

Administrator's Guide

Introduction	4
vyhodb distribution package, installation and starting.....	5
Download JRE	5
Download vyhodb	5
Configure path to JRE	5
Run	5
vyhodb directory layout	6
Architecture	7
Running modes.....	8
Standalone	8
Embedded mode	9
Local mode	10
Transaction types	11
Data model	11
Configuring	13
Configuration properties.....	13
Memory usage and configuring	14
JVM system properties.....	15
Components	16
RSI Client API	16
Local	16
TCP Single	17
TCP Pooled	17
TCP Balancer.....	17
RSI Server	22
RSI Service deployment.....	22
Configuration properties.....	22
Space API.....	24
Modify Records Cache.....	24
Dictionary	24
Page Storage.....	26
Storage	26
Data file	26
Log file	27

Caches and Buffers	28
Lock Manager	30
Admin	31
Slave Replication Agent (SRA)	32
Command line utilities	33
Slave list file.....	35
Replication.....	36
MASTER server	36
SLAVE server.....	37
SLAVE server configuring.....	37
Slave Storage	38
Slave Replication Agent (SRA)	38
SRA running modes and configuring	38
MASTER server unavailability.....	39
Out of Sync Error	39
Clustering	41
Cluster configuring	42
Location of balancer configuration file	42
Cluster configuration example	43
Inconsistent reading.....	44
Problem solving.....	44
Fault tolerance	45
Read-Only Cluster.....	45
Logging	47
Appendix A. System limits.....	48

Introduction

This guide describes vyhodb architecture, its components, their configuring and administration.

"Vyhodb" is a database management system which uses the network data model, supports ACID transactions, written in Java and is designed to be used by Java applications.

This guide is included in the vyhodb documentation package which consists of the following documents:

Document	Description
Getting Started	Fast start. Document gives idea what is vyhodb API about using simple code examples.
Developer's Guide	Describes different vyhodb APIs and how to use them.
Functions Reference	Functions API Reference. Describes functions with usage examples.
Administrator's Guide	Describes vyhodb architecture, configuring and administration.

vyhodb distribution package, installation and starting

Download JRE

vyhodb requires JRE version not lower than 7u64. The latest version, as well as instructions for installing JRE can be found here <http://www.oracle.com/technetwork/java/javase/downloads/index.html>.

Download vyhodb

The latest version of the vyhodb distribution package can be downloaded from [download](#) page. The distribution package is a zip file with directory **vyhodb-0.9.0** inside.

Configure path to JRE

After unzipping **vyhodb-0.9.0** directory, you need to specify path to JRE:

Windows:

- 1) Open file **bin-cmd\set-env.cmd**
- 2) Set **JRE_HOME** variable. For example: SET JRE_HOME=C:\Program Files\Java\jdk1.7.0_60\jre

Linux:

- 1) Open file **bin-sh/set-env.sh**
- 2) Set **JRE_HOME** variable. For example: JRE_HOME=/home/jdk1.7.0_79/jre

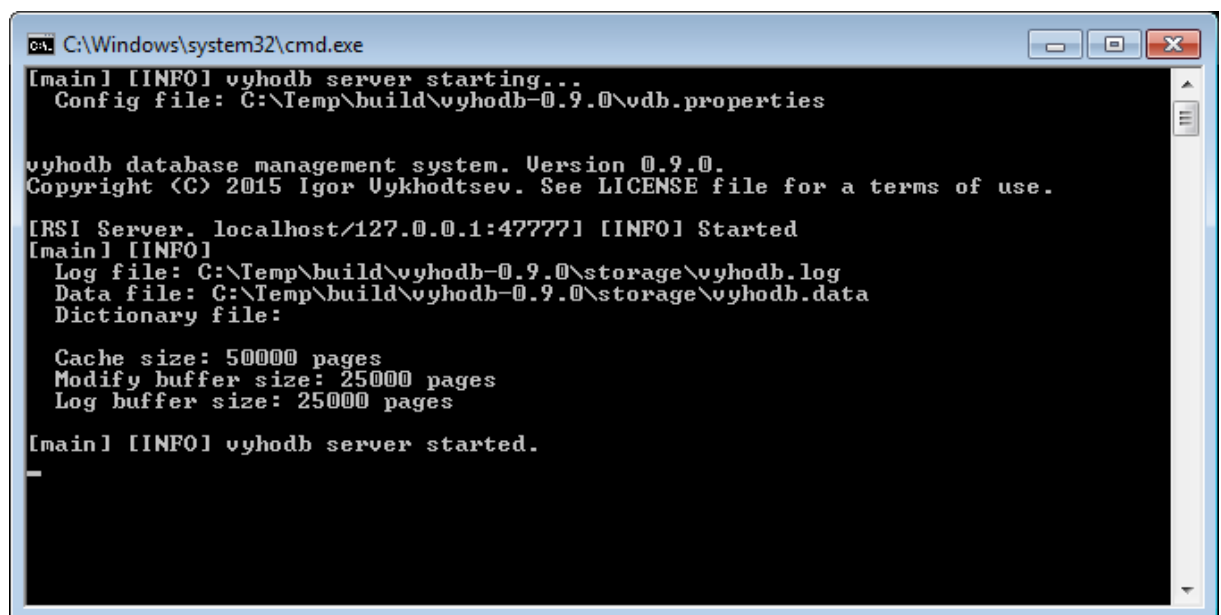
Run

Now you can start vyhodb server in so called stand-alone mode. Run one of the following scripts:

Windows: **vdb-start.cmd**

Linux: **vdb-start.sh**

Picture below shows console output of running vyhodb server:



```

C:\Windows\system32\cmd.exe
[main] [INFO] vyhodb server starting...
Config file: C:\Temp\build\vyhodb-0.9.0\vdb.properties

vyhodb database management system. Version 0.9.0.
Copyright (C) 2015 Igor Vykhodtsev. See LICENSE file for a terms of use.

[RSI Server, localhost/127.0.0.1:47777] [INFO] Started
[main] [INFO]
Log file: C:\Temp\build\vyhodb-0.9.0\storage\vyhodb.log
Data file: C:\Temp\build\vyhodb-0.9.0\storage\vyhodb.data
Dictionary file:

Cache size: 50000 pages
Modify buffer size: 25000 pages
Log buffer size: 25000 pages

[main] [INFO] vyhodb server started.

```

To stop vyhodb server use **Ctrl+C** combination or **vdb-close-remote** script (can be found in **bin-cmd** or **bin-sh** directories).

vyhodb directory layout

File	Description
bin-cmd	windows command line utilities
bin-sh	linux command line utilities
lib	vyhodb system jar files
services	Directory for RSI Services jar files
storage	Default directory for storage files (data file and log file)
LICENSE	vyhodb license agreement
vdb.properties	vyhodb configuration file
vdb-start.cmd	windows start script
vdb-start.sh	linux start script

Architecture

Diagram below illustrates vyhodb server running in standalone mode, and its components:

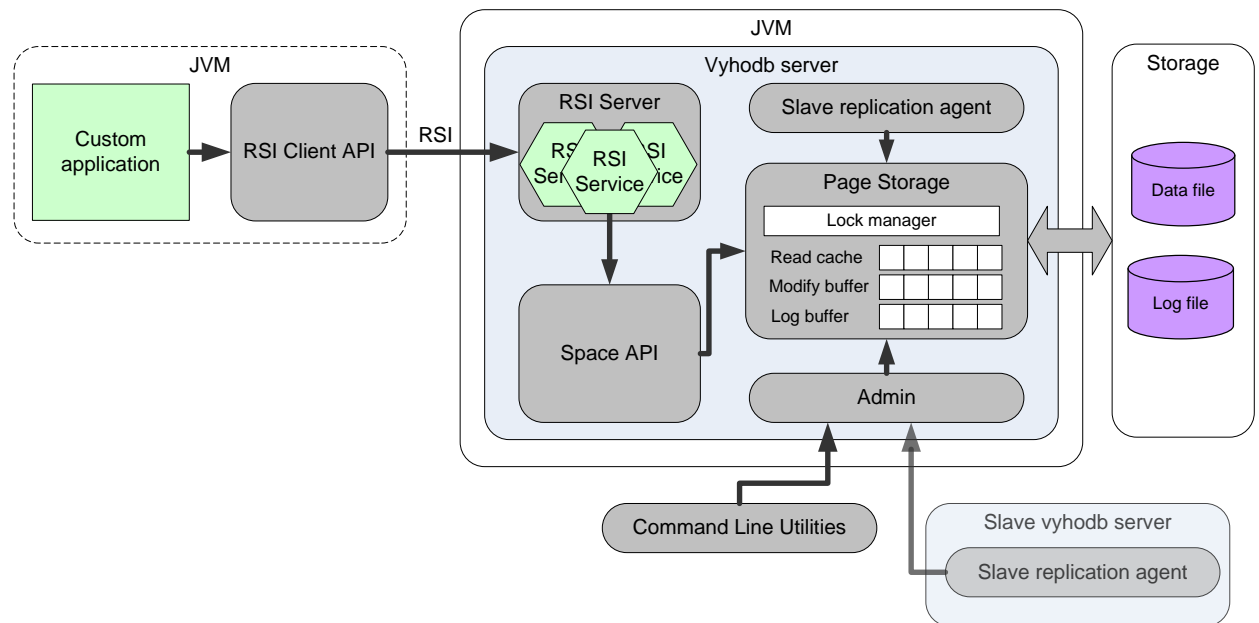


Table below gives brief component description. Components are described in more details in **Components** section.

Component	Description
RSI Client API	“External” component. Provides API to client application for remote invocation of RSI Services.
RSI Server	Accepts network connections from RSI Client API, instantiates RSI Service objects and invokes their methods.
Space API	Implements Space API. It transforms reading/modifying vyhodb data model into storage reading/writing requests (which are page oriented).
Page Storage	Implements lock management, page reading/writing (including caching and buffering) and transactions management.
Admin	Accepts network connections from command line utilities and Slave Replication Agents and handles their requests.
Slave Replication Agent (SRA)	Participates in replication and replicates changes from MASTER server into SLAVE server.
Command line utilities	Command line utilities. Perform administration tasks like creation backup files, new storage, etc. Can operate on closed storage (without running vyhodb server) or connect to Admin component on running vyhodb server.

Diagram above also shows elements, which are not vyhodb components:

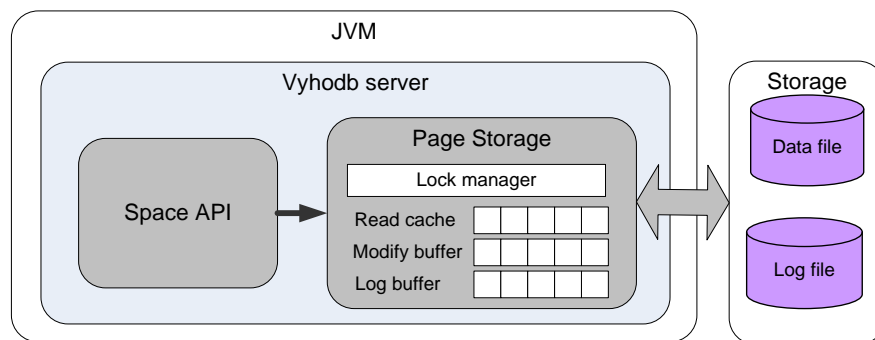
Element	Description
Custom Application	Uses vyhodb for storing and processing data according to business logic. Business logic is partly implemented as RSI Services (in standalone running mode).
RSI Service	Java class, which object is instantiated by RSI Server for remote method invocation. RSI Service methods implement business logic, which is executed on server side. RSI Service uses Space API to read and modify vyhodb data.
RSI	Remote service invocation mechanism.

	Propriety technology for remote method invocations.
Data file, Log file	Files, which are used by vyhodb for storing data (data file) and journaling transactions (log file).
Storage	Common term for data and log files.

The following components are not required and could be switched off/on by using configuration properties (see section **Configuring**):

- 1) RSI Server
- 2) Admin
- 3) Slave Replication Agent (SRA)

So, in minimal configuration, vyhodb server would look like:



Running modes

vyhodb supports three running modes:

- 1) Standalone
- 2) Embedded
- 3) Local

Standalone

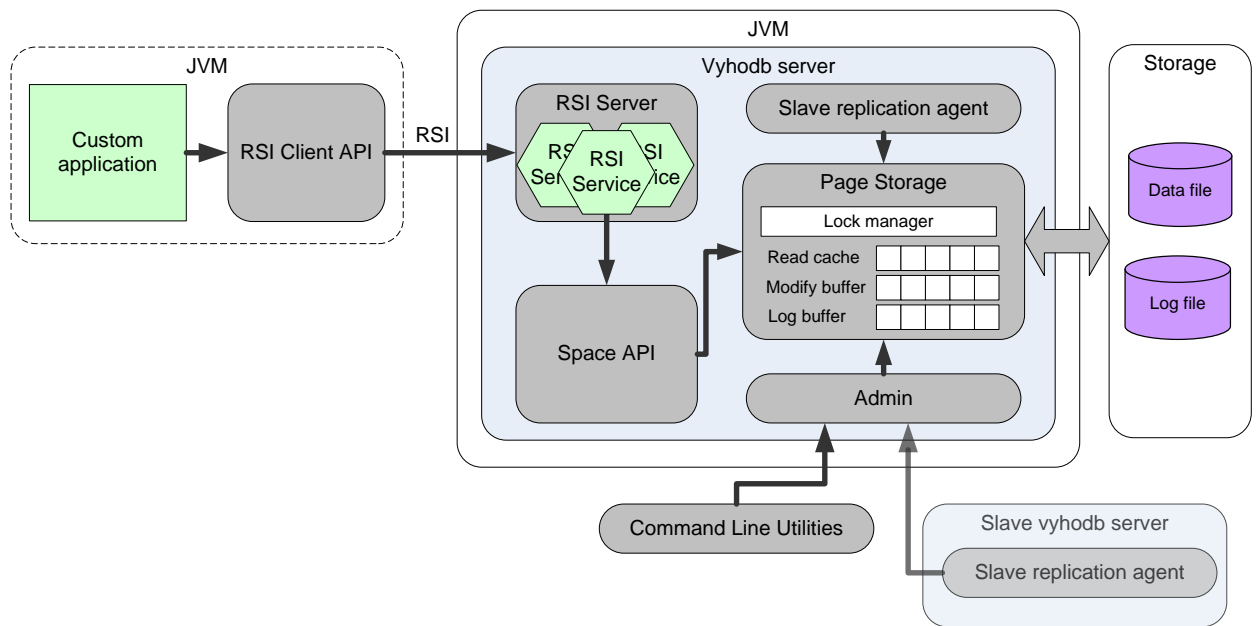
In this mode, custom application and vyhodb server are physically separated and running in different JVM. Custom application accesses vyhodb data by invoking methods of remote RSI Service. RSI Service realizes business logic and accesses data by Space API.

Scripts **vdb-start.cmd/vdb-start.sh** starts vyhodb server in standalone mode.

Configuring vyhodb server in this mode is performed by changing java properties file **vdb.properties**. Configuration properties are described in **Configuring** section.

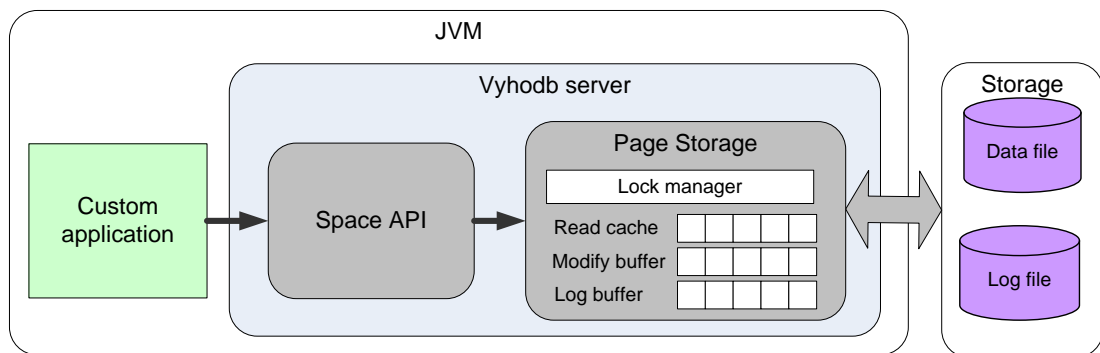
Jar archives with RSI Service classes should be placed in **services** directory in order to be available for vyhodb server.

Diagram below shows vyhodb server running in standalone mode (this is actually the diagram from **Architecture** section):



Embedded mode

Diagram below shows vyhodb server running in embedded mode (the following components are switched off: SRA, RSI Server, Admin):



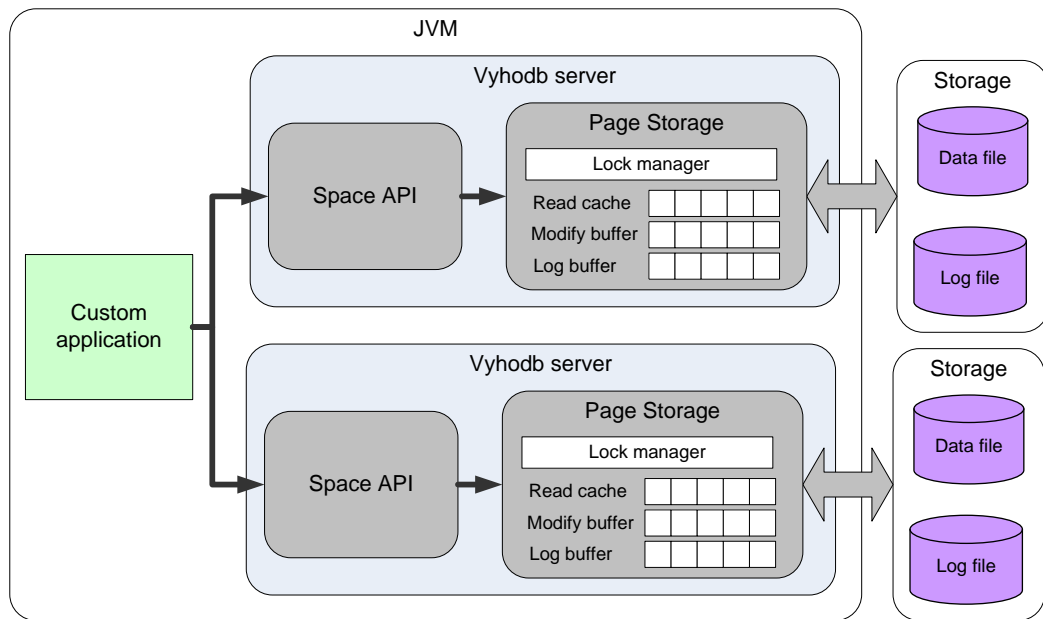
In embedded mode, custom application and vyhodb server are running in the same JVM. Custom application uses Space API directly for reading/modifying vyhodb data.

Custom application uses Server API for starting vyhodb in embedded mode and for transaction management (see “Developer’s Guide” document for more details).

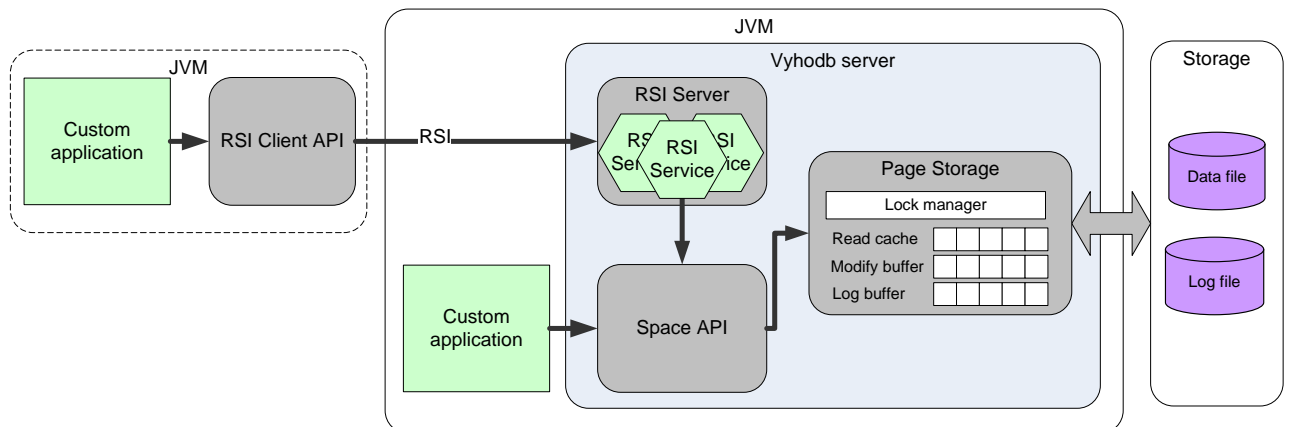
Configuring is done by creating **java.util.Properties** object with configuration properties and passing it into Server API. Configuration properties are common for any running modes. See more information about configuration properties in **Configuring** section.

Despite of the fact that vyhodb server is running “inside” custom application in embedded mode, it still can participate in replication (in both roles: MASTER and SLAVE) and be used by remote command line utilities. Admin component must be switched on for this.

Custom application can start many vyhodb embedded servers, if all of them are running against different storages. See diagram below:



Furthermore, embedded vyhodb server can have running RSI Server component (it is turned off by default in embedded mode). Remote application can access embedded vyhodb server by using RSI in this case:



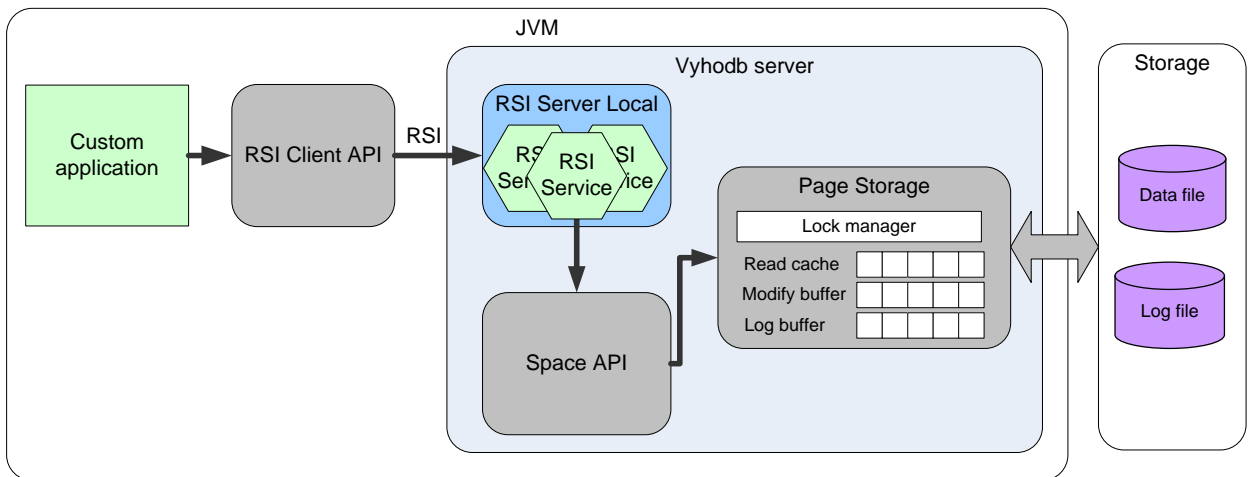
Local mode

In this mode custom application and vyhodb server are running inside the same JVM. They use RSI to communicate with each other.

To start vyhodb server in this mode, custom application creates special type of RSI connection, which is actually a reference to vyhodb server (see [Local](#) section for more details).

Vyhodb configuring in this mode is performed by external properties file (configuration properties are common for all running modes), path to which is passed in URL connection string.

Diagram below shows vyhodb server running in Local mode:



RSI Server Local component is highlighted on the diagram above, because it differs from ordinary RSI Server component. The difference between them is that **RSI Server Local** component doesn't open server sockets and doesn't accept any network connections.

It's also possible to activate **RSI Server** (in addition to **RSI Server Local**) component which would accept and handle RSI connections from remote custom applications. Configuration property **rsi.enabled=true** should be specified in order to activate **RSI Server** component.

Transaction types

Vyhodb supports two transaction types:

- Read
- Modify

Separation on read and modify transactions allows vyhodb to optimize locks, caching and many other aspects of transaction processing.

In case of embedded mode, custom application explicitly specifies which type of transaction should be open, using Server API.

In case of RSI communication (in standalone and local modes), RSI Service's contract defines (by using special Java annotations) which remote methods are invoked under which transaction types.

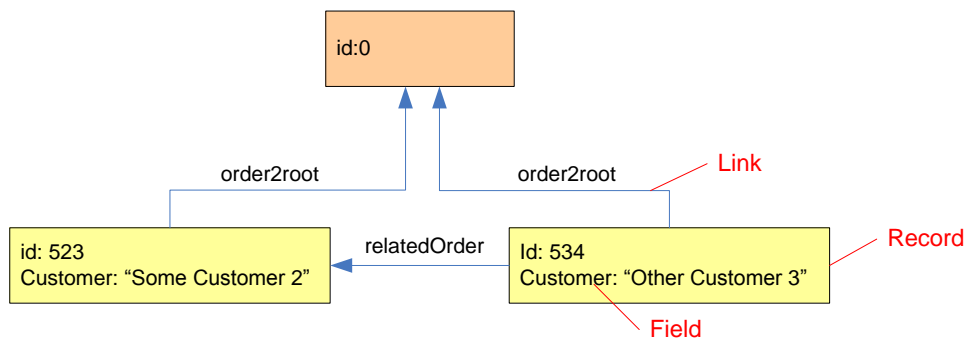
Transactions are bounded by one RSI invocation and can't span over many ones.

Data model

Vyhodb uses network data model for storing data. Vyhodb data model is schemaless (no types for records) and is comprised of the following entities:

- 1) **Space** – container of records (storage). Is used to create new records and retrieve record by its id.
- 2) **Record** – main concept, it is used to model some entity from real world. Similar to table's row in database concept. Has unique identifier (long type) and contains fields.
- 3) **Field** – named value which is stored inside record.
- 4) **Link** – connect two records by child->parent link.
- 5) **Indexes** – used for indexing children records of particular parent records to speed up searching.

Picture below shows Space (storage) with three records, which have fields and connected to each other by links:



To get more information about manipulating vyhodb data see “Developer’s Guide” document.

Configuring

Configuration properties are used for configuring vyhodb server. They can be specified in java properties file or passed as **java.util.Properties** object (depending on running mode). Names and meaning of configuration properties are the same for all running modes.

Only two properties are mandatory for starting vyhodb server and must be explicitly specified:
storage.data(path to data file), **storage.log** (path to log file).

Configuration properties

Table below gives brief information about vyhodb configuration properties, grouped by components.
 For more details about particular property see section of appropriate component.

Configuration property	Description	Default value
exitVmOnCriticalException	Specifies whether to stop JVM in case of vyhodb critical error. By default, in standalone mode, this property is set to true .	false
RSI Server		
rsi.enabled	Switches on/off RSI Server component.	false
rsi.host	RSI Server listening host name.	localhost
rsi.port	RSI Server listening port.	47777
rsi.backlog	RSI Server socket's backlog	100
rsi.cluster.enabled	Switches on/off inconsistency reading safety mechanism. Applicable only for cluster configuration. For more details see Inconsistent reading section.	false
rsi.cluster.probe.attempts	Number of attempts for probing whether required transaction has replicated or not. This property is applicable only when [rsi.cluster.enabled=true] For more details see Inconsistent reading section.	40
rsi.cluster.probe.timeout	Timeout (in milliseconds) between probes whether requires transaction has replicated or not. This property is applicable only when [rsi.cluster.enabled=true] For more details see Inconsistent reading section.	100
Page Storage		
storage.data	Path to data file.	
storage.log	Path to log file.	
storage.readDescriptorCount	Count of file descriptors which are opened for reading data file.	20
storage.durable	Specifies whether vyhodb should call fsync() on log file after modify transaction commit. By default this property is false and fsync() is called on log file only at checkpoints.	false

storage.cacheSize	Read cache size (in pages ¹).	50000
storage.modifyBufferSize	Modify buffer size (in pages).	25000
storage.logBufferSize	Log buffer size (in pages).	25000
storage.lock.timeout	Lock acquisition timeout (in seconds)	120
Space API		
space.dictionary	Path to directory file	
space.record.modifyCacheSize	Record cache's max size in modify transactions. Measure unit – count of vyhodb record.	300
space.record.maxRecordSize	Record's max size (in pages).	16384 (16 mb)
Admin		
admin.enabled	Switches on/off Admin component.	false
admin.host	Admin listening host name.	localhost
admin.port	Admin listening port.	46666
admin.backlog	Admin socket's backlog.	20
admin.connectionBufferSize	Admin's page buffer size (in pages).	64
Slave Replication Agent (SRA)		
slave.enabled	Switches on/off SRA component.	false
slave.master.host	MASTER host name	localhost
slave.master.port	Admin component's listening port on MASTER server.	46666
slave.mode	SRA mode. Two modes are supported: realtime, cron.	realtime
SRA realtime mode		
slave.ttl	SRA connection time-to-live (in milliseconds).	86400000 (24 hours)
slave.checkTimeout	Timeout between new transaction check (in milliseconds).	1000 (1 second)
SRA cron mode		
slave.cron	Cron expression for synchronization sessions.	* * * * *

In embedded mode, configuration properties are passed into Server API as `java.util.Properties` object (see "Developer Guide").

In standalone and local modes, configuration properties are specified in external file, which format is compatible with java properties file.

By default, in standalone mode, this file is located in root vyhodb directory and is **vdb.properties**. Location and name of configuration file can be changed by running **vdb-start** scripts with option **-config**.

Vyhodb server doesn't support changing configuration parameters on-flight: server must be restarted in order to apply new configuration parameters.

Memory usage and configuring

Vyhodb developed with one rule in mind – **all objects must die young**.

Vyhodb doesn't cache any objects in Java Heap Memory outside of the transaction scope. All needed objects are created from scratch (transaction space, records, links, fields, etc.) by reading appropriate data file pages from disk or cache/modify buffer.

¹ Data file is split on pages. Page's size is 1024 bytes.

Using default Java ergonomic setting should be suitable for most applications. However if your application needs specific memory settings or garbage collector parameters, than you can specify them in **vdb-start** scripts at a line of starting JVM.

Anyway, following guidelines below when configuring JVM behavior:

- Always use server JVM (option -server)
- Prefer throughput ergonomic goal and throughput garbage collector rather than low pause collector
- Keep large eden and survival spaces and small old generation space size

Modify buffer, Log buffer and Read cache (see [Caches and Buffers](#)) are allocated outside of JVM garbage collected heap (DirectByteBuffer(s) objects are used for these purposes).

By default, JVM limits memory which could be allocated for Direct Buffers. It might lead to OutOfMemoryError at vyhodb start time. To overcome this problem and increase JVM limit, JVM should be started with the following parameter:

-XX:MaxDirectMemorySize=size[g|G|m|M|k|K]

In case of standalone mode, this parameter should be specified in **vdb-start** scripts.

JVM system properties

Some aspects of system behavior are configured by Java system properties, because they are used at both client and server sides. Most of them relate to RSI mechanism. Table below shows those system properties and their default values:

Parameter	Description	Default value
RSI Client API / RSI Server		
com.vyhodb.rsi.max_chunk	RSI chunk size for input/output interactions (in bytes). RSI implementation uses old-style blocking sockets. According to http://www.evanjones.ca/software/java-bytebuffers.html article, it is better to use arrays of particular size in input/output operations, otherwise native implementation would use malloc() and free() for each read/write invocation. This property sets size of such array. It is required that for Linux 64bit systems administrator explicitly set this property to 64 kb.	2048 - for windows 8192 - for other OS
com.vyhodb.rsi.kryo.init_buffer_size	Initial kryo serialization buffer size (in bytes).	16384 (16k)
com.vyhodb.rsi.kryo.max_buffer_size	Max kryo serialization buffer size (in bytes).	33554432 (32m)
com.vyhodb.rsi.pool.size	Maximum connections in pooled connection. Used, when size of particular connection pool isn't specified (see TCP Pooled).	1
com.vyhodb.rsi.pool.ttl	Connection time-to-live in pooled connection (in milliseconds). Used, when TTL of particular connection pool isn't	86400000 (24 hours)

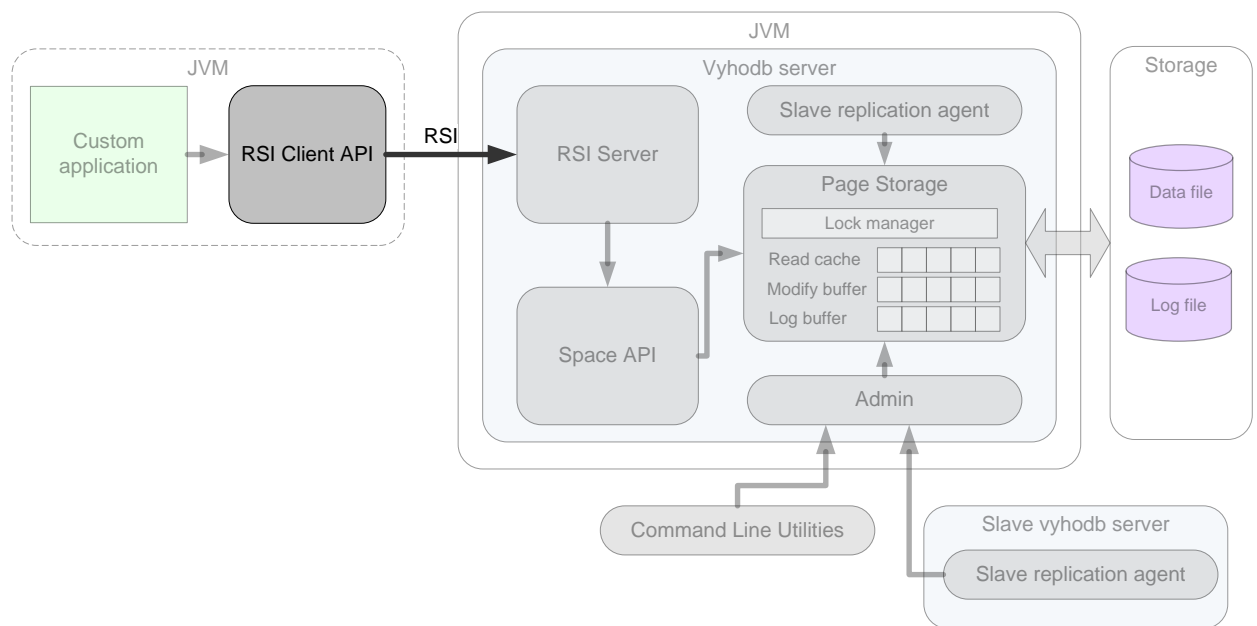
	specified (see TCP Pooled).	
com.vyhodb.rsi.pool.debug	Prints debug info for pooled connection lifecycle.	false

Components

RSI Client API

This component is used by custom application for remote invocation of RSI Service methods.

It establishes network connections to RSI Server. Connection type and its parameters are specified by URL which is passed into RSI Client API at connection create time.



Currently the following connection types are supported:

Type	Description
Local	Is used for start vyhodb in Local mode. It doesn't create network connection.
TCP Single	Creates single TCP connection to remote RSI Server.
TCP Pooled	Creates TCP connections' pool to remote RSI Server.
Balancer	Creates and supports pools of TCP connections with many RSI Servers. It is used for balancing RSI requests over many vyhodb servers. It is a critical part of vyhodb cluster configuration along with vyhodb replication technology.

Local

The following URL is used for creation local connect and starting vyhodb server in local mode:

local:[vyhodb config file URL]

where **[vyhodb config file URL]** is an URL which specifies vyhodb configuration file.

Examples:

local:http://192.168.0.101/vdb/vdb.properties

local:file:c:/vdb-0.9.0/vyhodb.properties

Closing local connection leads to shutdown of vyhodb server.

TCP Single

The following URL pattern is used to create single TCP connection:

tcp://[hostname]:[port]

where **[hostname]** is a RSI Server host name, **[port]** is a RSI Server port number.

Example:

tcp://localhost:47777

TCP connection is opened until it explicitly closed by custom application.

TCP Pooled

To create pooled connection use URL pattern:

tcp:[hostname]:[port]/?[parameters]

where **[hostname]** is a RSI Server host name, **[port]** is a RSI Server port number, **[parameters]** - connection parameters, separated by &. The following parameters are supported:

Parameter	Description
pool=true	Mandatory parameter. Specifies that current RSI connection is pooled.
poolSize=10	Max count of TCP connections in pool.
poolTTL=360000	Max TCP connection time-to-live (in milliseconds).
poolDebug=true	Switches on/off (true/false) output of debug info onto System.out. By default it is switched off.

Example:

tcp://localhost:47777/?pool=true&poolSize=10&poolTTL=360000

New connections in pool are created only when there are no free ones in a pool.

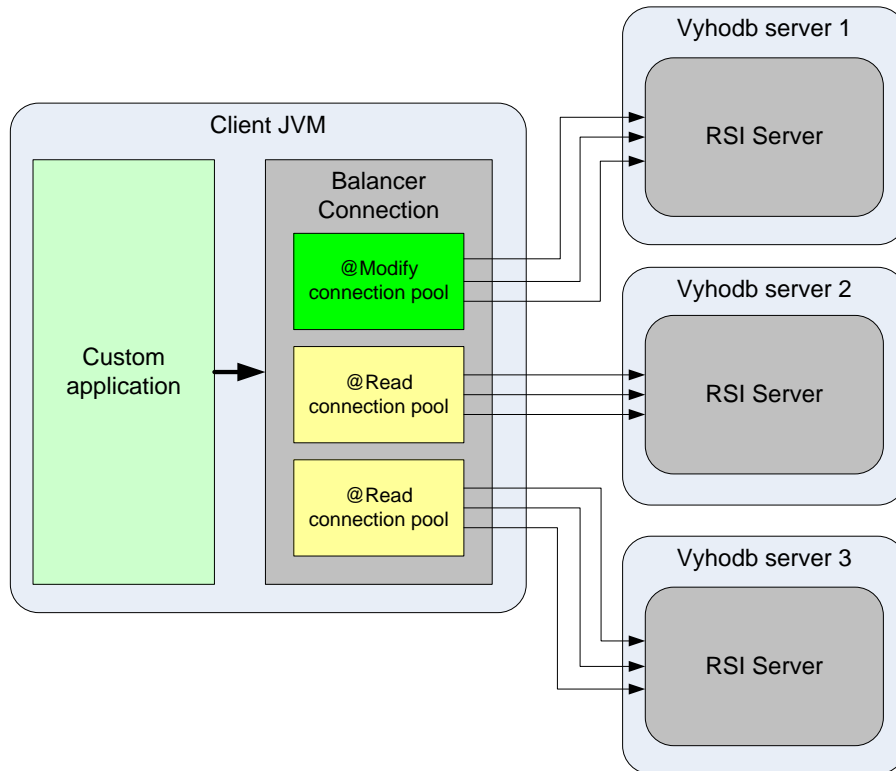
TCP Balancer

Balancer creates several **@Read Connection Pools** and one **@Modify Connection Pool**. Each connection in pools is a TCP connection.

@Read Connection Pools are used for remote invocations of @Read annotated methods (RSI Server opens Read transactions for those methods). @Read Connection Pool is chosen randomly during RSI invocation for particular request.

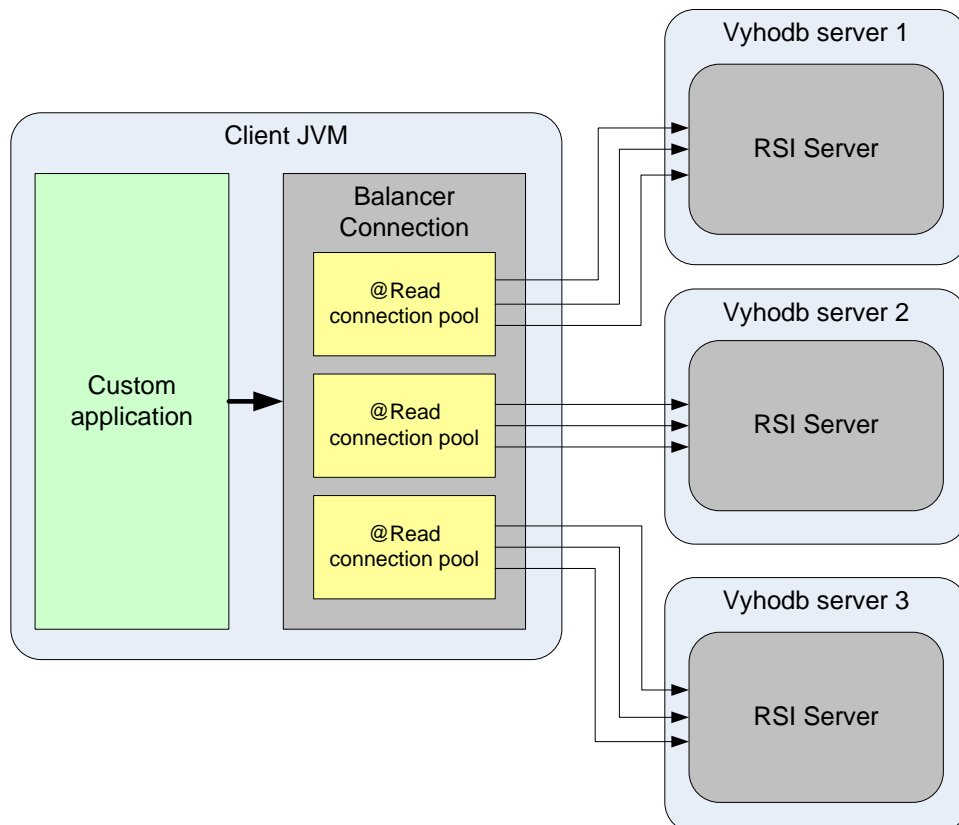
@Modify Connection Pool is used for remote invocations of @Modify annotated methods (RSI Server opens modify transactions for those methods).

Diagram below illustrates how balancer is working:



Balancer is configured by java properties file, which properties are described further in this section.

Balancer can be configured to operate in read-only mode without @Modify Connection Pool. Any attempt to invoke @Modify method would fail in this mode (**com.vyhodb.rsi.RsiClientException**). This mode is illustrated by diagram below:



URL format

String of the following URL format is used to create balancer connection:

balancer:[balancer config file URL]

where **[balancer config file URL]** is an URL, which specifies location of balancer configuration file.

Examples:

balancer:file:c:/vdb-0.9.0/balancer.properties

balancer:http://master.cluster/balancer.properties

Handling with network errors

This section describes how balancer handles network errors (java.io.IOException) during new connection creation or sending RSI request using existed one. Balancer behaves differently for different pool types (@Read and @Modify).

In case of network error (for both @Read, @Modify pools) balancer reloads configuration file and changes configuration if file has been modified. This allows changing configuration on the fly, for instance to change/add new vyhodb server when some server gets unavailable.

@Read invocation

In case of error during invocation of @Read method, balancer performs the following steps:

- 1) Reloads configuration file. If it has been changed then balancer closes all existed pools and creates new ones according to new configuration.
- 2) Waits for a period, which is specified by property **[error.timeout]**.
- 3) Attempts getting new connection and sending RSI request.

Steps described above will be repeated **[error.attempts]** times until successful invocation. If all attempts are failed then balancer throws java.io.IOException to custom application.

Therefore, in case of network error or vyhodb server unavailability, @Read invocations are seamlessly forward to available vyhodb server.

@Modify invocation

In case of error during invocation of @Modify method, balancer takes the following steps:

- 1) Reloads configuration file and applies changes if file has been changed.
- 2) Throws java.io.IOException exception to custom application.

Configuration file

Balancer configuration file is a java properties file:

```

debug = false
refreshConfigTimeout = 600000
read.poolCount = 2

error.attempts = 6
error.timeout = 1000

modify.host = localhost
modify.port = 60000
modify.poolSize = 5

```

```

modify.poolTTL = 3600000

read.0.host = localhost
read.0.port = 45000
read.0.poolSize = 5
read.0.poolTTL = 3600000

read.1.host = localhost

```

In example above we create one @Modify pool and two @Read pools.

Balancer periodically reloads configuration file and, if it has been changed, recreates new connection pools. Property [**refreshConfigTimeout**] specified timeout (in milliseconds) between configuration file reloading.

All configuration properties can be split into three types:

- 1) Common balancer properties
- 2) @Modify connection pool properties
- 3) @Read connection pools properties

Common properties

Property	Description	Default value
debug	Specifies whether or not print debug information on console (System.out, System.err).	false
refreshConfigTimeout	Timeout (in milliseconds) between reloading moments of balancer configuration file.	600000 (10 minutes)
read.poolCount	Count of @Read Connection pools. Property is mandatory and must be specified.	-
error.attempts	Count of attempts for getting new connection from pool after network error (java.io. IOException).	6
error.timeout	Timeout (in milliseconds) between attempts of getting new connection from pool after network error (java.io. IOException).	1000 (1 second)

@Modify Connection Pool properties

If property [**modify.hostname**] isn't specified, then balancer doesn't create @Modify connection pool and works in read-only mode.

Property	Description	Default value
modify.host	RSI Server's host name	
modify.port	RSI Server's port	47777
modify.poolSize	Max pool size (in TCP connections)	5
modify.poolTTL	Connection's time-to-live in pool.	3600000 (1 hour)

@Read Connection Pool properties

Count of @Read Connection Pools is specified by property [**read.poolCount**]. For configuring particular @Read pool, its number (starting from 0) must be specified in property name. In table below, [N] should be superseded by pool's number:

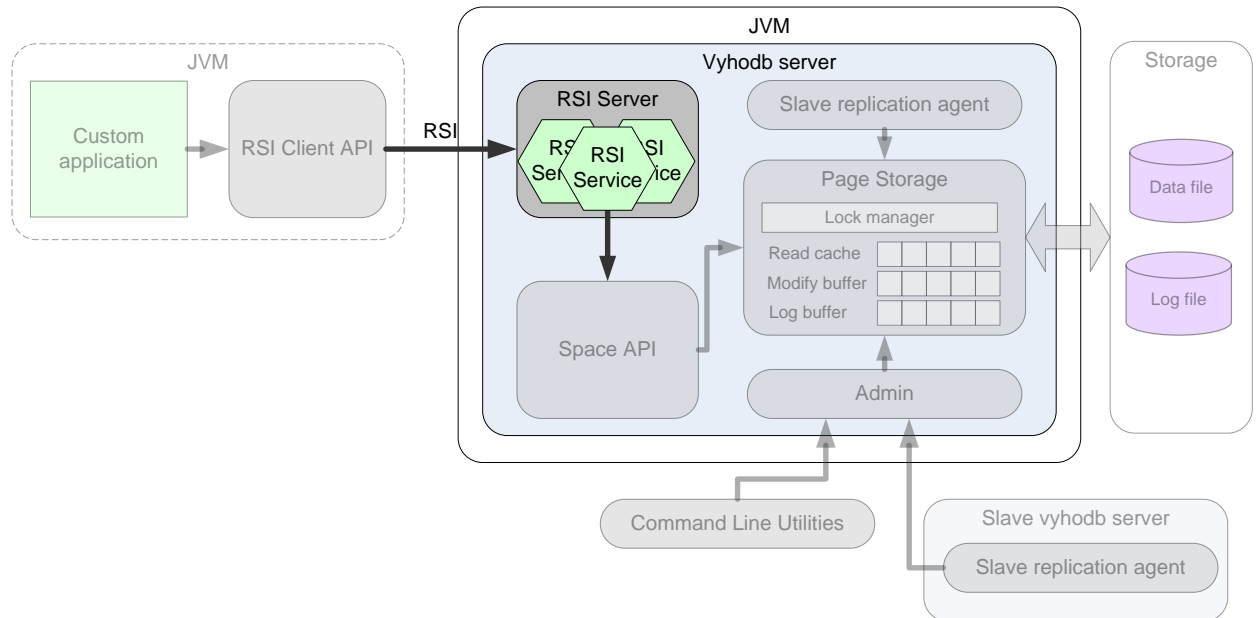
Property	Description	Default value
read.[N].host	RSI Server's host name	
read.[N].port	RSI Server's port	47777
read.[N].poolSize	Max pool size (in TCP connections)	5

read.[N].poolTTL	Connection's time-to-live in pool.	3600000 (1 hour)
------------------	------------------------------------	------------------

RSI Server

RSI Server component processes RSI requests from custom application and is a container of RSI Services.

It is listening for network connections from RSI Client API and creates dedicated Thread for each established connection (Thread-Per-Connection model is used). This thread handles RSI invocation requests, creates RSI Service objects and manages transactions. For more information about RSI Service lifecycle and how RSI Server manages ones, see “Developer Guide”.



RSI Service deployment

In standalone mode, jar files with RSI Service classes should be placed in **services** directory. vyhodb starting scripts (**vdb-start.sh/vdb-start.cmd**) just add all references to jar files in **services** directory into **CLASSPATH** variable.

In local and embedded mode, RSI Service classes can be anywhere; there is only one requirement for them: they should be accessible by class loader. In simple case it means, that they should be included in JVM **classpath**.

RSI Server doesn't support “hot” deployment of RSI Service classes. vyhodb server (or JVM in case of local, embedded modes) should be restarted in order to load new version of classes.

RSI server component uses the same class loader as for vyhodb system classes.

Configuration properties

Following properties are used for configuring RSI Server component:

Property	Description	Default value
rsi.enabled	Switches on/off RSI Server component.	false
rsi.host	RSI Server listening host name.	localhost
rsi.port	RSI Server listening port.	47777
rsi.backlog	RSI Server socket's backlog	100
rsi.cluster.enabled	Switches on/off inconsistency reading safety mechanism. Applicable only for cluster configuration.	false

	For more details see Inconsistent reading section.	
rsi.cluster.probe.attempts	<p>Number of attempts for probing whether required transaction has replicated or not.</p> <p>This property is applicable only when [rsi.cluster.enabled=true]</p> <p>For more details see Inconsistent reading section.</p>	40
rsi.cluster.probe.timeout	<p>Timeout (in milliseconds) between probes whether requires transaction has replicated or not.</p> <p>This property is applicable only when [rsi.cluster.enabled=true]</p> <p>For more details see Inconsistent reading section.</p>	100

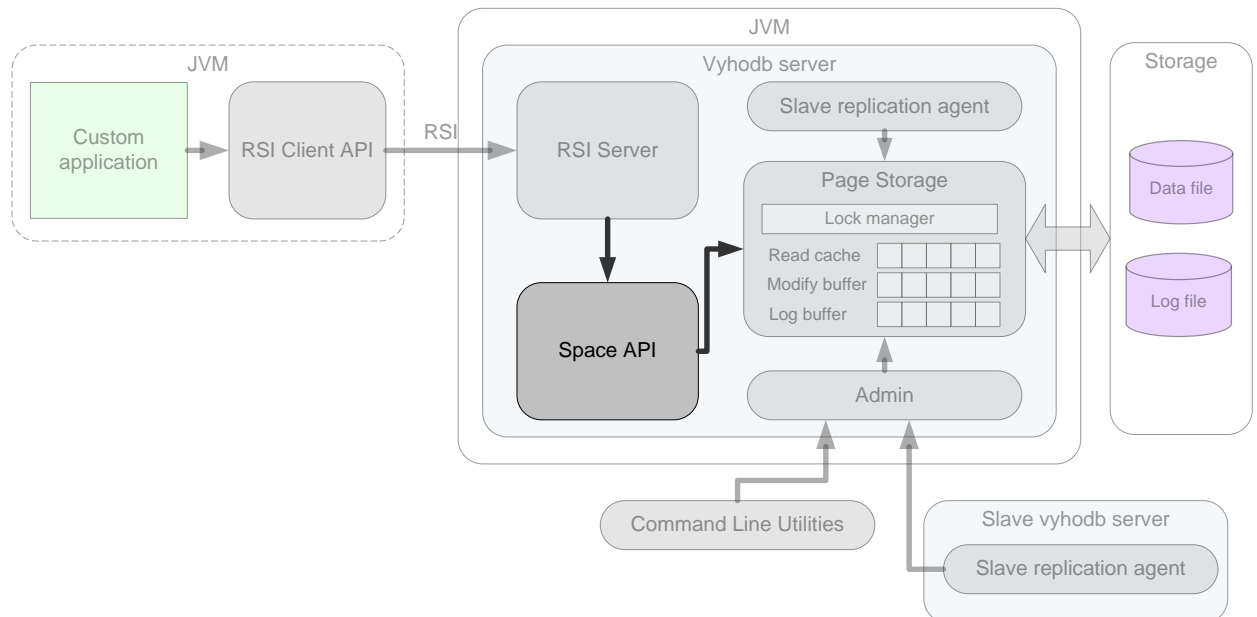
By default, RSI Server is switched on (rsi.enabled = true) in standalone mode and switched off in embedded and local mode.

In local mode, vyhodb server creates so called RSI Server Local component, which doesn't open server socket and therefore doesn't accept network connections from custom applications. In order to start normal **RSI Server** component in addition to RSI Server Local component in local mode, configuration parameter [rsi.enabled= true] must be specified.

Space API

Space API provides API for manipulating vyhodb data (records, fields, links). It also transforms read/write operations on vyhodb data into storage page read/write requests.

For more information about vyhodb data model and Space API see “Developer Guide”.



Maximal size of vyhodb record is specified by property:

Configuration property	Description	Default value
space.record.maxRecordSize	Record's max size (in pages).	16384 (16 mb)

Modify Records Cache

In case of Modify transaction, Space API creates cache of read/modified records which is bound to Modify transaction. In case of Read transaction, record cache isn't created and record objects are read and created each time when custom application (or RSI Server) retrieves reference to it.

When cache of records gets full, Space API flushes all changed records and clears cache, so it can be used again. Optimal size for Modify Record Cache is an average count of read/modified records per Modify transaction. Usually default value suits for most Modify transactions.

Maximal size of modify record cache is specified by property:

Configuration property	Description	Default value
space.record.modifyCacheSize	Record cache's max size in modify transactions. Measure unit – count of vyhodb record.	300

Dictionary

By default, names of fields, links, and indexes are stored inside each vyhodb record. This leads to increasing amount of disk space, occupied by vyhodb storage.

In order for increasing performance and decreasing occupied space, application developer can create so called dictionary file, which maps string constants to integer codes. Once dictionary file is created and configured (see below), vyhodb starts writing integer codes in stored records instead of strings.

Dictionary file is a java properties file. Example of this file is presented below:

```
Name = 1
Price = 2
product2root = 3
order2root = 4
```

Path to dictionary file is configured by the following property:

Configuration property	Description	Default value
space.dictionary	Path to directory file	

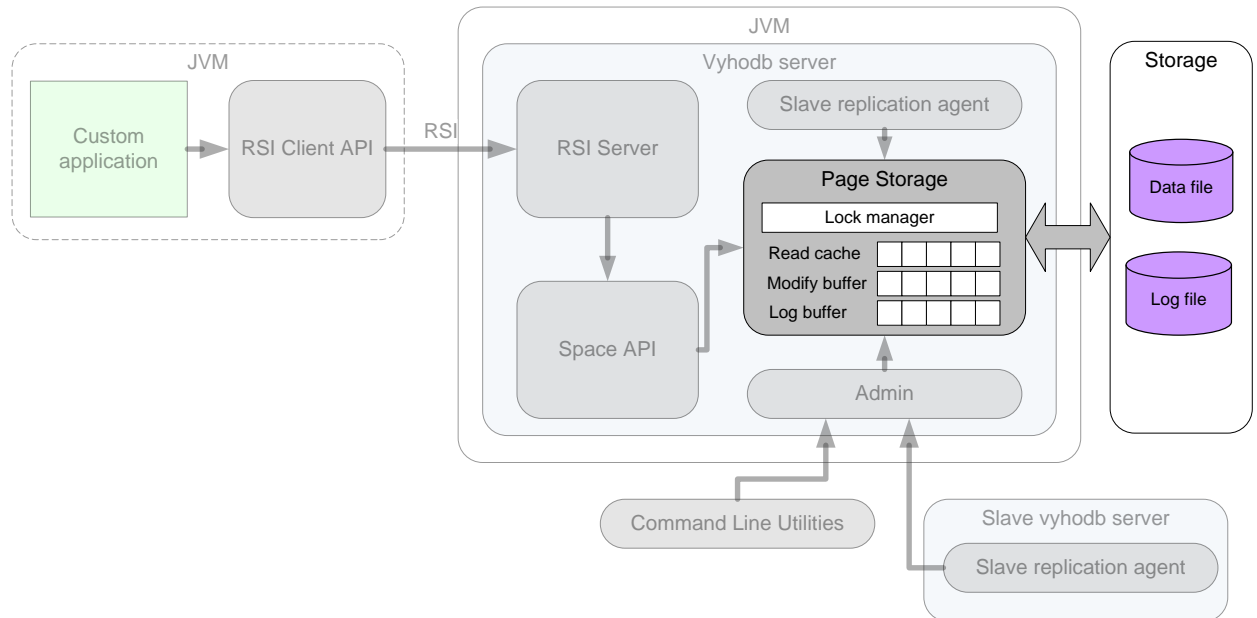
Vyhodb reads dictionary file at startup time. To actualize changes in dictionary file, vyhodb server should be restarted.

If vyhodb can't find string for integer code during record reading, than current transaction will be rolled back and **com.vyhodb.server.TransactionRolledbackException** will be thrown.

Page Storage

Page Storage is a component, which responsible for:

- 1) Reading/writing data file pages
- 2) Caching data file pages
- 3) Implementing transactions on page level
- 4) Storage recovery after system fault



Storage

Vyhodb data and system info are stored in so called storage. Storage consists of two files:

- 1) Data file
- 2) Log file (transaction journal file)

Despite of the fact, that data and log files can be placed in different directories and even in different disks, file systems, etc., they are logically connected to each other (have the same logId, see further for more details). Because of this logical connection, vyhodb prevents starting itself with files from different storages.

By default, both files are located in **storage** directory. Their locations can be changed by the following properties:

Configuration property	Description	Default value
storage.data	Path to data file.	
storage.log	Path to log file.	

New storage is created by **vdb-new** command line utility (see [Command line utilities](#)).

Storage can be opened by only one running vyhodb server at the time.

Data file

All vyhodb data (records, fields, links and indexes) are stored in data file. Data file consists of pages. Each page size is 1024 bytes. First page is reserved and used for storing system info. Page numeration

starts from 0 (for second page). Maximal theoretical data file size ~ 8191 Pb, but in fact it is limited by file system.

Page storage component automatically increases size of data file when new data is added. Space occupied by deleted records isn't recycled, that is why data file can't be truncated – it always grows.

Page storage opens several file descriptors for read pages from data file. It decreases concurrency between threads for file descriptors. Count of opened file descriptors is specified by property:

Configuration property	Description	Default value
storage.readDescriptorCount	Count of file descriptors which are opened for reading data file.	20

Log file

Log file is a write-ahead transaction journal. It contains data file's pages, modified by transactions.

Main purpose of log file – is restoring storage (data file and log file) in consistent state after system failure. At start time, Page Storage component checks whether storage was successfully closed at last time (by analyzing log info structure). If it didn't (because of system fault or vyhodb process destroying), then vyhodb starts recovery procedure. During recovery procedure, Page Storage tries to read transactions from log file and apply them to the data file.

Log file is also used to create consistent backup file for running vyhodb server (see **vdb-backup-remote** utility in **Command line utilities** section for more details).

First page of log file is used for storing **log info** structure. Page numeration starts from

-9223372036854775807. Maximal log file page number is 9223372036854775807.

During storage restoration from backup file (see **vdb-restore** utility in **Command line utilities** section) page counter of log file is set to its minimal value (-9223372036854775807). So backup/restore cycle can be used to reset log file page count when it reaches maximum value.

Log file automatically grows in size.

Truncate log file

In contrast to the data file, log file can be truncated in size. The following command line utilities are used for this: **vdb-shrink** (for closed storage), **vdb-shrink-remote** (for opened storage by running vyhodb server).

If storage of truncated log file participates in replication as MASTER storage, then network address list of running SLAVE servers should be passed to utilities **vdb-shrink/vdb-shrink-remote**. This is done by using **-slavelist** option. This allows utility to understand, by asking slave servers, which transactions aren't replicated yet and should be left in MASTER log file.

Otherwise, after log file truncation, replication would be destroyed and SLAVE servers would fall out of synchronization with "Out of Sync" error.

Log info

Log info is a system info, which describes log file and the whole storage. It is stored in first page of log file.

Command line utilities **vdb-info**, **vdb-info-remote** are used to read **log info** from closed or open storage. Listing below shows command line utility output:

```
Log info
-----
    logId: 26f6d505-139d-4a34-afb1-888f1a68b31a
    version: 0
-----
    slave: false
masterLogId: 00000000-0000-0000-0000-000000000000
-----
successfulStop: true
-----
```

Log info structure consists of the following fields:

Field	Type	Description
logId	UUID	Storage and log file identifier.
version	int	Version of storage format.
slave	boolean	SLAVE storage flag (see Replication section).
masterLogId	UUID	MASTER storage id. Filled on SLAVE storages only (see Replication section).
successfulStop	boolean	Successful storage close flag.

Durability

By default, `fsync()` system call on log file is invoked only at **checkout** (described in [Modify Buffer](#) section). This could lead to loss of committed transactions in case of system failure. When transaction durability is important and `fsync()` call should be done after each modify transaction, the following property should be set to **true**:

Configuration property	Description	Default value
storage.durable	Specifies whether vyhodb should call fsync() on log file after modify transaction commit. By default this property is false and fsync() is called on log file only at checkpoints.	false

Note, that majority of database management systems don't call `fsync()` on transaction journal after each transaction commit, by default. This is because `fsync()` call is time consuming and decrease overall database performance.

Caches and Buffers

Page Storage creates following caches and buffers for storing pages:

- 1) Read Cache
- 2) Log Buffer
- 3) Modify Buffer

All caches and buffers are implemented as Direct ByteBuffers and are allocated at Page Storage start time. Memory which is used by caches and buffers is out of Java Heap and isn't a subject of garbage collection.

By default, JVM limits memory which could be allocated for Direct Buffers. It might lead to OutOfMemoryError at vyhodb start time. To overcome this problem and increase JVM limit, JVM should be started with the following parameter:

-XX:MaxDirectMemorySize=size[g|G|m|M|k|K]

In case of standalone mode, this parameter should be specified in **vdb-start.sh/vdb-start.cmd** scripts.

Read Cache

Read cache stores clean, unmodified data file pages.

When page is modified it is moved from Read Cache to Modify Buffer where it waits checkpoint, during which all dirty pages are written to data file.

Read Cache size is the most important criteria of vyhodb performance: **the larger Read Cache, the better vyhodb performance!**

Configuration property	Description	Default value
storage.cacheSize	Read cache size (in pages).	50000

Log Buffer

Log Buffer is used for buffering pages, which have been changed by Modify transaction, before writing them into log file.

Optimal log buffer size should be equal to average number of pages modified by one transaction. Default size (~25 mb) is quite enough for most modify transactions.

Log Buffer's size is configured by the following property:

Configuration property	Description	Default value
storage.logBufferSize	Log buffer size (in pages).	25000

Modify Buffer

Modified pages are not written to the data file at once. After successful completion of modify transaction, all dirty pages are put into Modify Buffer and are waiting there till **checkpoint** operation.

When Modify Buffer gets full, Page Storage starts checkpoint operation, which in turns performs the following steps:

- 1) Fixes log file by invoking **fsync()** system call.
- 2) Writes all dirty pages from Modify Buffer into data file and clears Modify Buffer
- 3) Fixes data file by invoking **fsync()** system call.

Once checkpoint operation successfully completed, it is guarantee that all dirty pages changed since previous checkpoint are successfully written in data file, and that data file and log file are in consistent state. At vyhodb shutdown time, Page Storage automatically performs **checkpoint** operation.

The larger size of Modify Buffer the more period of time between **checkpoint** operations and therefore the more pages are written to data file at checkpoint time (which isn't good from performance perspective).

Modify Buffer's size is specified by the following parameter:

Configuration property	Description	Default value
storage.modifyBufferSize	Modify buffer size (in pages).	25000

Lock Manager

Lock Manager is responsible for locks between transactions.

Lock granularity is the whole vyhodb storage. There are differences in lock mechanisms between Read and Modify transactions: at the same time vyhodb server can handle many Read transactions and only one Modify transaction. For more details about transaction and ISOLATION LEVEL see "Developer Guide".

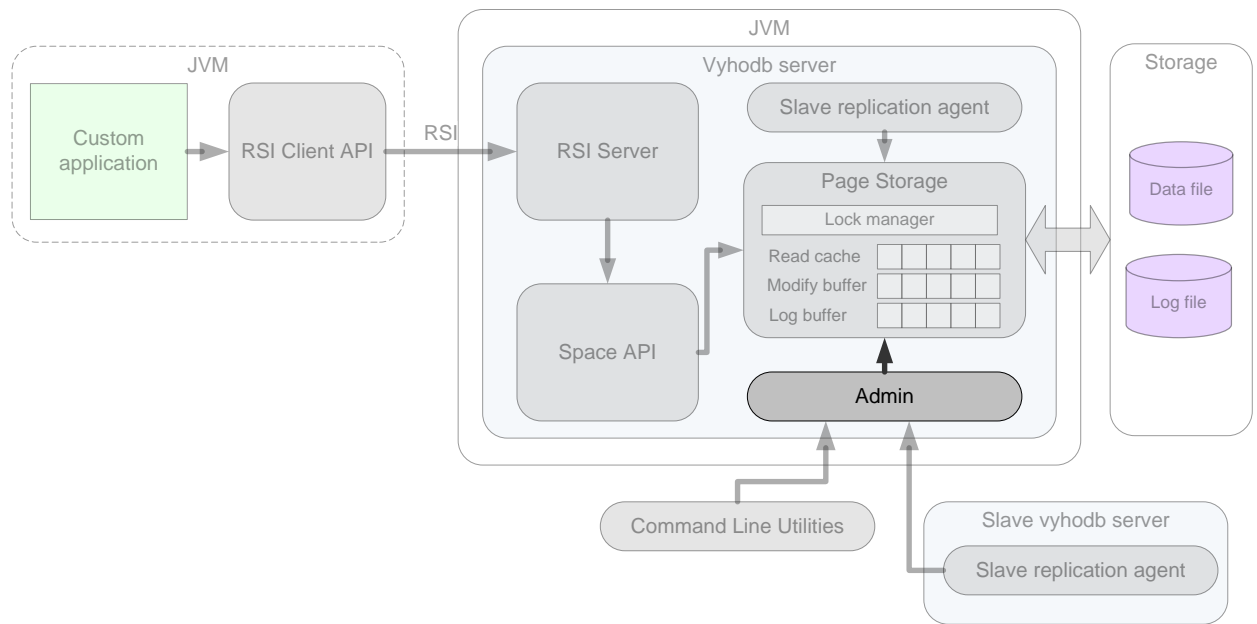
At commit time, Modify transaction stops opening new Read transactions and waits completion of running Read transactions. Once Read transactions are completed, Page Storage perform Modify transaction commit (with moving pages to Modify Buffer and checkpoint if needed). After that new Read and Modify transactions can be open.

Each Thread acquires lock at the beginning of new transaction (Read, Modify) and at commit phase (Modify). There is a timeout for waiting lock acquisition, which is configured by property below. When timeout expires, transaction is rolled back and exception **com.vyhodb.server.TransactionRolledbackException** is thrown.

Configuration property	Description	Default value
storage.lock.timeout	Lock acquisition timeout (in seconds)	120 (2 minutes)

Admin

Admin component is responsible for running facilitating administrative tasks on running vyhodb server.



Component accepts network connections from remote command line utilities and Slave Replication Agents. When connection established, Admin component receives and execute commands.

Admin component, as well as RSI Server, uses Thread-Per-Connection model. For copying pages (which is essential part of backup and SRA functionalities), each Thread allocates buffer, which size is specified by property `[admin.connectionBufferSize]`.

Table below shows configuration properties which relate to Admin component:

Configuration property	Description	Default value
<code>admin.enabled</code>	Switches on/off Admin component.	false
<code>admin.host</code>	Admin listening host name.	localhost
<code>admin.port</code>	Admin listening port.	46666
<code>admin.backlog</code>	Admin socket's backlog.	20
<code>admin.connectionBufferSize</code>	Admin's page buffer size (in pages).	64

Slave Replication Agent (SRA)

Slave Replication Agent component is described in section dedicated to Replication. See **Slave Replication Agent (SRA)**.

Command line utilities

Command line utilities (CLU) are used for administration tasks like new storage creation, storage's backup creation, storage restoring, log file truncation, etc.

Command line utilities are located in **bin-cmd** directory for Windows platform and in **bin-sh** directory for Linux. Table below describes CLU:

CLU	Description
vdb-new	<p>Creates new storage (data and log files).</p> <pre>vdb-new -log=c:\storage\vyhodb.log -data=c:\storage\vyhodb.data</pre> <p>-log Path to the log file -data Path to data file</p> <p>Both parameters (log, data) are mandatory.</p>
vdb-backup	<p>Creates backup file for storage.</p> <p>Storage must be closed (no running vyhodb server).</p> <pre>vdb-backup -log=vyhodb.log -data=vyhodb.data -backup=vyhodb.bak -step=20</pre> <p>-log path to log file -data path to data file -backup path to backup file -step progress step in percent</p> <p>-log, -data, -backup parameters are mandatory.</p>
vdb-backup-remote	<p>Connects to remotely running vyhodb server and creates backup file for its storage.</p> <p>Backup file is created locally. Admin component should be active on vyhodb server.</p> <pre>vdb-backup-remote -host=localhost -port=46666 -backup=vyhodb.bak -step=20</pre> <p>-host vyhodb host name -port Admin component's listening port on running vyhodb server -backup path to backup file -step progress step in percent</p> <p>-host, -port, -backup parameters are mandatory.</p>
vdb-restore	<p>Restores vyhodb storage from backup file.</p> <p>This utility is also used for SLAVE storage creation (see Replication section).</p> <pre>vdb-restore -backup=vyhodb.bak -log=vyhodb.log -data=vyhodb.data -step=20</pre> <pre>vdb-restore -backup=vyhodb.bak -log=slave.log -data=slave.data -slave</pre> <p>-backup path to existed backup file -log path to new log file</p>

	<p>-data path to new data file</p> <p>-slave creates SLAVE storage when this option is presented</p> <p>-step Progress step in percent</p> <p>-log, -data, -backup parameters are mandatory.</p>
vdb-clearslave	<p>Turns SLAVE storage into ordinary storage.</p> <p>Storage must be closed (no running vyhodb server). After successful utility completion, specified storage can't participate in replication as SLAVE storage.</p> <p>vdb-clearslave -log=slave.log</p> <p>-log path to log file of SLAVE storage</p> <p>-log parameter is mandatory.</p>
vdb-clearslave-remote	<p>Turns SLAVE storage of running vyhodb server into ordinary one.</p> <p>Slave Replication Agent is stopped (if it is running) and server starts processing Modify transactions.</p> <p>vdb-clearslave-remote -host=localhost -port=46666</p> <p>-host vyhodb host name</p> <p>-port Admin component's listening port on running vyhodb server</p> <p>Both parameters (-host, -port) are mandatory.</p>
vdb-shrink	<p>Truncates log file.</p> <p>Storage must be closed (no running vyhodb server).</p> <p>Attention! If truncated storage participates in replication as MASTER storage, than option [-slavelist] must be used. Otherwise, replication will be destroyed and SLAVE servers will fall out of synchronization with "Out of Sync" error.</p> <p>vdb-shrink -log=vyhodb.log -slavelist=slaves.txt</p> <p>-log path to log file</p> <p>-slavelist path to a file with network addresses of running SLAVE servers. See Slave list file.</p> <p>Parameter -log is mandatory.</p> <p>If option -slavelist is specified then utility asks running SLAVE servers and determinates optimal position for log truncation, so that not replicated transactions couldn't be corrupted.</p> <p>For more information about "Out of Sync" error see Out of Sync Error section.</p>
vdb-shrink-remote	<p>Truncates log file on running vyhodb server.</p> <p>Attention! If truncated storage participates in replication as MASTER storage, than option [-slavelist] must be used. Otherwise, replication will be destroyed and SLAVE servers will fall out of synchronization with "Out of Sync" error.</p>

	<p>vdb-shrink-remote -host=localhost -port=46666 -slavelist=slaves.txt</p> <p>-host vyhodb host name -port Admin component's listening port on running vyhodb server -slavelist path to a file with network addresses of running SLAVE servers. See Slave list file.</p> <p>Parameters -log and -port are mandatory.</p> <p>If option -slavelist is specified then utility asks running SLAVE servers and determinates optimal position for log truncation, so that not replicated transactions couldn't be corrupted. For more information about "Out of Sync" error see Out of Sync Error section.</p>
vdb-info	<p>Prints Log Info of closed storage.</p> <p>vdb-info -log=vyhodb.log</p> <p>-log path to log file</p> <p>-log parameter is mandatory.</p>
vdb-info-remote	<p>Prints Log Info of opened storage (by running vyhodb server).</p> <p>vdb-info-remote -host=localhost -port=46666</p> <p>-host vyhodb host name -port Admin component's listening port on running vyhodb server</p> <p>Both parameters (-host, -port) are mandatory.</p>
vdb-close-remote	<p>Stops running vyhodb server.</p> <p>If vyhodb server is running in Standalone mode, then stops JVM as well.</p> <p>vdb-close-remote -host=localhost -port=46666</p> <p>-host vyhodb host name -port Admin component's listening port on running vyhodb server -timeout timeout before server stopping. Specified in seconds. Deafult value: 1 second.</p> <p>Both parameters (-host, -port) are mandatory.</p>

Slave list file

Slave list file contains list of network addresses of running SLAVE servers.

This file is a text file where each line specifies particular network address of Admin component (must be activated) in SLAVE server. Line format:

[host name/IP address]:[admin port]

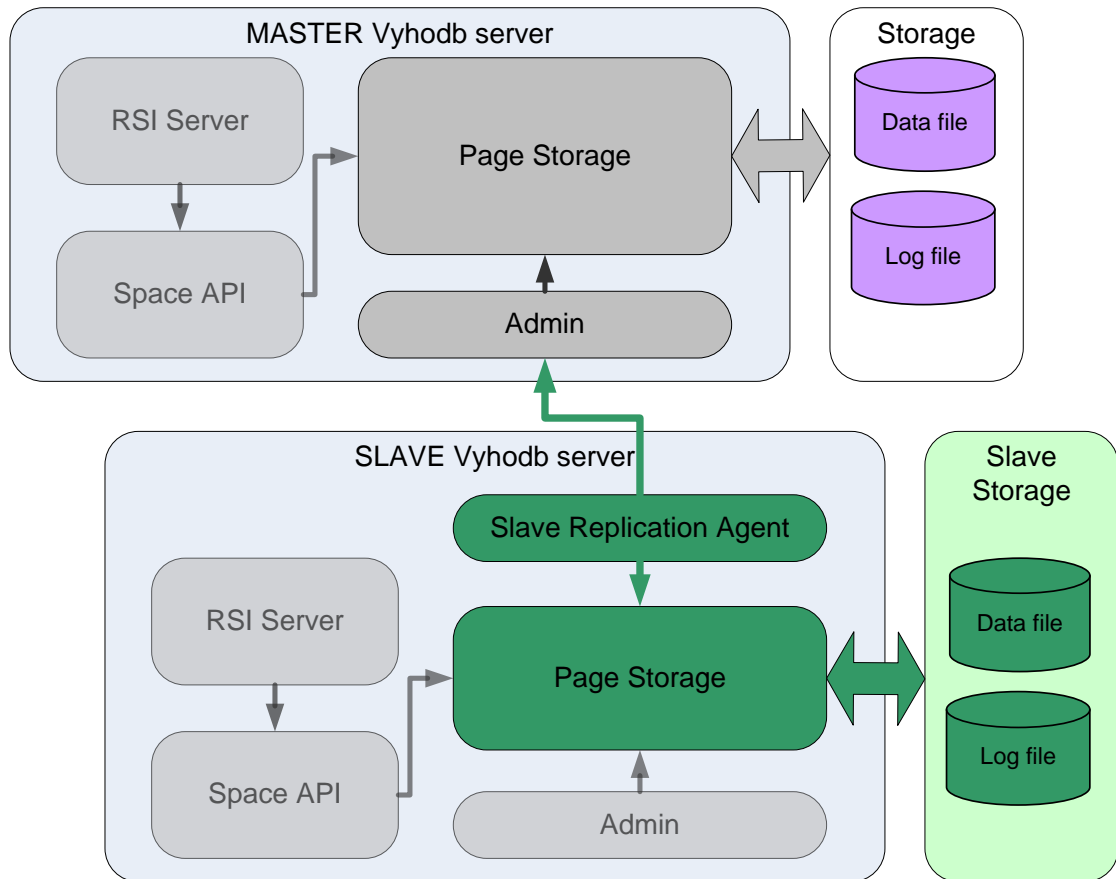
Slave list file example:

```
slave1.cluster.com:46666
slave2.cluster.com:46666
10.0.2.15:43333
```

Replication

One way replication of modify transaction can be configured between two vyhodb servers. Source vyhodb server plays MASTER role, while destination vyhodb server plays SLAVE role.

Replication process is asynchronous. Modify transactions are completed in usual mode on MASTER server and don't wait their replication into SLAVE servers.



Replication doesn't involve RSI Service classes (jar files) and dictionary files (see **Dictionary**).

Administrator should manually support identical versions of RSI Service's jar files and dictionary files on MASTER and SLAVE servers.

There is no need to keep RSI Service's jar archives identical if MASTER server isn't used in RSI communications (its RSI Server component is switched off). The same is true for dictionary files if MASTER server is used as intermediate server in cascade replication and isn't used by custom applications (by Server API, Space API or RSI).

MASTER server

Any running vyhodb server (in any running modes) with active **Admin** component can participate in replication as MASTER server.

MASTER server has no information about its SLAVE servers and doesn't depend on them.

No configuration is required on MASTER server for setting up replication. All configuring is taking place on SLAVE servers.

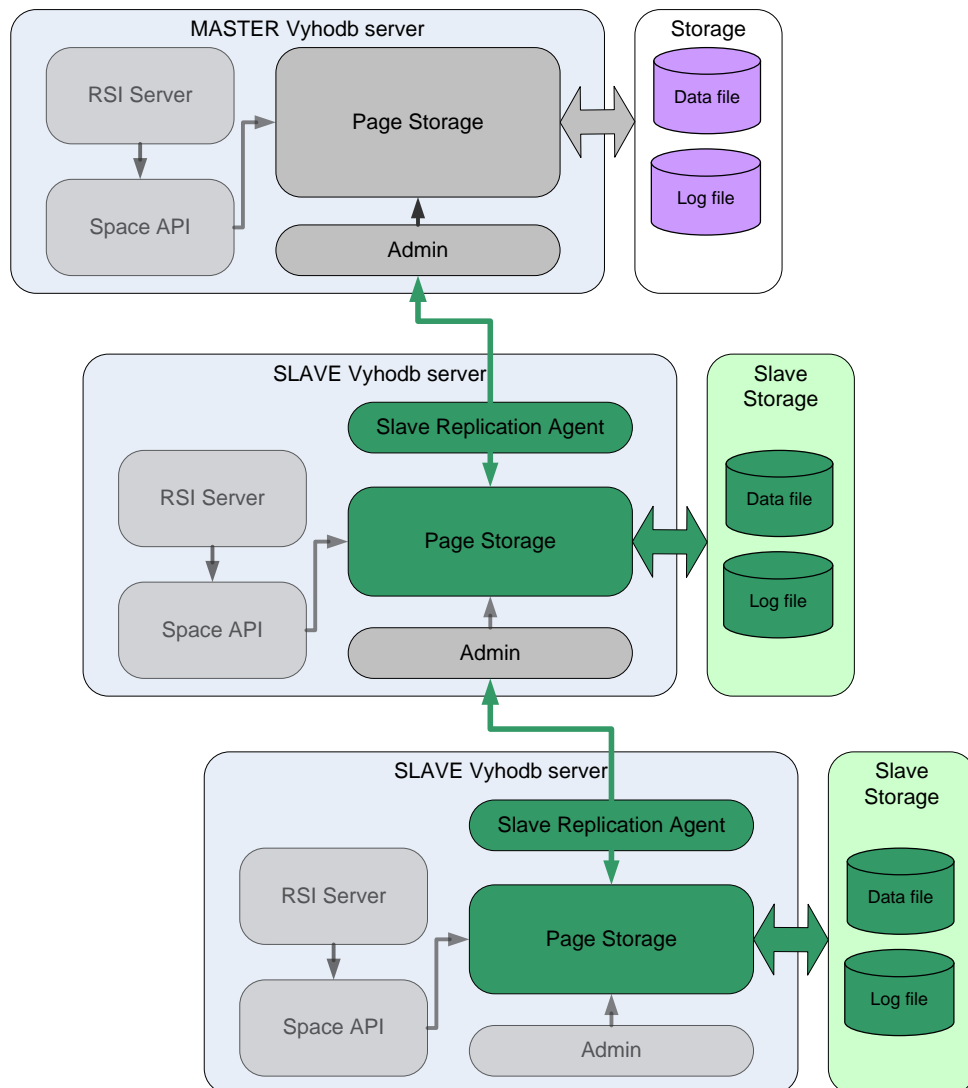
SLAVE server

SLAVE server has the following distinctions:

- 1) It operates over SLAVE storage
- 2) It has running “Slave Replication Agent” (SRA) component.

SLAVE server can be started in any mode (Standalone, Embedded, Local).

SLAVE server can play MASTER role for other vyhodb servers, creating so called cascade replication configuration:



SLAVE server configuring

To create SLAVE server the following steps should be done:

- 1) Create Slave Storage
- 2) Configure SLAVE vyhodb server, namely:
 - a. Configure SRA component
 - b. Configure Slave Storage

Next sections describe how to create Slave Storage (**Slave Storage**) and configure SRA component (**Slave Replication Agent (SRA)**).

Slave storage configuring means specifying paths to slave storage files (data and log files).

Below is an example of **vdb.properties** file for SLAVE server:

```
storage.log = storage/slave.log
storage.data = storage/slave.data

# Slave enabled
slave.enabled = true

# Master host name
slave.master.host = master1.company.com

# Master admin port
slave.master.port = 46666

# Slave running mode
#
# Possible values:
# 1) realtime
# 2) cron
#
slave.mode = realtime

# Master new updates check timeout. In milliseconds.
# Only for [realtime] running mode.
slave.checkTimeout = 500
```

Slave Storage

Slave storage allows only Read transactions and invoking only @Read methods of RSI Services. Any attempt to start Modify transaction would fail with **com.vyhodb.server.TransactionRolledbackException** exception. Only SRA component has permission to make changes in Slave Storage by writing replicated transactions.

Slave Storage is created by restoring from backup file of MASTER storage. Utility **vdb-restore** with **–slave** option is used for this (see [Command line utilities](#)). Utility fills the following fields of **Log info** structure:

Field	Type	Description
slave	boolean	Sets to true .
masterLogId	UUID	Sets to logId of MASTER storage.

Fields above are used in “Out of sync” check which is done every time when SRA establishes connection to MASTER server.

Slave Storage can be turned into ordinary storage by executing **vdb-remote-clearslave** or **vdb-clearslave** utility. Fields (**slave**, **masterLogId**) are cleared and current storage can’t participate in replication as SLAVE storage any more.

Slave Replication Agent (SRA)

The purpose of SRA component is retrieving new transactions from MASTER server and applying them into SLAVE storage. In order to do this, SRA establishes network connection with Admin component on MASTER server.

SRA running modes and configuring

SRA can be configured for running in one of the following modes:

- 1) realtime
- 2) cron

Configuration properties in the table below are common for both modes:

Configuration property	Description	Default value
slave.enabled	Switches on/off SRA component.	false
slave.master.host	MASTER host name	localhost
slave.master.port	Admin component port's number on MASTER server.	46666
slave.mode	SRA mode. Two modes are supported: realtime, cron.	realtime

In realtime mode, SRA establishes long-running network connection to Admin component on MASTER server and uses it for checking new transactions and obtain them.

Properties which are applicable for realtime mode:

Configuration property	Description	Default value
slave.ttl	SRA connection time-to-live (in milliseconds). Applicable only for realtime mode.	86400000 (24 hours)
slave.checkTimeout	Timeout between new transaction check (in milliseconds). Applicable only for realtime mode.	1000 (1 second)

In cron mode, SRA establishes connection to MASTER server at periods, specified by cron expression. SRA retrieves all new transactions at those periods and closes connection till next cron event.

Configuration property	Description	Default value
slave.cron	Cron expression for synchronization sessions. Applicable only for cron mode.	* * * * *

MASTER server unavailability

In case when SRA connection to MASTER server is broken or MASTER server gets unavailable, SRA keeps trying to establish connection to MASTER server.

Timeout between attempts for **realtime** mode is specified by property [**slave.checkTimeout**].

In case of **cron** mode, [**slave.cron**] expression specifies periods when attempt to establish new connection is made (this is usual behavior of cron even without any failures).

Out of Sync Error

After establishing new connection to Admin component on MASTER server, SRA checks for “Out of Sync” error (in both modes). If out of synchronization took place between MASTER and SLAVE storages, then SRA stops working with “Out of Sync” Error and closes SLAVE server.

If configuration property [**exitVmOnCriticalException**] is set to **true**, then SRA stops its JVM as well.

Possible causes of “Out of Sync” error:

- 1) Different field values: masterLogId (on SLAVE storage) and logId (on MASTER storage). See [Log info](#).
- 2) MASTER's log file was truncated without considering SLAVE server. See [Truncate log file](#).

- 3) MASTER server lost transactions which had already been replicated to SLAVE server. This situation is possible after MASTER server failure and some transaction don't reach log file but replicated to SLAVE server or when log recovery can't read corrupted transactions from log file after failure. Both cases can be prevented by enable durability (see **Durability**), which, unfortunately, affects performance.
- 4) Retrieved transaction is corrupted.

There is no way to fix "Out of Sync" error except first case (just configure the right MASTER Server).

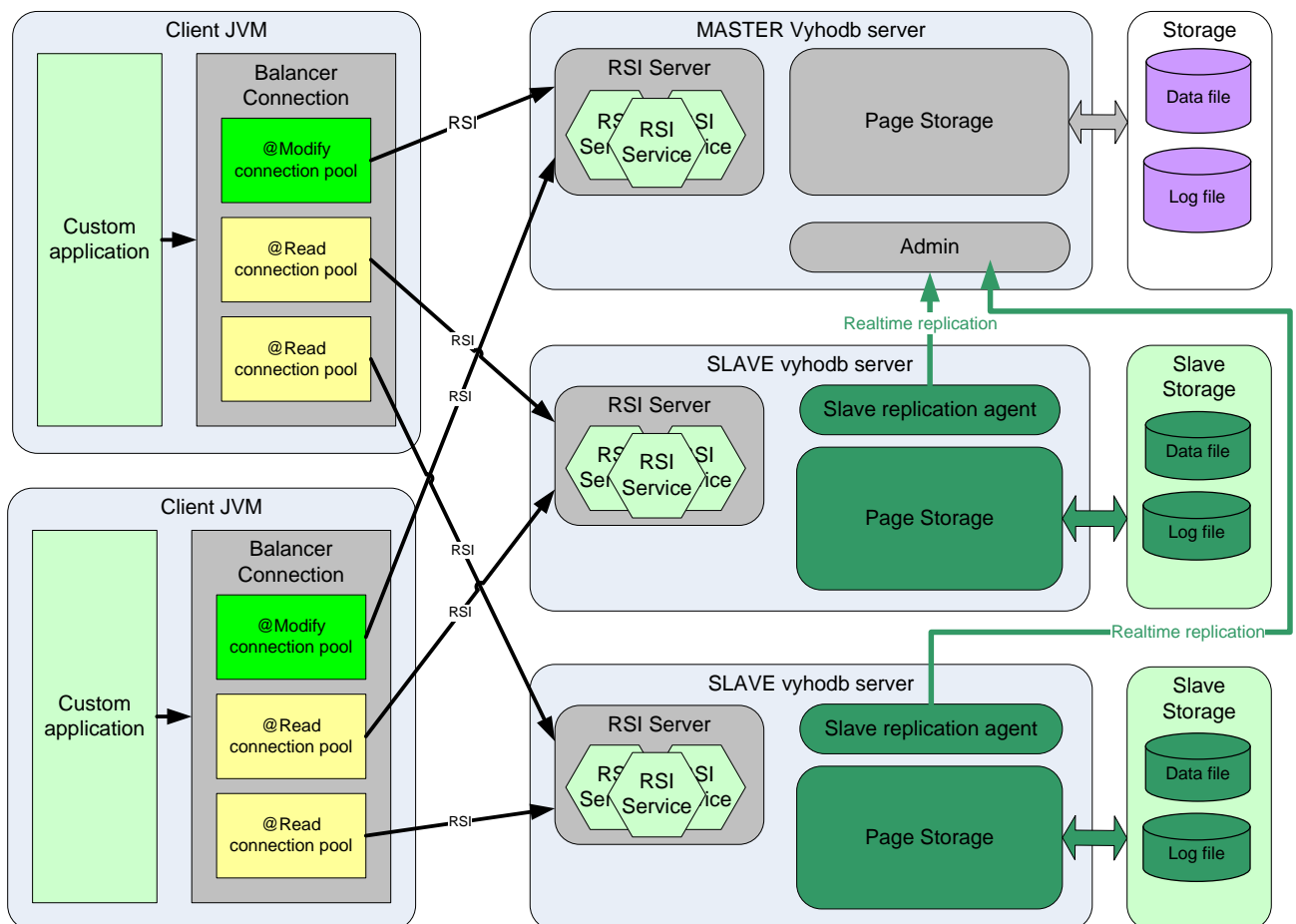
System administrator should make a decision about which server use as a MASTER: existed or one of SLAVE servers. And recreate all other SLAVE servers (if they can't synchronize with new MASTER).

Clustering

Several vyhodb servers can be configured to work as cluster for the purpose of increasing performance by balancing RSI @Read method invocations between cluster nodes.

RSI @Modify invocations are not balanced and send to dedicated server (which is plays MASTER replication role for other servers in cluster).

Clustering is transparent for custom application and doesn't require changes in code. Example of clustering configuration is shown below:



Clustering is based on two technologies:

- 1) **Replication**
- 2) **TCP Balancer**

There is a replication between servers with realtime mode. MASTER server processes RSI @Modify requests and modifies vyhodb data. Changed data are replicated to SLAVE servers which process RSI @Read requests.

Custom application must be implemented using RSI technology (see "Developer Guide", **RSI Client API**, **RSI Server** for more details). In this case, cluster is just a single vyhodb server from application's point of view. Custom application establishes RSI connection (which must be of TCP Balancer type). This RSI connection sends all @Modify requests to MASTER server whereas @Read requests are distributed between SLAVE servers.

Note, that replication supports neither synchronization of RSI Service classes (jar files) nor dictionary files. System administrator should manually keep the same versions of RSI Service jars and dictionary files on every cluster node. See sections **RSI Server** and **Dictionary** for more information about RSI Service classes and dictionary file.

Cluster configuring

The following steps should be done:

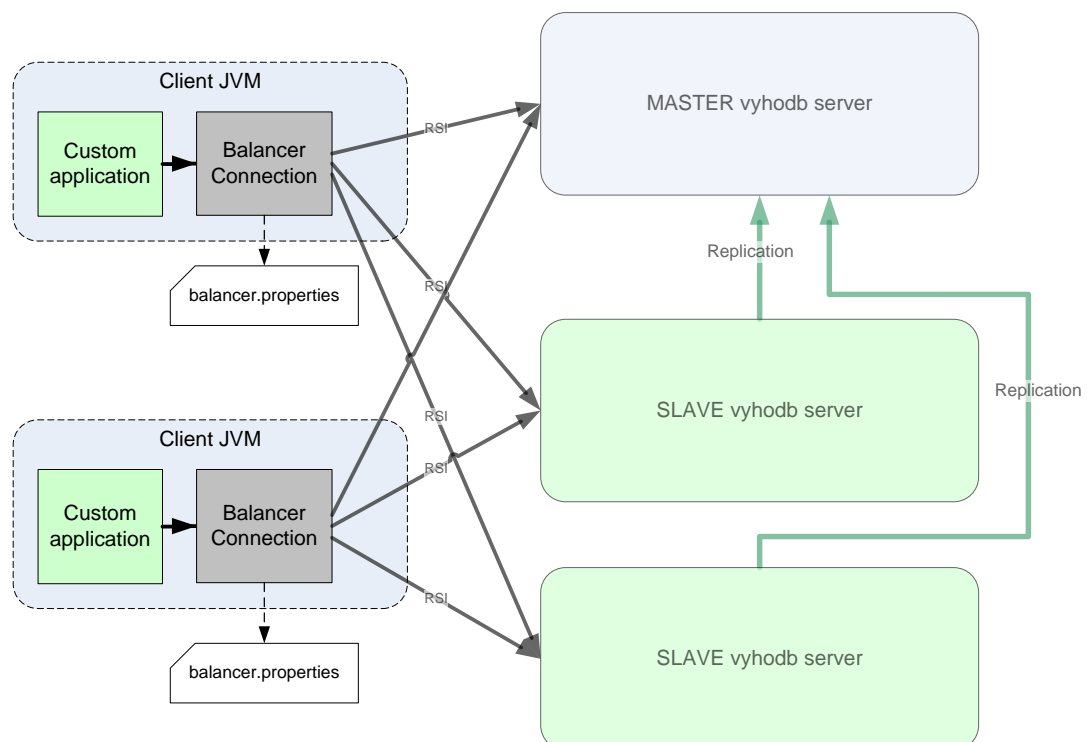
- 1) Configure MASTER and SLAVE servers and realtime replication between them.
- 2) Set configuration property [**`rsi.cluster.enabled=true`**] on each server (see **Inconsistent reading** section below).
- 3) Create balancer's configuration file and put it in centralized storage (dedicated HTTP/FTP server) or locally (where custom applications are running).
- 4) Change connection string of custom application to balancer URL.

After that, cluster servers and clients can be started and running.

Location of balancer configuration file

RSI Client API uses connection string, passed at connection creation time to determine location of balancer configuration file.

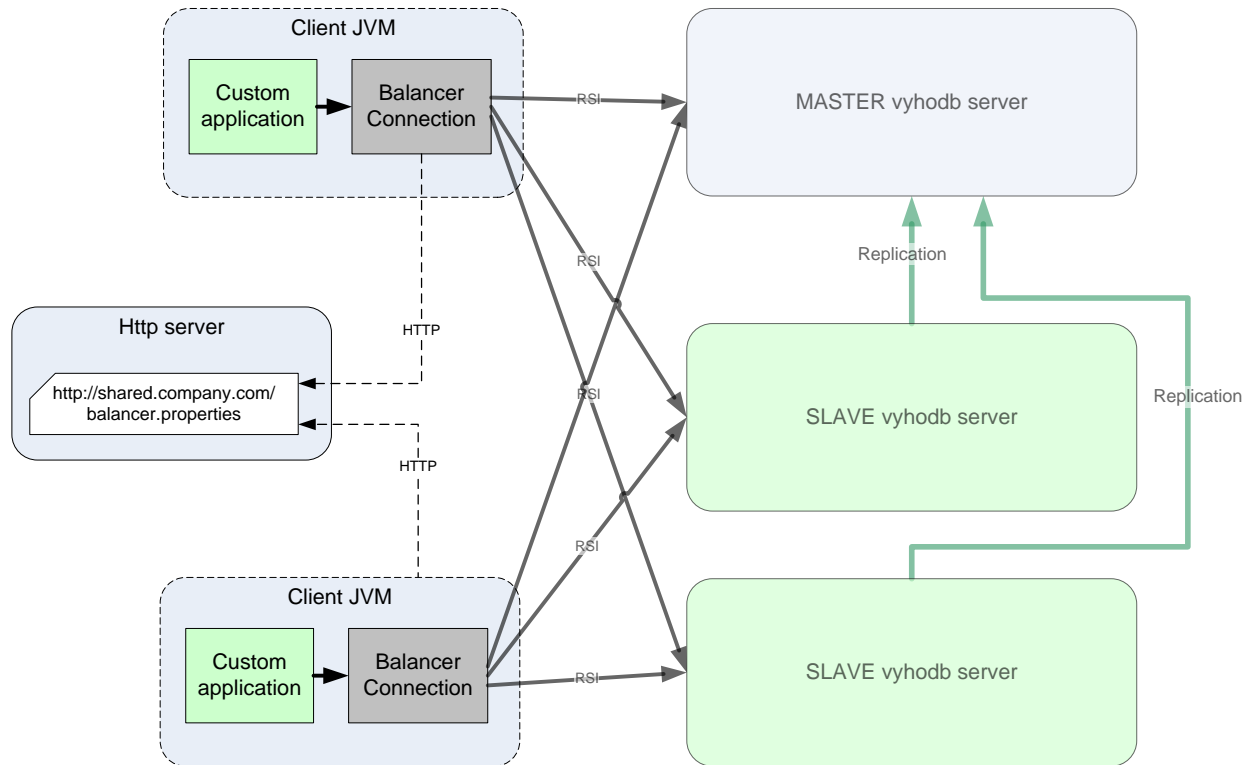
Balancer's configuration file can be placed locally on the nodes where custom applications are running. URL of the following format is used in this case: **`"balancer:file:[path to config file]"`**. Diagram below shows such configuration:



Balancer configuration file can also be placed on dedicated server (http, ftp), which is accessible from every client node where custom applications are running. Connection string will have the following format in this case: **`"balancer:[ftp, http URL]"`**

System administrator has more flexibility in this approach, because he/she could change cluster configuration (balancer's config file) in one place without updating on every client node.

Diagram below illustrates this approach:



Cluster configuration example

There are examples of configurations files below. Cluster consists of one MASTER and two SLAVES servers:

MASTER vdb.properties

```

rsi.enabled = true
rsi.host = master.cluster.com
rsi.port = 47777
rsi.cluster.enabled = true

storage.data = storage/vyhodb.data
storage.log = storage/vyhodb.log

admin.enabled = true
admin.host = master.cluster.com
admin.port = 46666
  
```

SLAVE 1 vdb.properties

```

rsi.enabled = true
rsi.host = slave1.cluster.com
rsi.port = 47777
rsi.cluster.enabled = true

storage.data = storage/vyhodb.data
storage.log = storage/vyhodb.log

slave.enabled = true
slave.master.host = master.cluster.com
slave.master.port = 46666
  
```

```
slave.mode = realtime
```

SLAVE 2 vdb.properties

```
rsi.enabled = true
rsi.host = slave2.cluster.com
rsi.port = 47777
rsi.cluster.enabled = true

storage.data = storage/vyhodb.data
storage.log = storage/vyhodb.log

slave.enabled = true
slave.master.host = master.cluster.com
slave.master.port = 46666
slave.mode = realtime
```

balancer.properties

```
read.poolCount = 2

modify.host = master.cluster.com
modify.port = 47777

read.0.host = slave1.cluster.com
read.0.port = 47777

read.1.host = slave2.cluster.com
read.1.port = 47777
```

Inconsistent reading

Inconsistent reading can occur on cluster configuration. The following case illustrates what is inconsistent reading.

Consider Java custom application which sequentially invokes @Modify and @Read methods which are changing and reading the same data. In case, when replication timeout [**slave.checkTimeout**] is longer than period between methods invocations, @Read method might read old version of data, because changed data on MASTER server haven't been replicated to SLAVE server yet.

This case isn't a problem, when @Modify and @Read invocations are parts of different business transactions: for instance adding new line into order (@Modify) and creating sales report (@Read).

The situation is getting worse, when @Modify and @Read invocations are part of the same business transaction: adding new line into order (@Modify) and reading the whole order (@Read).

Problem solving

For preventing inconsistent reading, property [**rsi.cluster.enabled=true**] must be configured on all cluster nodes (MASTER and SLAVE servers). This property changes RSI Server behavior the following way:

- 1) With @Modify method invocation response, RSI Server component sends information about last Modify transaction. RSI Client API binds this information to custom application's thread, which invoked @Modify method.
- 2) When custom application invokes @Read method, using the same **com.vyhodb.rsi.Connection** object as in @Modify invocation, RSI Client API sends info about last Modify transaction to RSI Server (as part of invocation request).

- 3) RSI Server component (on SLAVE server) checks whether required Modify transaction has been replicate or not. If it hasn't been replicated yet, than it suspends RSI invocation for a period, specified by configuration property [**rsi.cluster.probe.timeout**].
- 4) If after [**rsi.cluster.probe.attempts**] attempts (with [**rsi.cluster.probe.timeout**] timeout between them) required Modify transaction hasn't been replicated, then RSI Server component terminates RSI invocation with exception.

Therefore, for preventing inconsistent read in cluster configuration, the following properties should be configured on each server:

Configuration property	Description	Default value
rsi.cluster.enabled	Switches on/off inconsistency reading safety mechanism. Applicable only for cluster configuration. For more details see Inconsistent reading section.	false
rsi.cluster.probe.attempts	Number of attempts for probing whether required transaction has replicated or not. This property is applicable only when [rsi.cluster.enabled=true] For more details see Inconsistent reading section.	40
rsi.cluster.probe.timeout	Timeout (in milliseconds) between probes whether requires transaction has replicated or not. This property is applicable only when [rsi.cluster.enabled=true] For more details see Inconsistent reading section.	100

Fault tolerance

Fault tolerance is implemented partially: if SLAVE server is unavailable, then balancer tries to forward RSI request to available SLAVE server seamlessly. If MASTER server is unavailable, then balancer terminates current RSI invocation and throws an exception back to custom application.

Cluster configuration doesn't support automatic server recovery. System administrator is responsible for this. However, automatic server recovery can be implemented using operation system facilities, virtualization and shared disk systems.

Read-Only Cluster

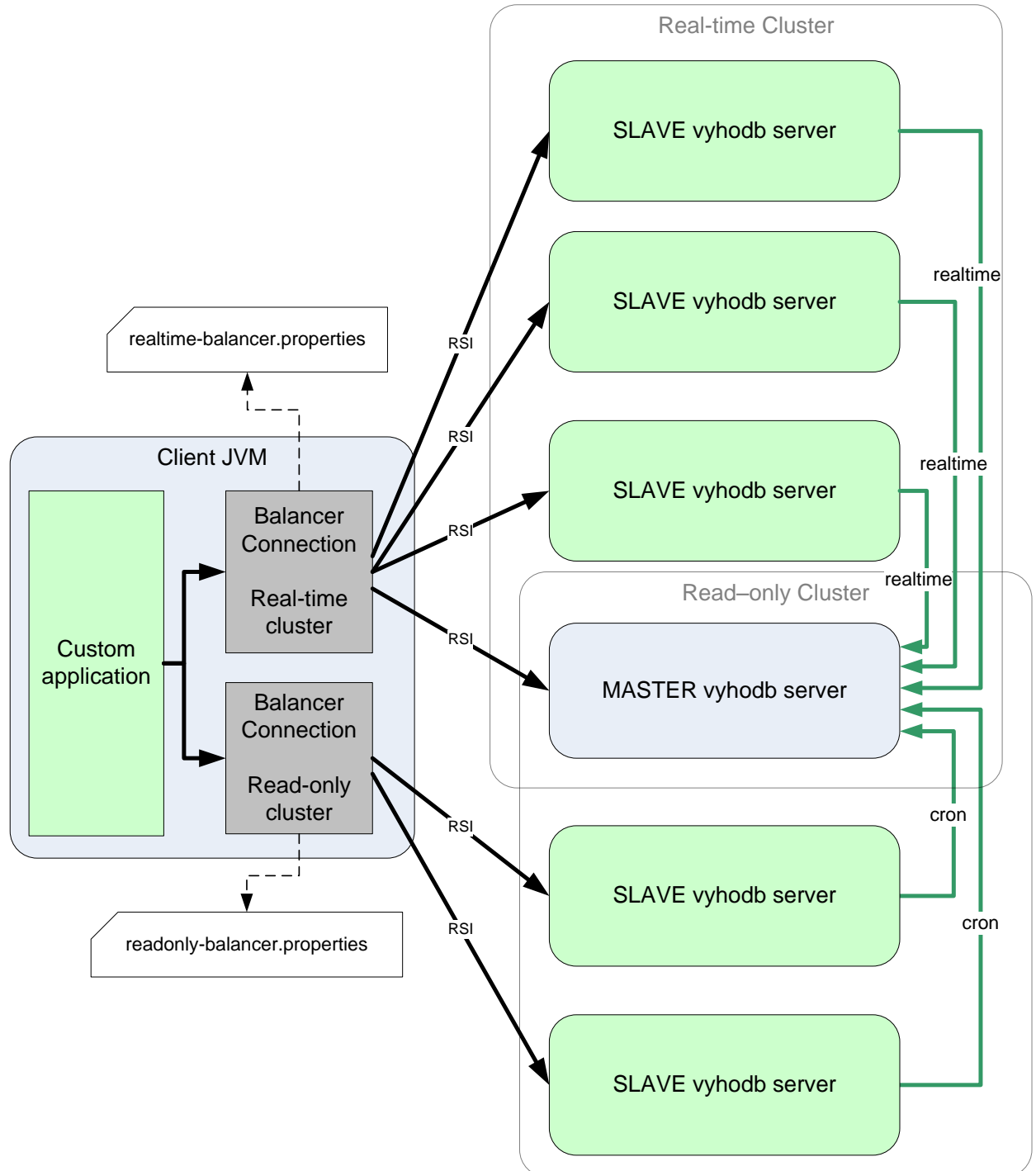
Some transaction types don't require actual data (for instance report generating). Read-only cluster configuration can be used for these purposes, which is intended for reducing replication and heavy read transactions overhead from MASTER server.

Read-only cluster distinctive traits:

- 1) Configuration property [**rsi.cluster.enabled**] is switched off (rsi.cluster.enabled=false), because custom applications invokes only @Read methods and inconsistent read can't happen (if MASTER server participate in real-time cluster configuration then this property must be set to **true** on it).

- 2) It is better to use **cron** mode for Slave Replication Agents on SLAVE servers, so that replication sessions occur only at low workload hours.

Diagram below show example of custom application which has two connections: the first one with real-time cluster for OLTP transactions (read and modify data) and the second one with read-only cluster for running heavy read transactions.



Logging

Vyhodb uses slf4j framework (www.slf4j.org) for logging. vyhodb is shipped with Simple Logger implementation, which just prints on console all log messages (<http://www.slf4j.org/api/org.slf4j/impl/SimpleLogger.html>).

To create more complex logging scenarios (message filtering, forwarding, etc) system administrator should download required logger framework implementation and configure it.

Table below shows logger names, which are used by vyhodb components:

Logger name	Components
com.vyhodb.storage	Space API, Page Storage
com.vyhodb.rsi	RSI Server
com.vyhodb.admin	Admin
com.vyhodb.slave_agent	Slave Replication Agent

Appendix A. System limits

	Minimal	Maximum
Record size	139 bytes	1073741824 bytes (1 Gb)
Field count in record	0	32767
Field count in composite index	2	32767
Child records count per one parent	0	$(2^{63}) - 1$
Read Cache size	0 pages ²	2147483646 pages (~ 2 Tb)
Log Buffer size	5 pages	2097149 pages (~ 2 Gb)
Modify Buffer size	0 pages	2097149 pages (~ 2 Gb)
Data file size	2 pages	9007199254740991 pages (~ 8191 Pb)
Log file size	3 pages	9007199254740991 pages (~ 8191 Pb)

² Page size is 1024 bytes