locally (in microseconds) per class
AvgLocalRollbackTime	long	Average time it takes to execute the rollback command locally (in microseconds)
AvgLocalRollbackTimeParam	long	Average time it takes to execute the rollback command locally (in microseconds) per class
AvgLockHoldTime	long	Average lock holding time (in microseconds)
AvgLockHoldTimeParam	long	Average lock holding time (in microseconds) per class
AvgLockWaitingTime	long	Average time waiting for the lock acquisition (in microseconds)
AvgLockWaitingTimeParam	long	Average time waiting for the lock acquisition (in microseconds) per class
AvgNTCBTime	long	Average NTCB time (in microseconds)
AvgNTCBTimeParam	long	Average NTCB time (in microseconds) per class
AvgNumNodesCommit	long	Average number of nodes in Commit destination set
AvgNumNodesCommitParam	long	Average number of nodes in Commit destination set per class
AvgNumNodesCompleteNotification	long	Average number of nodes in Complete Notification destination set
AvgNumNodesCompleteNotificationParam	long	Average number of nodes in Complete Notification destination set per class
AvgNumNodesPrepare	long	Average number of nodes in Prepare destination set
AvgNumNodesPrepareParam	long	Average number of nodes in Prepare destination set per class
AvgNumNodesRemoteGet	long	Average number of nodes in Remote Get destination set
AvgNumNodesRemoteGetParam	long	Average number of nodes in Remote Get destination set per class
AvgNumNodesRollback	long	Average number of nodes in Rollback destination set

AvgNumNodesRollbackParam	long	Average number of nodes in Rollback destination set per class
AvgNumOfLockLocalTx	long	Average number of locks per write local transaction
AvgNumOfLockLocalTxParam	long	Average number of locks per write local transaction per class
AvgNumOfLockRemoteTx	long	Average number of locks per write remote transaction
AvgNumOfLockRemoteTxParam	long	Average number of locks per write remote transaction per class
AvgNumOfLockSuccessLocalTx	long	Average number of locks per successfully write local transaction
AvgNumOfLockSuccessLocalTxParam	long	Average number of locks per successfully write local transaction per class
AvgNumPutsBySuccessfulLocalTx	long	Average number of puts performed by a successful local transaction
AvgNumPutsBySuccessfulLocalTxParam	long	Average number of puts performed by a successful local transaction per class
AvgPrepareAsync	long	Average asynchronous Prepare duration (in microseconds)
AvgPrepareAsyncParam	long	Average asynchronous Prepare duration (in microseconds) per class
AvgPrepareCommandSize	long	Average prepare command size (in bytes)
AvgPrepareCommandSizeParam	long	Average prepare command size (in bytes) per class
AvgPrepareRtt	long	Average Prepare Round-Trip Time duration (in microseconds)
AvgPrepareRttParam	long	Average Prepare Round-Trip Time duration (in microseconds) per class
AvgPutsPerWrTransaction	long	Average number of put operations per (local) read-write transaction
AvgPutsPerWrTransactionParam	long	Average number of put operations per (local) read-write transaction per class
AvgReadOnlyTxDuration	long	Average successful read-only transaction duration (in microseconds)
AvgReadOnlyTxDurationParam	long	Average successful read-only transaction duration (in microseconds) per class
AvgRemoteCommitTime	long	Average time it takes to execute the commit command remotely (in microseconds)
		Average time it takes to execute

AvgRemoteCommitTimeParam	long	the commit command remotely (in microseconds) per class
AvgRemoteGetRtt	long	Average Remote Get Round-Trip Time duration (in microseconds)
AvgRemoteGetRttParam	long	Average Remote Get Round-Trip Time duration (in microseconds) per class
AvgRemoteGetsPerROTransaction	long	Average number of remote get operations per (local) read-only transaction
AvgRemoteGetsPerROTransactionParam	long	Average number of remote get operations per (local) read-only transaction per class
AvgRemoteGetsPerWrTransaction	long	Average number of remote get operations per (local) read-write transaction
AvgRemoteGetsPerWrTransactionParam	long	Average number of remote get operations per (local) read-write transaction per class
AvgRemoteLockHoldTime	long	Average lock remote holding time (in microseconds)
AvgRemoteLockHoldTimeParam	long	Average lock remote holding time (in microseconds) per class
AvgRemotePrepareTime	long	Average time it takes to execute the prepare command remotely (in microseconds)
AvgRemotePrepareTimeParam	long	Average time it takes to execute the prepare command remotely (in microseconds) per class
AvgRemotePutsPerWrTransaction	long	Average number of remote put operations per (local) read-write transaction
AvgRemotePutsPerWrTransactionParam	long	Average number of remote put operations per (local) read-write transaction per class
AvgRemoteRollbackTime	long	Average time it takes to execute the rollback command remotely (in microseconds)
AvgRemoteRollbackTimeParam	long	Average time it takes to execute the rollback command remotely (in microseconds) per class
AvgRemoteTxCompleteNotifyTime	long	Average time it takes to execute the rollback command remotely (in microseconds)
AvgRemoteTxCompleteNotifyTimeParam	long	Average time it takes to execute the rollback command remotely (in microseconds) per class
AvgResponseTime	long	Average Response Time
AvgRollbackAsync	long	Average asynchronous Rollback duration (in microseconds)
		Average asynchronous Rollback

AvgRollbackAsyncParam	long	duration (in microseconds) per class
AvgRollbackRtt	long	Average Rollback Round-Trip Time duration (in microseconds)
AvgRollbackRttParam	long	Average Rollback Round-Trip Time duration (in microseconds) per class
AvgRollbackTime	long	Average local rollback duration time (2nd phase only) (in microseconds)
AvgRollbackTimeParam	long	Average local rollback duration time (2nd phase only) (in microseconds) per class
AvgTCBTime	long	Average TCB time (in microseconds)
AvgTCBTimeParam	long	Average TCB time (in microseconds) per class
AvgTxArrivalRate	double	Average transaction arrival rate, originated locally and remotely (in transaction per second)
AvgTxArrivalRateParam	double	Average transaction arrival rate, originated locally and remotely (in transaction per second) per class
AvgWriteTxDuration	long	Average successful write transaction duration (in microseconds)
AvgWriteTxDurationParam	long	Average successful write transaction duration (in microseconds) per class
AvgWriteTxLocalExecution	long	Average write transaction local execution time (in microseconds)
AvgWriteTxLocalExecutionParam	long	Average write transaction local execution time (in microseconds) per class
LocalActiveTransactions	long	Number of concurrent transactions executing on the current node
LocalContentionProbability	double	Local Contention Probability
LocalContentionProbabilityParam	double	Local Contention Probability per class
LocalExecutionTimeWithoutLock	long	Local execution time of a transaction without the time waiting for lock acquisition
LocalExecutionTimeWithoutLockParam	long	Local execution time of a transaction without the time waiting for lock acquisition per class
LocalTopGets	java.util.Map	Show the top 20000 keys most read locally by this instance
LocalTopPuts	java.util.Map	Show the top 20000 keys most

LocalTopPuts	java.util.Map	write locally by this instance
LockContentionProbability	double	Lock Contention Probability
LockContentionProbabilityParam	double	Lock Contention Probability per class
NLocalTopGets	java.util.Map	Show the top n keys most read locally by this instance
NLocalTopPuts	java.util.Map	Show the top n keys most write locally by this instance
NRemoteTopGets	java.util.Map	Show the top n keys most read remotely by this instance
NRemoteTopPuts	java.util.Map	Show the top n keys most write remotely by this instance
NTopContendedKeys	java.util.Map	Show the top n keys most contended
NTopLockFailedKeys	java.util.Map	Show the top n keys whose lock acquisition failed
NTopLockedKeys	java.util.Map	Show the top n keys most locked
NTopWriteSkewFailedKeys	java.util.Map	Show the top n keys whose write skew check was failed
NumAbortedTxDueDeadlock	long	The number of aborted transactions due to deadlock
NumAbortedTxDueDeadlockParam	long	The number of aborted transactions due to deadlock per class
NumAbortedTxDueTimeout	long	The number of aborted transactions due to timeout in lock acquisition
NumAbortedTxDueTimeoutParam	long	The number of aborted transactions due to timeout in lock acquisition per class
NumNodes	long	Number of nodes in the cluster
NumberOfCommits	long	Number of committed transactions since last reset
NumberOfCommitsParam	long	Number of committed transactions since last reset per class
NumberOfGets	long	Number of gets performed since last reset
NumberOfGetsParam	long	Number of gets performed since last reset per class
NumberOfLocalCommits	long	Number of local committed transactions since last reset
NumberOfLocalCommitsParam	long	Number of local committed transactions since last reset per class
NumberOfPuts	long	Number of puts performed since last reset
NumberOfPutsParam	long	Number of puts performed since last reset per class
NumberOfRemoteGets	long	Number of remote gets performed since last reset

NumberOfRemoteGetsParam	long	Number of remote gets performed since last reset per class
NumberOfRemotePuts	long	Number of remote puts performed since last reset
NumberOfRemotePutsParam	long	Number of remote puts performed since last reset per class
PercentageSuccessWriteTransactions	double	Percentage of Write transaction executed in all successfully executed transactions (local transaction only)
PercentageSuccessWriteTransactionsParam	double	Percentage of Write transaction executed in all successfully executed transactions (local transaction only) per class
PercentageWriteTransactions	double	Percentage of Write transaction executed locally (committed and aborted)
PercentageWriteTransactionsParam	double	Percentage of Write transaction executed locally (committed and aborted) per class
PercentileLocalRWriteTransaction	double	K-th percentile of local write transactions execution time
PercentileLocalRWriteTransactionParam	double	K-th percentile of local write transactions execution time per class
PercentileLocalReadOnlyTransaction	double	K-th percentile of local read-only transactions execution time
PercentileLocalReadOnlyTransactionParam	double	K-th percentile of local read-only transactions execution time per class
PercentileRemoteReadOnlyTransaction	double	K-th percentile of remote read-only transactions execution time
PercentileRemoteReadOnlyTransactionParam	double	K-th percentile of remote read-only transactions execution time per class
PercentileRemoteWriteTransaction	double	K-th percentile of remote write transactions execution time
PercentileRemoteWriteTransactionParam	double	K-th percentile of remote write transactions execution time per class
RemoteContentionProbability	double	Remote Contention Probability
RemoteContentionProbabilityParam	double	Remote Contention Probability per class
RemoteGetExecutionTime	long	Average cost of a remote get
RemoteGetExecutionTimeParam	long	Average cost of a remote get per class
RemotePutExecutionTime	long	Average cost of a remote put
RemotePutExecutionTimeParam	long	Average cost of a remote put per class
RemoteTopGets	java.util.Map	Show the top 20000 keys most read remotely by this instance

RemoteTopPuts	java.util.Map	Show the top 20000 keys most write remotely by this instance
ReplicationDegree	long	Number of replicas for each key
StatisticsEnabled	void	Show the top n keys whose write skew check was failed
Throughput	double	Throughput (in transactions per second)
ThroughputParam	double	Throughput (in transactions per second) per class
TopContendedKeys	java.util.Map	Show the top 20000 keys most contended
TopKValue	void	Set K for the top-K values
TopLockFailedKeys	java.util.Map	Show the top 20000 keys whose lock acquisition failed by timeout
TopLockedKeys	java.util.Map	Show the top 20000 keys most locked
TopWriteSkewFailedKeys	java.util.Map	Show the top 20000 keys whose write skew check was failed
WriteSkewProbability	double	Write skew probability
WriteSkewProbabilityParam	double	Write skew probability per class

Transaction Protocol

Overview

Passive Replication based commit protocol (a.k.a. Passive Replication, PB)

Passive Replication is a single-master scheme which allows read-only transactions to be executed on top of any node, whereas write transactions are executed exclusively by the primary (or master) node. The master serializes them via a local concurrency control mechanism, and propagates the updates synchronously towards the remaining nodes (typically called the backups).

This protocol behaves better than a multi-master scheme in high contention scenarios, since the master is able to detect locally conflicts in the lock acquisition, thus avoiding expensive distributed lock contentions. On the other hand, given that only one node is entitled to execute update transactions, it may deliver suboptimal throughput in case of write intensive, low contention scenarios.

Two Phase Commit based commit protocol (2PC)

Two Phase Commit is a multi-master scheme which allows any node to process read-only and write transactions. The concurrency control is implemented by means of locking each key in the primary owner; data consistency in ensured through a commit protocol which acts in two phases:

- **Prepare phase**. First, the transaction coordinator (i.e., the node on which the transaction was originated) acquires the locks relevant to the keys whose it is primary owner; then, it performs a local validation. If this succeeds, the transaction's read/write sets are sent to all the other nodes involved in the transaction. Otherwise, the transaction is rolled back. The set of contacted nodes depends on the employed replication scheme: in full replication, it includes all nodes; in partial, it only includes the nodes which are owner of at least one key in the read/write set of the transaction. Upon the receipt of a "prepare" message, a node tries to acquire all the necessary locks and perform the transaction validation. If both operations succeeds, then an "ACK" message is sent back to the coordinator; otherwise a "NACK" message is sent.
- **Commit phase**: the transaction coordinator collects all the replies of the 1st phase. If all of them are "ACK" messages, than a final "COMMIT" message is sent and the modifications of the transaction are persisted in memory; otherwise an "ABORT" message is sent and the tentative modifications of the transactions are discarded.

This multi-master scheme behaves well in low contention scenarios, as all the nodes can serve both read-only and update transactions, and multiple transactions can be validated concurrently. On the other hand, it performs poorly in high contention scenarios, because of the distributed lock contentions, and, consequently, deadlocks, that arise.

Primary Owner

The Primary Owner of a key is the node responsible for acquiring the lock and performing the validation for that key. The

function which maps a key to its primary owner is deterministic. In the particular case of the Full Replication scheme, the primary node is the same for all the keys in the system, and coincides with the coordinator of the underlying JGroups layer .

Total Order based commit protocol (TO)

The Total Order based commit protocol is a multi-master scheme as 2PC above. This protocol relies on the concept of totally ordered delivery of messages which, informally, implies that each node which delivers a set of messages M, delivers them in the same order.

This protocol comes with two advantages. First, transactions can be committed in one phase, as they are delivered in the same order by the nodes that receive them. Second, it totally avoids deadlocks. The weakness point of this protocol is the fact that its implementation relies on a single thread per node which validates and commit transactions. Thus, this protocol delivers best performance in scenarios of "moderate" contentions, in which it can benefit from the single-phase commit and the validator thread is not the bottleneck.

Configuration

The protocol to use is chosen in the Infinispan configuration file.

Passive Replication

```
<default>
  ...
  <locking
      ...
      useLockStriping="false"
      lockAcquisitionTimeout="500"/>
  ...
  <transaction
      ...
      transactionMode="TRANSACTIONAL"
      syncRollbackPhase="false"
      syncCommitPhase="true"
      lockingMode="OPTIMISTIC"
      transactionProtocol="PASSIVE_REPLICATION"/>
  ...
<default>
```

Two Phase Commit

```
<default>
  ...
  <locking
      ...
      useLockStriping="false"
      lockAcquisitionTimeout="500"/>
  ...
  <transaction
      ...
      transactionMode="TRANSACTIONAL"
      syncRollbackPhase="false"
      syncCommitPhase="true"
      lockingMode="OPTIMISTIC"
      transactionProtocol="TWO_PHASE_COMMIT"/>
  ...
<default>
```

Total Order

```
<default>
  ...
  <transaction
      ...
      transactionMode="TRANSACTIONAL"
```

```
6           syncRollbackPhase="false"
7           syncCommitPhase="true"
8           transactionProtocol="TOTAL_ORDER"/>
9       ...
10  <default>
```

Configuration

Parameter	Description
useLockStriping	If true, a pool of shared locks is maintained for all entries that need to be locked. Otherwise, a lock is created per entry in the cache. Lock striping helps control memory footprint but may reduce concurrency in the system.
lockAcquisitionTimeout	Maximum time to attempt a particular lock acquisition (milliseconds)
transactionMode	Configures whether the cache is transactional or not. Possible values are TRANSACTIONAL and NON_TRANSACTIONAL
syncRollbackPhase	If true, the cluster-wide rollback phase in 2PC transactions will be synchronous, so Infinispan will wait for responses from all nodes to which the rollback was sent. Otherwise, the rollback phase will be asynchronous. Keeping it as false improves performance of 2PC transactions.
syncCommitPhase	If true, the cluster-wide commit phase in 2PC transactions will be synchronous, so Infinispan will wait for responses from all nodes to which the commit was sent. Otherwise, the commit phase will be asynchronous. Keeping it as false improves performance of 2PC transactions, since any remote failures are trapped during the prepare phase anyway and appropriate rollbacks are issued.
lockingMode	Determines whether the cache uses optimistic or pessimistic locking. If the cache is not transactional, then the locking mode is ignored. Possible values are OPTIMISTIC and PESSIMISTIC
transactionProtocol	Configures the commit protocol to use. Possible values are TWO_PHASE_COMMIT, TOTAL_ORDER and PASSIVE_REPLICATION

Data Placement Optimizer

Overview

The main goal is of this module is to improve the system performance by reducing the communication costs paid to transfer data items between nodes during the execution of a transaction. Two mechanisms are employed to achieve this goal

1. Moving data on the nodes where they are most accessed
2. Increasing or decreasing the number of owners (i.e., the replication degree)

Moving Data

To support this mechanism, each node maintains a Object Lookup table to keep track of the actual location of moved keys. When data replacement is triggered , this algorithm is performed on each node N.

1. Collect locally the set K composed by the local and remote most accessed keys by N. A key K is "local" to N if N is an owner of K. Otherwise it is "remote".
2. Send to each node N the list of the of keys in K whose N is primary owner, together with the number of remote and local access to them.
3. Receive from each node N information about their accesses to keys K whose N is primary owner
4. Decide the location of the keys whose N is primary owner (this ensures that the replication degree is respected):
    1. remove the keys from the nodes with lower local accesses;
    2. place the keys onto the nodes with higher local accesses.
5. Broadcast this information to all the node.

6. Trigger a state transfer to actually move the data items.

To determine the owners of a key k, the following steps are performed:

1. check in the Object Lookup table entry relevant to k:
    1. if it returns no owners, then it means that the key has never been moved,then the owners are determined by the original consistent hash implemented by Infinispan;
    2. otherwise, return the new owners.

Object Lookup

The Object Lookup maintains a table containing the relation between keys and their current owners. Two implementations are available and they are the following

- **Hash Map Object Lookup**: It keeps the table in a HashMap. The advantages of this implementation are that it keeps trace of the correct current mapping between key and owner and allows a relative faster lookup. The drawback is that the memory footprint grows linearly with the number of moved keys.
- **C5.0 Decision Tree Object Lookup**: It keeps the table in a Decision Tree calculated by the C5.0 Decision Tree. It uses Machine Learner techniques to determine patterns in the keys moved in addition to Bloom Filters to keep track of the keys moved. The advantage is a lower memory footprint with a cost of some incorrect mappings. In order to use this Object Lookup, you should have the C5.0 executable in a folder with permissions to read, write and execute.

Changing Replication Degree

The change of the replication degree for the keys happens by means of a Replication Degree Manager. When triggered, it induces a State Transfer aimed at reaching the desired replication degree for each key. Three cases can occur:

1. the new replication degree is equals to the old one: nothing has to be done, since no data needs to be moved/created/deleted;
2. the new replication degree is higher than the old one: new copies of the data items must be created. The Replication Degree Manager adds the new owners, computed through the original consistent has function of Infinispan, to the owner list returned by the Object Lookup, if any, until the desired replication degree is met.
3. the new replication degree is lower than the old one: some owners should remove the keys. To this end, the Replication Degree Manager simply removes the exceeding owners from the relevant list, until the desired replication degree is met.

Note that the decision on the nodes which have to become new owners for a key or have to dismiss this role is computed on each node, by the means of a deterministic function.

Configuration

The Replication Degree Optimizer is automatically enabled in both configurations.

Hash Map Object Lookup

```
<default>
 ...
 <dataPlacement
   enabled="true"
   objectLookupFactory="org.infinispan.dataplacement.hm.HashMapObjectLookupFactory"
   coolDownTime="30000"
   maxNumberOfKeysToRequest="1000">
 </dataPlacement>
 ...
</default>
```

C5.0 Decision Tree Object Lookup

```
<default>
 ...
 <dataPlacement
   enabled="true"
   objectLookupFactory="org.infinispan.dataplacement.c50.C50MLObjectLookupFactory"
   coolDownTime="30000"
   maxNumberOfKeysToRequest="1000">
 <properties>
  <property
```

```
      name="keyFeatureManager"
      value="org.radargun.cachewrappers.TpccKeyFeaturesManager"/>
    <property
      name="location"
      value="/tmp/ml"/>
    <property
      name="bfFalsePositiveProb"
      value="0.01"/>
  </properties>
 </dataPlacement>
 ...
</default>
```

C5.0 Object Lookup Factory Properties

Property	Description
keyFeatureManager	The Machine Learner needs some keys attributes (Feature) in order to find patterns between them. This manager contains all the features and splits the keys in features. This is mandatory for C5.0!
location	The location of the C5.0 executable.
bfFalsePositiveProb	The Bloom Filter false positive probability (between 0.0 and 1.0)

Configuration

Parameter	Description
enabled	Enables or disables Data Placement.
objectLookupFactory	The ObjectLookupFactory implementation to store the new key mapping.
coolDownTime	The minimum amount of time (in milliseconds) to wait between two consecutive invocation to the Data Placement Optimizer.
maxNumberOfKeysToRequest	The maximum number of the most accessed remote keys to collect.
properties	Some Object Lookup Factories relies on properties for configuration. These properties are passed directly to the Object Lookup Factory.

Add a custom Object Lookup Factory

It is possible to implement a custom Object Lookup Factory. In order to do it, you just need to implement the interface `org.infinispan.dataplacement.lookup.ObjectLookupFactory` to construct the custom Object Lookup that must implement the interface `org.infinispan.dataplacement.lookup.ObjectLookup`.

The Object Lookup Factory has access to all the Configuration parameters and you can add your own as properties without changing any code in Infinispan.

Trigger the optimizer

To trigger the data placement optimizer you need to enable the JMX in Infinispan in the following way:

```
<global>
  <globalJmxStatistics
      enabled="true"
      jmxDomain="{jmx-domain}"/>
  ...
</global>
```

Then, you should create the ObjectName corresponding to the Data Placement:

`<jmx-domain>:type=Cache,name="<cache name>",manager="<cache manager name>",component=DataPlacementManager`

Finally, you have these four methods that you can invoke at runtime

```
void dataPlacementRequest();
void setReplicationDegree(int replicationDegree);
void setCoolDownTime(int milliseconds);
void setMaxNumberOfKeysToRequest(int value);
```

Replication Protocol Optimizer

Overview

The aim of this module is to support the switch among transaction commit protocols at runtime, with low overhead and blocking time for the system. The module supports three kinds of switches.

1. **Fast Switch**: this kind of switch is supported if the programmer can explicitly implement a protocol which allows the switch between two commit protocols without blocking or aborting transactions in the transitory phase between the two.
2. **Stop and Go (lazy)**: this is used when no Fast Switch between two protocols is available (or when its invocation is forced). In this case, the system blocks the execution of new transactions until the transactions running on top of the current protocol have finished their execution. When all nodes signal that all such transactions have finished, all the nodes switch to the new protocol.
3. **Stop and Go (eager)**: this is used when no Fast Switch is available (or when its invocation is forced). In this case, the system blocks the execution of new transactions and aborts the transactions running on top of the current protocol. When all nodes signal that all such transactions are over, all the nodes switch to the new protocol. This mechanisms offers a fast stop and go.

Configuration

Nothing to configure.

Trigger the optimizer

To trigger the optimizer you need to enable the JMX in Infinispan in the following way:

```
<global>
  <globalJmxStatistics
      enabled="true"
      jmxDomain="{jmx-domain}"/>
  ...
</global>
```

Then, you should create the ObjectName corresponding to the Data Placement:

```
<jmx-domain>:type=Cache,name="<cache name>",manager="<cache manager name>",
component=ReconfigurableReplicationManager
```

Finally, you have this three operations that you can execute in runtime:

```
void register(String clazzName);
void switchTo(String protocolId, boolean forceStopTheWorld, boolean abortOnStop);
void setSwitchCoolDownTime(int seconds);
```

Update Serializability (a.k.a GMU)

Overview

This scheme provides serializable isolation level in Infinispan and it can be used with any replication protocol.

The novelty of this version is the core distributed concurrency control scheme, the GMU algorithm, that ensures (i) a consistent evolution of the system's state and (ii) that all the values returned by the read operations of a transaction T always belong to a consistent state. More precisely, this means that changes to the state of the data grid always produce consistent snapshots, namely states resulting from the commit of a set of update transactions on multiple nodes as if they were executed sequentially in a single centralized transactional node. In addition, as in classical multiversion concurrency

control schemes (MVCC), this scheme entails storing multiple versions for a same data item and allows transactions to always observe consistent snapshots of the data grid. Therefore, transactions that only execute read operations, namely read-only transactions, can be always successfully committed without incurring in any inter-replica synchronization process and as if they were executed instantaneously between their begin and the commit request.

However, unlike most of the existing distributed MVCC schemes, GMU does not order transactions commit events by relying on a centralized or fully replicated global clock. Conversely, it is designed as a novel, highly scalable, fully distributed synchronization scheme that exploits vector clocks to achieve a twofold objective: (i) determining which data item versions have to be returned by read operations issued by transactions; (ii) ensuring agreement among the nodes replicating the data items updated by a transaction T on the scalar clock to be used when locally applying the write-set of T , as well as on the vector clock to associate with the commit event of T.

Configuration

```
<default>
  ...
  <locking
      ...
      isolationLevel="SERIALIZABLE"/>
  <versioning
      enabled="true"
      versioningScheme="GMU"/>
  <garbageCollector
      enabled="true"
      transactionThreshold="1000"
      versionGCMaxIdle="60"
      viewGCBackOff="120"
      l1GCInterval="60"/>
  ...
</default>
```

</table

Configuration

Parameter	Description
isolationLevel	Cache isolation level. Possible values are READ_COMMITTED, REPEATABLE_READ and SERIALIZABLE.
versioningScheme	The scheme to use when versioning entries. Possible values are NONE, SIMPLE and GMU.
transactionThreshold	The number of transactions committed to trigger the garbage collection of old values.
versionGCMaxIdle	The maximum idle time (in seconds) between two consecutive garbage collection of old values.
viewGCBackOff	The time to wait (in seconds) to trigger the garbage collection of old cache views, after a view change event occurs.
l1GCInterval	The time interval (in seconds) between garbage collection of old value in L1 Cache.