Distributed Tasks Manager Application

General architecture specification, phase A

The DTM application is multithreaded server class software. Inter process interactions uses ZMQ TCP sockets connections server and client type. In process interactions uses ZMQ inproc sockets connections servers and clients. Asynchronous inproc resources access as well as requests processing provided by MOM-based transport provided by ZMQ sockets technology. No system or POSIX access controlling objects like mutexes or semaphores used. The only one kind of processing item is message. The several messages containers types like queues, dictionaries, lists and so on can be used to accumulate and organize processing sequences. The multi-parametric selections engine container is SQLite.

There are three listened by DTM application TCP ports:

1. Administration.
2. Client.
3. Tasks state update.

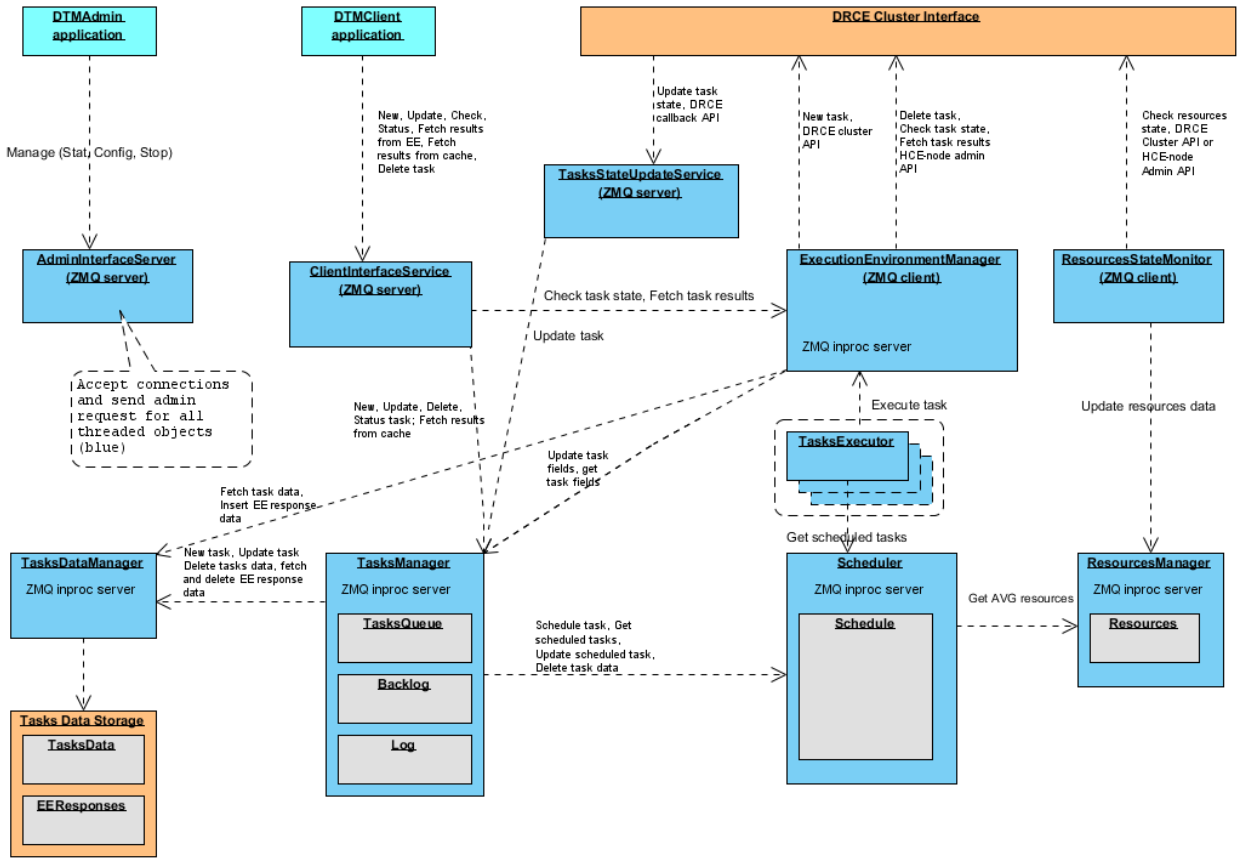
Administration – handles requests from tool client applications. Requests of that type fetches statistical information from all threading objects as well as can be used to tune some configurations settings of DTM application directly at runtime period.

Client – handles requests from client side API or tool application. Requests of that type are main functional queries for tasks and related with HCE DRCE interaction protocol.

Tasks state update – handles requests from some clients that are belongs to execution environment and informs about tasks state changes.

All requests are represented by json format of message. The structure of requests and main tasks logic principles are separated from concrete execution environment. This gives potential possibility to use some several execution environments engines, but not only HCE DRCE cluster.

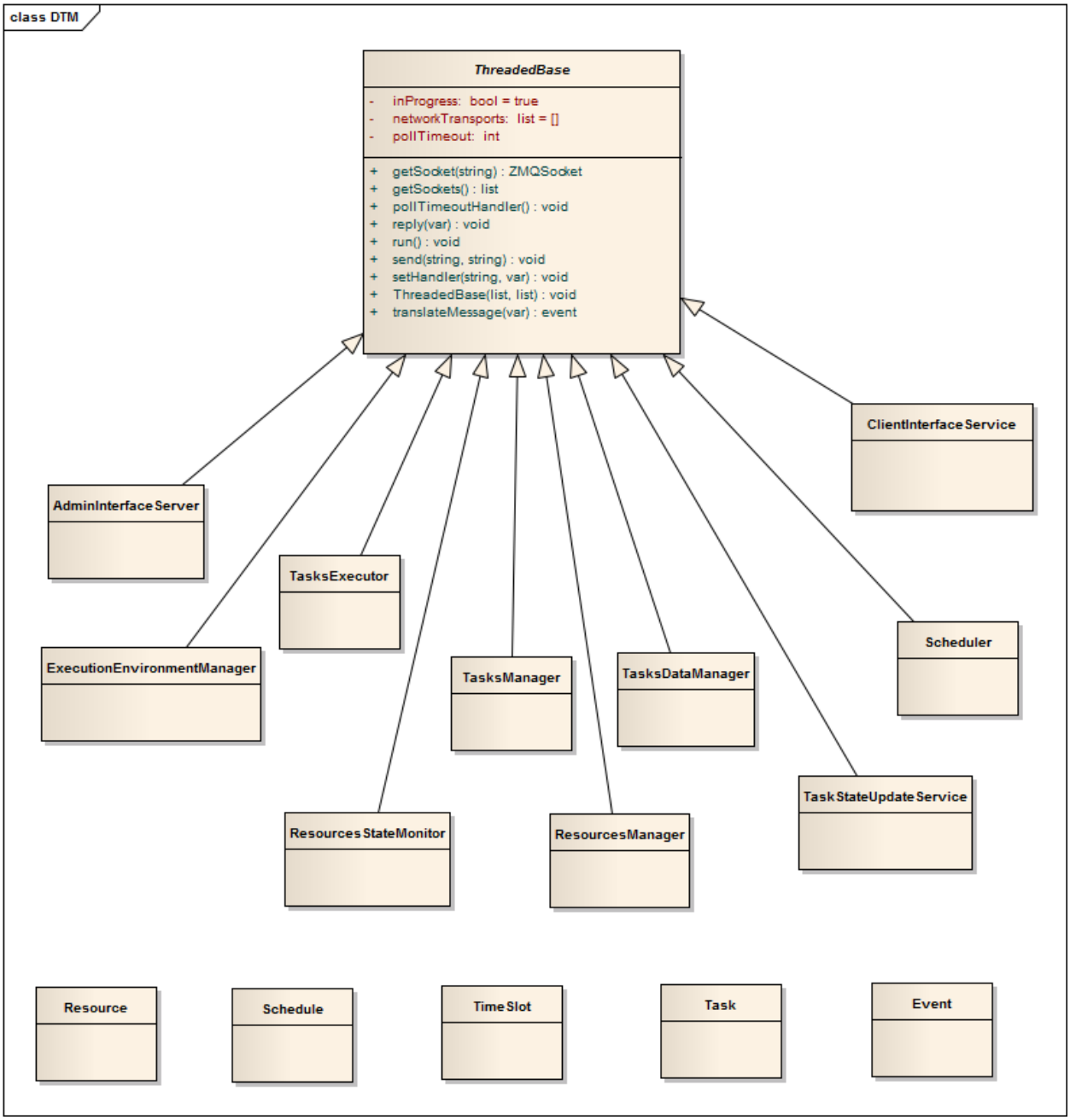
General vision



Object model

All active networking usage objects (blue primitives on architecture UML diagram) are based on threaded class object. This threaded base class structure. Also, all threaded objects uses list of transport level items of two types: client and server. Both are based on ZMQ sockets, server type can be TCP or inproc. TCP used for external connections, inproc – for all internal inter-threading events interactions.

Classes



ClientInterfaceService object

Listen TCP server connection, receives message events from external client application API and handle them, receives responses from inproc connections and send response to the external client application. There are two kinds of events:

- For the TasksManager – New, Update, Delete and Status of the task.

- For the ExecutionEnvironmentManager – “Check state” and “Fetch results” of the task.

Both are just redirections of the correspondent events to the proper inproc server connection. Require initialization with TCP listen port. If message event received from the TasksManager or the ExecutionEnvironmentManager object it redirects it with inheritance of the event type to the client application connection.

AdminInterfaceService object

The AdminInterfaceService implements dedicated server for DTM application management and monitoring. It handles request messages events from the admin management application client. There are three types of events that it process:

- **Stat** – request on complete statistical information (list of properties) from specified threaded class;
- **ConfigSet** – request on update of some internal properties that can be changed at runtime from specified threaded class. List of properties and values provided in the event object.
- **ConfigGet** – request on read of some internal properties that can be accessible at runtime from specified threaded class. List of properties and values provided in the event object.
- **Stop** – request to cancel all operations and to stop application. When this message arrives the AdminInterfaceService object re-sends this event to all connected to inproc connection clients – to all threaded objects. They stops his infinite polling and finish threads. Then the AdminInterfaceService thread also finishes and DTM application exits.

This object uses two server connections TCP for external client requests and inproc – for all threaded classes.

TasksDataManager object

This object operates with two DB storages – task’s objects data and Execution Environment (EE) responses data. Both are caches and TTL aligned for task object. The task’s objects table stores complete serialized task object with “files” container in form key-value. The key is a task Id, the value is serialized (probably Jason format) string representation of the task object. The EE responses table stores complete serialized EE response object in form key-value. The key is a task Id, the value is serialized (probably Jason format) string representation of the EEResponse object. The TasksDataManager process two types of messages from the:

- TasksManager object – insert and delete task’s data, fetch and delete EE data;
- ExecutionEnvironmentManager object – fetch task’s data and insert EE data;

and correspondent responses for six types of requests.

Both types of handled messages processed to requests to the DB backend and correspondent response event message created and returned as result of operation. Requires configuration on creation and possible runtime of DB backend dependent type (backend name, host, port, table, etc…) and request timeout. The default backend type is “sqlite”.

TasksStateUpdateService object

This object implements dedicated service for callback monitoring of the tasks state. HCE nodes clients use this connection and send requests messages with task-specific information. The incoming message structure defined in the DRCE FO protocol specification as “**Response protocol**”.

When “**update task**” request message received from the DRCE callback API it handled and new “**UPDATE_TASK**” event created and-it sends this event to the TasksManager object that updates correspondent data on several task-related structures. Properties of this event correspond with the Backlog table fields, like the “**State”** field, values are the same. For example, the “**state**” field from incoming request message translated to the “**State**” field of the event **UpdateTask** object. No response event from the TasksManager.

ExecutionEnvironmentManager object

This class represents a collection of interfaces to the Execution Environment that is used to execute the task job. The first supported interface is HCE in form of two connections - persistent “**DRCE cluster API**” and not persistent “**node admin API**”. The concrete EE can be configured at start or runtime period. The main work of this class is to translate the Task object in to the EE request, execute request, get a response and send a response message event back. It listen two connections and make tree connections. There are two types of requests and connection events processed, from:

- ClientInterfaceService – “Check state” and “Fetch task’s results”;
- TasksExecutor – “New task” and “Delete task”.

In both cases when request message event received it converts them to the correspondent EE protocol request and using correspondent EE API – sends the request to the EE then wait on another event from any of polled connections. If response on EE request event is arrived – it converts it from the EE API representation to the correspondent response event and sends response back to the submitter.

To process an “**ExecuteTask**” request event it needs to get a task’s data object from the TasksDataManager (only the task Id is received from the TasksScheduler). The “**FetchTaskData**” synchronous request sends to the TasksDataManager object. The task’s data can be an object of two types:

- **NewTask** – used to execute correspondent “Set new task” request to the EE. In case of DRCE it is DRCE Cluster API request.
- **DeleteTask** – used to execute correspondent “Terminate task” request to EE. In case of DRCE it is HCE node Admin API request.

The type of object, returned on "**FetchTaskData**" request defines the operation type that will be applied to the EE interface. In case of DRCE Cluster – two different APIs can be involved.

After the "**NewTask**" or "**DeleteTask**" request response got from the EE – it sends "**UpdateTaskFields**" request to the TasksManager and "**InsertEEResponseData**" to the TasksDataManager.

To process "**FetchResults**" and "**CheckState**" the "**GetTasksFields**" additional synchronous request to the TaskManager need to be sent to get task related data (for DRCE it is the host and the port that is used to query HCE node by Admin API).

After response from the EE got, in case of "**FetchResults**" request additional request to the TasksDataManager "**InsertEEResponse**" before EEResponseData object returned to the ClientInterfaceService. In case of "**CheckTaskState**" request the EEResponseData object just returned in correspondent response to the ClientInterfaceService.

The ExecutionEnvironmentManager never responds on "**ExecuteTask**" event to the TasksExecutor.

TasksExecutor object

The task executor object can to have multiple independent instances. Each instance works as two connections client. It connects to the ExecutionEnvironmentManager object and the Scheduler object. Main job of this object is a selection of scheduled tasks from the schedule and send them to the ExecutionEnvironmentManager object to set them to the Execution Environment for execution. The selection criterion is only **time slots range**. After creation object enter in polling of the Scheduler object connection with timeslot period. If no response event from the Scheduler object – it enters next iteration of polling. If response from the Scheduler is not empty – request to the Execution Environment Manager sent for each task item in the received list one by one or by all items in list (must be configurable at start and runtime period). The time of this operation need to be evaluated and compared with the time slot value. In case of time slot value is grater, it sleeps this time diff to provide the system to execute another threads and to not to load CPU. The value of time slot size is key parameter that defines the latency of the whole DTM application as a task scheduling system, so it must to be configured on the application level and for the class instance at creation and runtime. Depends on intensity of requests on "New task" from client side and tasks planning strategy, the time slot size value can be from 1000 to 10 msecs.

ResourcesStateMonitor object

ResourceStateMonitor object it is the watchdog of DRCE cluster's computational resources (CPU load, threads number, processes number, disk space size and so on). It queries the DRCE cluster by router type request or DRCE cluster's data hosts directly by admin type requests, collects information about computational resource items and send **"update state"** requests message events object to the ResourcesManager object. Requires configuration on creation of the type of collect algorithm:

- "**DRCE cluster**" (host, port, timeout for router connection)
- "**HCE host**".(hosts, ports, timeouts for hce-nodes admin connections)

and period of information update. One of possible way is auto-configuration on start or on demand with cluster scanning or update from external source.

In case of positive response from EE it sends “update data” request message event object to the ResourcesManager object and no response waiting. In case of timeout of EE update request only logging action performed.

ResourcesManager object

This object process “update data” request message event from the ResourceStateMonitor object and updates information about DRCE cluster computational resources units inside own resources container. Also, it process the “get resources” request message event from the Scheduler object, fetches resources data summarized by average calculations. The **Resources object** container structure definition represented below.

The response object for “**GetAVGResources**” request returns values of average resources usage rates in % pre-calculated after each resource update event and stored as runtime data inside the ResourceManager object.

The “**UpdateResourcesData**” request receives one or more resource items and updates its data in case of resource exists in the Resources object container and inserts new if not. After all resource items processed the ResourcesAVG data value recalculated. The calculation algorithms:

Field	Formula
cpu	AVG(Resources.cpu)
io	AVG(Resources.io)
ramR	AVG((Resources.ramRU/Resources.ramR)*100)
ramV	AVG((Resources.ramVU/Resources.ramV)*100)
swap	AVG((Resources.swapuU/Resources.swap)*100)
disk	AVG((Resources.diskU/Resources.disk)*100)
uDate	MIN(uDate)
cpuCores	AVG(Resources.cpuCores)
threads	AVG(Resources.threads)
processes	AVG(Resources.processes)

only records with “**state=0**” included in average calculation.

Resources object

The **Resources** container object implemented as embed DB table (possible implementation is slite DB).

Resources container fields schema:

- **nodeId** – the unique node identifier in format “host:port”, String;
- **name** – node name from hce-node specification, String;
- **host** – the TCP host name, String;
- **port** – tcp port, Integer;
- **cpu** – CPU load, %, Integer;
- **io** – i/o wait, **%**, Integer;
- **ramRU** – resource RAM size used, byte, BigInteger;
- **ramVU** – virtual RAM size used, byte, BigInteger;
- **ramR** – total physical RAM size, byte, BigInteger;
- **ramV** – total virtual (physical + swap) RAM size, byte, BigInteger;
- **swap** – total swap size, bytes, BigInteger;
- **swapU** – total swap size used, byte, BigInteger;
- **disk** – disk size, byte, BigInteger;
- **diskU** – disk size used, byte, BigInteger;
- **state** – the current state depends on last data update operation. Possible values are: 0 – active, last update operation is okay; 1 – undefined – last update operation failed; 2 – inactive – several last update information failed or state defined by another external criterion and updated by admin request message event, Integer.
- **uDate** – last update local timestamp, msec DateTime.
- **cpuCores** – number of CPU cores, BigInteger
- **threads** – number of run threads, BigInteger
- **processes** – number of run processes, BigInteger

TasksManager object

It is a main object that manages tasks.
When **new task message event** received from the ClientInterfaceService it does:

- stores the complete task object in the TaskDataStorage storage by send “**InsertTask**” request to the TaskDataManager. If fault response received, removes task record by the task Id from TasksQueue and Backlog, sends “DeleteTask” to the Scheduler, send response to the ClientInterfaceService with correspondent error code.
- erase the “files” container and stores the task object in the TasksQueue container,
- inserts new record in to the Backlog table,
- send set “ScheduleTask” request to the Scheduler. If fault response received from the Scheduler – check error state and in case of rescheduling possible, change task’s properties and send “RescheduleTask” request to the Scheduler. In case of rescheduling is not possible – delete task from TasksQueue, delete task from backlog, send delete task’s data request for

TaskDataManager and send response to the ClientInterfaceService with correspondent error code.

When **update task message event** received from the ClientInterfaceService it does:

- update the complete task object in the TaskDataStorage storage by send "**UpdateTask**" request to the TaskDataManager if task present in the Backlog. If fault response received and task is not already run, removes task record by the task Id from TasksQueue and Backlog, sends "DeleteTask" to the Scheduler, send response to the ClientInterfaceService with correspondent error code. If task is already run – return response to the ClientInterfaceService with correspondent error code.
- update the TaskQueue
- update the Backlog record if task not run or still in progress
- send "UpdateTask" request to the Scheduler.
- send response to ClientInterfaceService.

When "DeleteTask" event arrives the TaskManager updates state of the task in the "Backlog" to "scheduled as delete", updates task's data in the TasksDataManager and set this task in to the schedule ASAP type. When "Update" event arrives from the "TaskStateUpdateService" with status information "TaskDeleted", it set state of the task in the "Backlog" as "Terminated", moves task's record from the "Backlog" to "Log" container, delete task's data in the TaskDataManager (both task's and EE responses' data).

TaskQueue container

Task queue it is a container of tasks objects in form of accepted from the client side with "New task request". Objects items accessible fast way by the task Id. Possible implementation is dictionary or hash array.

Backlog – container for tasks that scheduled but are not executed or assigned for continuous (long or periodic) execution. Log – container for tasks that was executed and completely finished, not periodic and not continuous execution. Backlog supposes active runtime update, possible require completely in memory location. The size of backlog is limited by execution environment and maximum tasks number that was set to be executed in future. Log size is huge, possible saves data for long time of application session. Log supposes less activity that Backlog and inserts operation as most often used. Both requires dumping and save state for application restart. Possible implementation as two independent internal database tables (sqlite) supports SQL CRUD operations.

During application life time tasks, after scheduled, inserted in Backlog table. Then, can be updated with state information (execution time, and, resources usage) and with finished state, finally. After task was updated as finished state it moved from Backlog to Log table.

Backlog and Log container fields schema:

- **id** – task id, BigInteger;
- **pId** – the OS-specific parent artifact process Id, BigInteger;
- **nodeName** – node name from hce-node specification, String;

- **cDate** – creation date, DateTime;
- **sDate** – scheduled date, DateTime;
- **rDate** – Run date, DateTime;
- **fDate** – finish date, DateTime;
- **pTime** – progress time, msec, BigInteger;
- **pTimeMax** – maximum time that task can be in progress state. If PTime > PTimeMax task will be terminated by EE, BigInteger;
- **state** – task state, 0 – finished, 1 – in progress, 2 – set as new, 3 – not found, 4 – terminated before normal finish, 5 – crashed, 6 – not set as new cause to limits of system resources (CPU, RAM, Disk, threads, processes, etc), Integer;
- **uRRAM** – utilization of RAM, byte, BigInteger;
- **uVRAM** – utilization of virtual RAM, byte, BigInteger;
- **uCPU** – % of CPU utilization, Integer;
- **uThreads** – number of POSIX threads that was created by task's process. Updated from EE, Integer;
- **tries** – number tries to run, Integer.
- **host** – EE task's host, String.
- **port** – EE task's port, String.

All fields require indexes for fast records selection with combination of any fields as criterions.

Scheduler object

The Scheduler object implements algorithms of tasks scheduling and container with scheduled tasks – the Schedule object. It listen server connection and handles event messages from two type of objects – the TasksManager and TasksExecutor object (can have more than one instance) and make requests using one client connection to the Resource Manager object.

The "**SheduleTask**" request event processed as a kind of planning algorithm for new task. Depends on concrete algorithm some internal structures used to check conditions. The main information about already planned tasks stored in the **Schedule** container object, see definition below. Most useful and first for implementation is an **ASAP strategy**.

ASAP strategy – supposes that new task need to be placed in to the schedule container in to the first position free position nearest in time, according with other additional criterions, like resources limits, possible specified. The operation includes get last planned tasks for the nearest time slot, evaluate availability of free place for one new task, and repeat this action until the place will not be found. When place is found, the time and another limits need to be verified. If time is out of specified hard range, or some resource limit exceeding – planning fault response message sent to the TasksManager and no task inserted in to the Schedule container. If the time limit has no hard range defined – it can be moved forward and next free time slot position can be used to set the task in to the Schedule.

If no resource limits defined the new task item filled with default values of limits will be inserted in to the Schedule and positive success response returned to the TasksManager. In case of some resource limits defined in the new Task – the Scheduler object make request event GET_AVG_RESOURCES to the **ResourcesManager** object and fetches AVG resources status information as ResourcesAVG event object. The ResourcesAVG event object contains average data that is pre-calculated after last resources information update from the EE by **ResourceStateMonitor** object.

Fixed time strategy – supposes that new task need to be placed in to the concrete time slot or nearest available slot in future. The nearest future time slot left border calculation formula:

leftBorderMs = floor(CUR_TIME_MS / TIME_SLOT_MS)

Right border value calculated as:

rightBorderMs = leftBorderMs + TIME_SLOT_MS

Next time slot left border value calculated as:

nextLeftBorderMs = leftBorderMs + TIME_SLOT_MS

The strategy fields DATE, DATE_MAX, DATE_SHIFT and MAX_TASKS used to evaluate the place of the new task in concrete time slot. To calculate current number of tasks in the time slot – tasks that already scheduled for time range from left border to right need to be calculated.

Other strategies will be described later.

Get scheduled tasks request handling from the TasksExecutor.

After selection by the time slot period provided in request tasks list filtration by several criterions goes on. This process uses current EE resources provided by the ResourcesManager from AVGResources object. Filtration criterions and conditions equation to remove task from selected list:

- **CPU utilization per threads.** Condition:
  (ResourcesAVG.threds/ResourcesAVG.cpuCores) > strategy.CPU
- **CPU load max.** Condition:
  ResourcesAVG.cpu > strategy.CPU_LOAD_MAX
- **IO wait max.** Condition:
  ResourcesAVG.io > strategy.IO_WAIT_MAX
- **Committed (RSS) RAM utilization.** Condition:
  ResourcesAVG.ramR > strategy.RAM_FREE

The "**GetScheduledTask**" request event from the TasksManager object processed as check in schedule the task by several select criterions like Id and returns list of the **ScheduledTask** event objects

represents items of the Schedule container. This request can be used by the TasksManager to ensure the task state and evaluate planning algorithm result.

The "**UpdateScheduledTasks**" request events from the TasksManager object processed as update of the existing tasks in the schedule according with new value of the planning strategy. This request can be used for periodic tasks re-activation or the repeat try to execute in case of some task errors or faults. The response of this operation is operation state success or fault. Request operation grouped, response is list of GeneralResponse objects for each updated item.

The "**DeleteTaskData**" request processed as complete removing of the specified task item from the Schedule container.

Schedule object

Schedule object it is a container of items that represents tasks assigned to time slots. Time slots is not represented directly as items of some container, but used as time interval for selection of planned tasks from the schedule object. The task manager sends new tasks to the scheduler and it executes the planning process and inserts new schedule item in positive result of planning or not in negative. The planning process compares requested planning strategy, resources state, time space and evaluates possibility to insert (to plan) one more task according with the limited conditions criterions specified with the task. Planned tasks ordered by planned time to run are selected by the Tasks Executor and forwarded to the Execution Environment Manager (EEM) to be sent in to the Execution Environment (EE). After EEM operation done, response from EE sent to the Task Manager that updates task state and queries the Scheduler to remove task from the Schedule or update its state in case of the task run fault and it need to be selected in next time slot or later (rescheduled). Item removed from the schedule container in case of the task is successfully set to EE.

Schedule container fields schema:

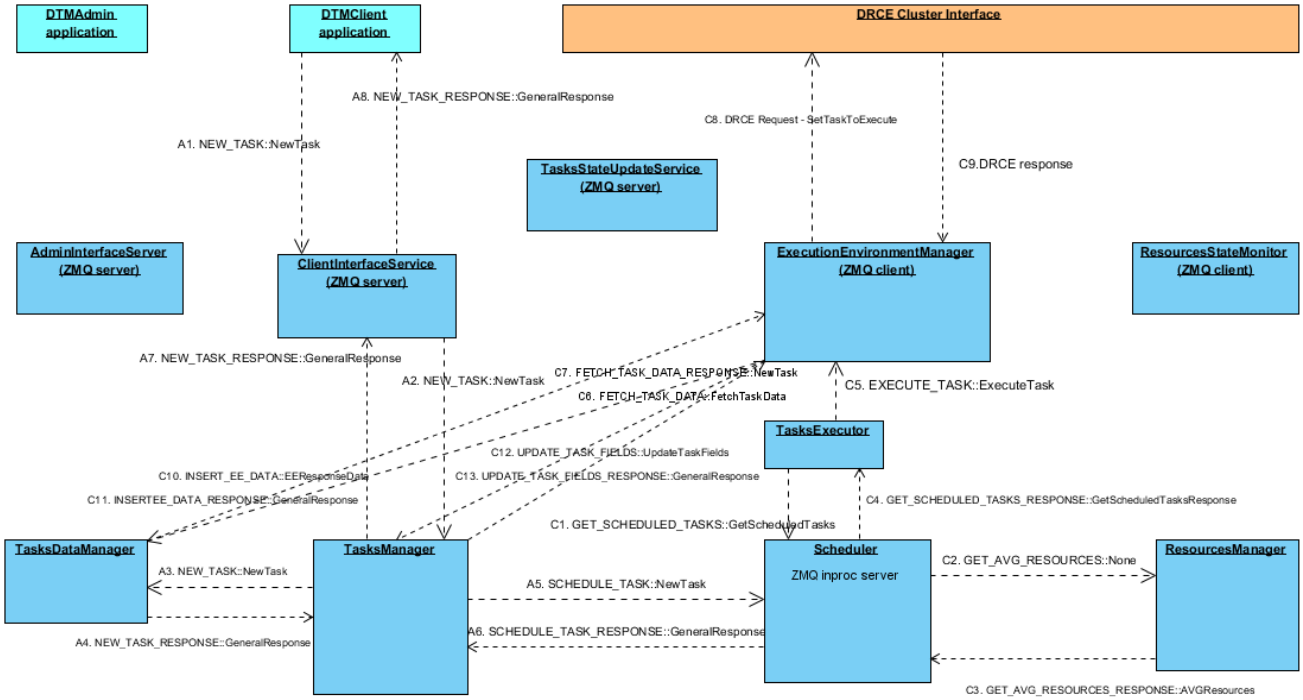
- **id** – task id, BigInteger;
- **rTime** – time to run, msec. Zero value signs ASAP way and item will be selected first, BigInteger;
- **rTimeMax** – max value of current time that enables run. If Task Executor can't to run selected task until this time the Tasks Scheduler need to delete this task from the schedule or reschedule it. Set from the "date_max" of the task's "strategy", BigInteger.
- **state** – state of the task in planning process. 1 – task planned, 2 – task was selected by Task Executor to set to EE; Updated after item selection before returned to the Task Executor, Integer.
- **priority** – the priority that used as next sort order criterion for tasks selection after the time to run. Higher value means higher priority, Integer.
- **strategy** – the serialized strategy container from NewTask object. String. Contains serialized data of all fields that was set when new task arrives. Strategy fields used for tasks list filtration during GET_SCHEDULED_TASKS event processing after selecting by time slot.
- **tries** – number of select task tries. If task was selected to run but not enough resources in the EE it will be eliminated by filtration and tries will be incremented, BigInteger.

NewTask object, UpdateTask object, CheckTasksState object, GetTasksStatus object, FetchTasksResults object, FetchTasksResultsFromCache, DeleteTasks object

These objects are similar stricture and represents fields containers used as client side user API objects for event in correspondent request. They hold fields defined in the "**fields**" container of **DTM application client message protocol** specification for correspondent request operation defined below.

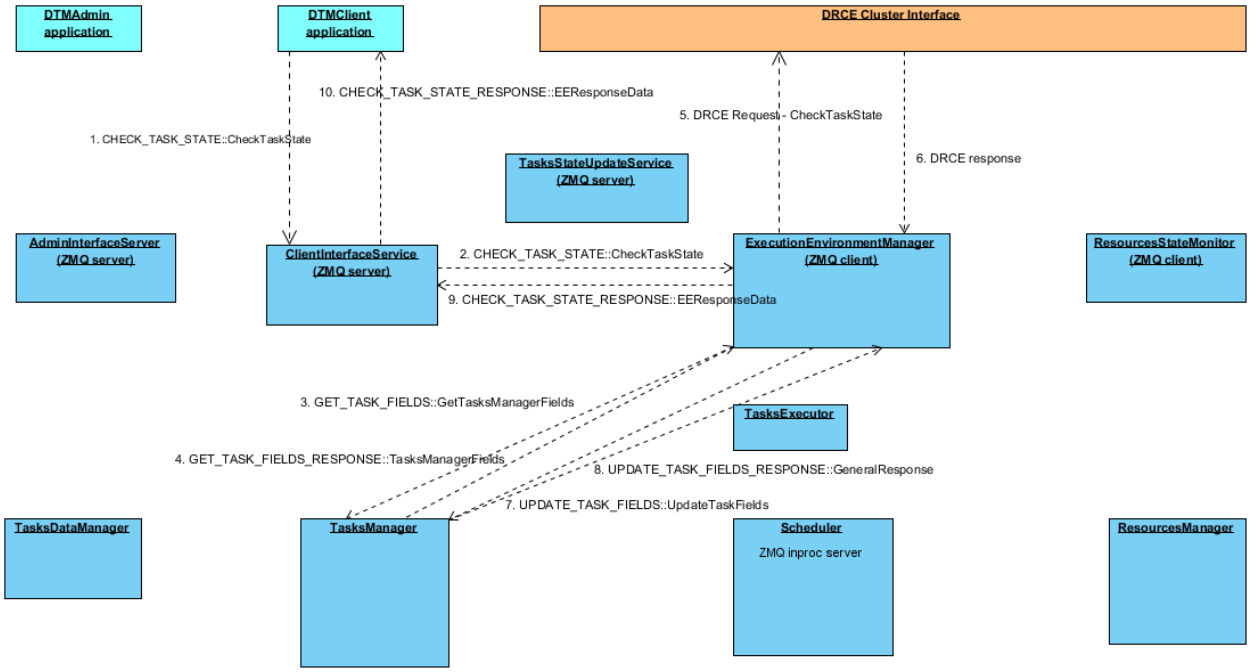
Task life-cycles

New task

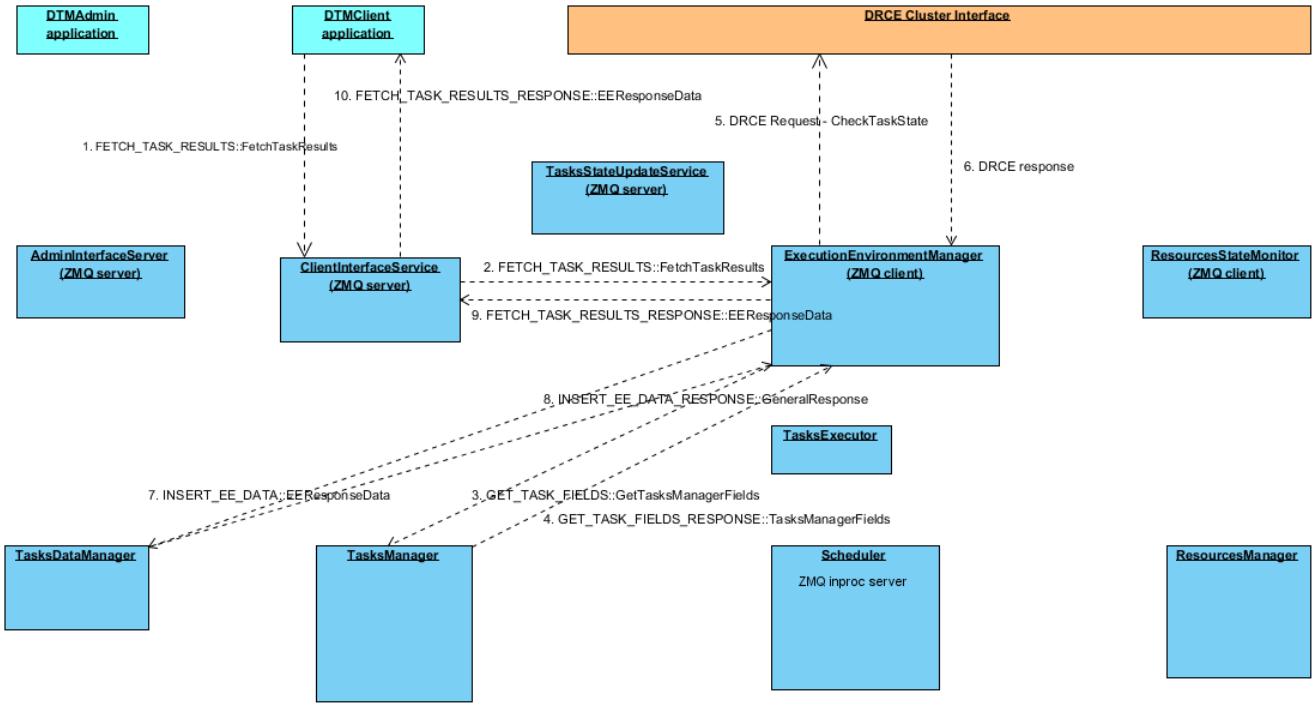


Update task

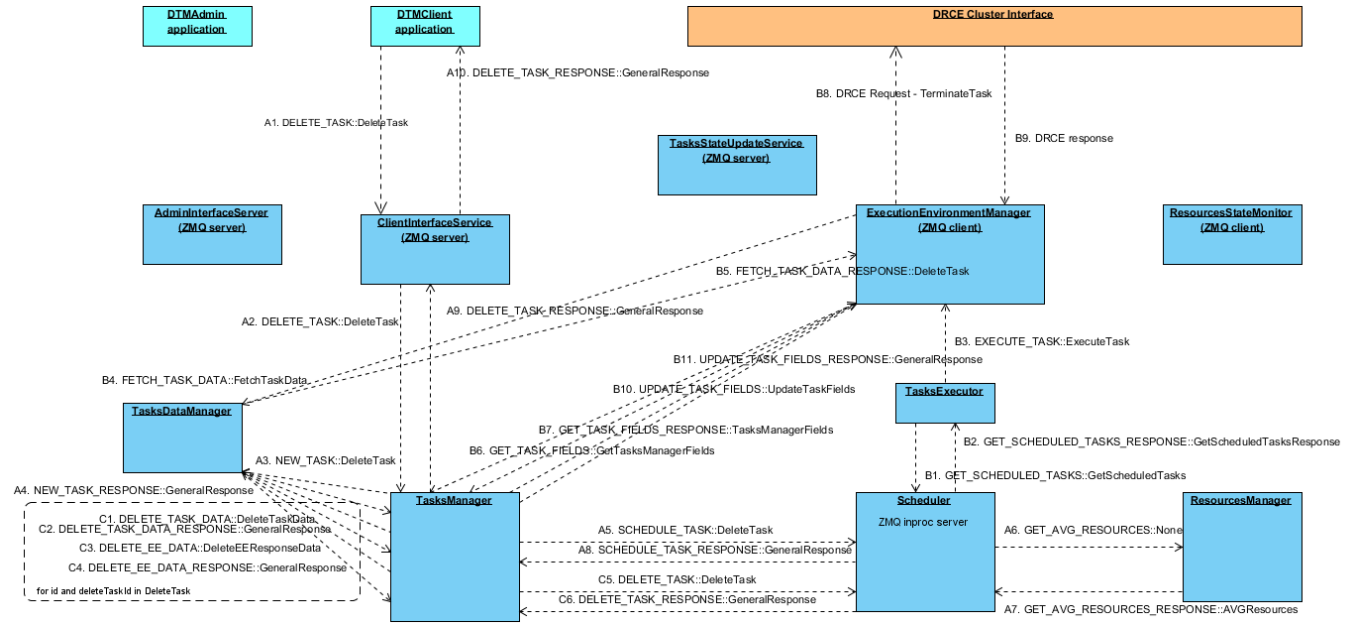
Check task state



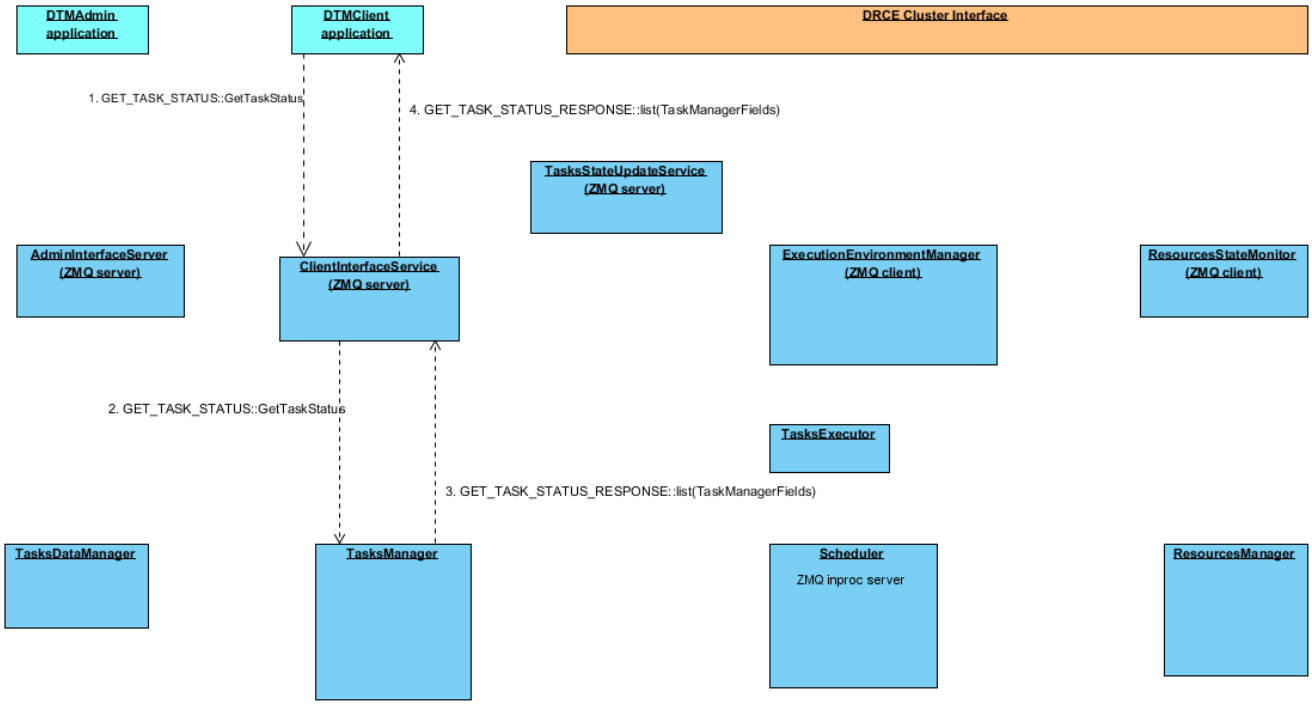
Fetch task results



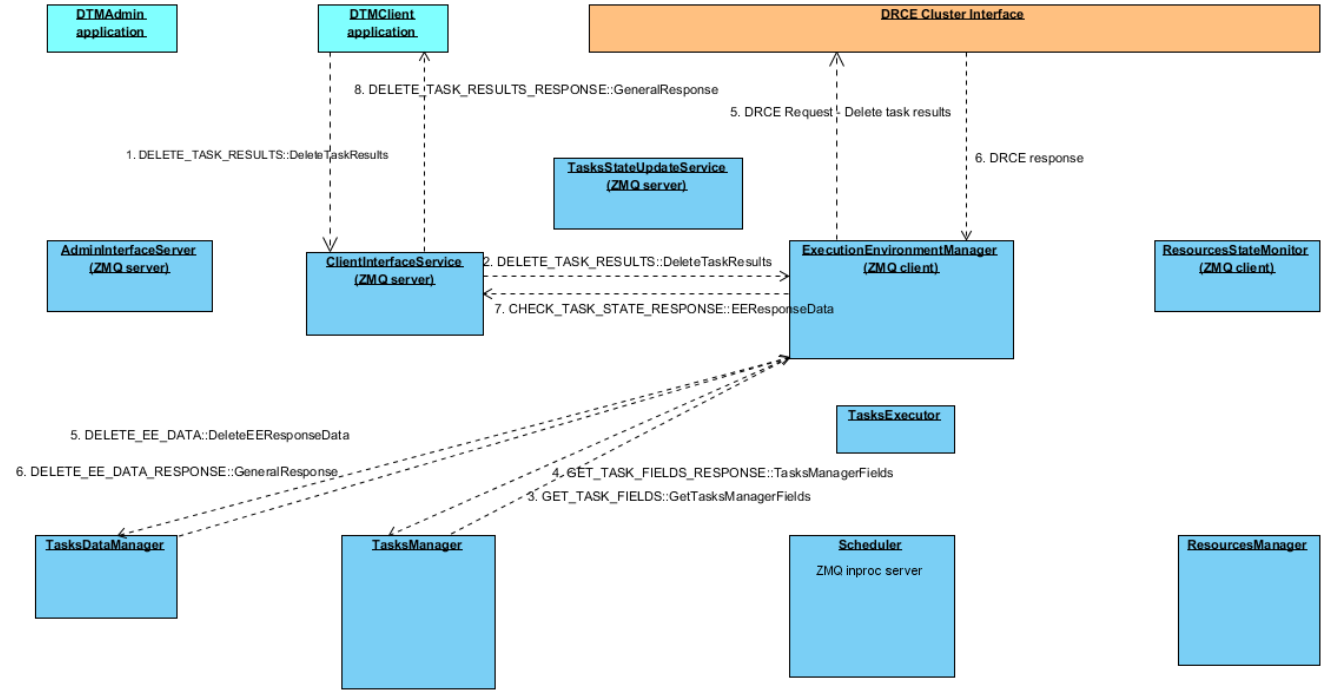
Delete task



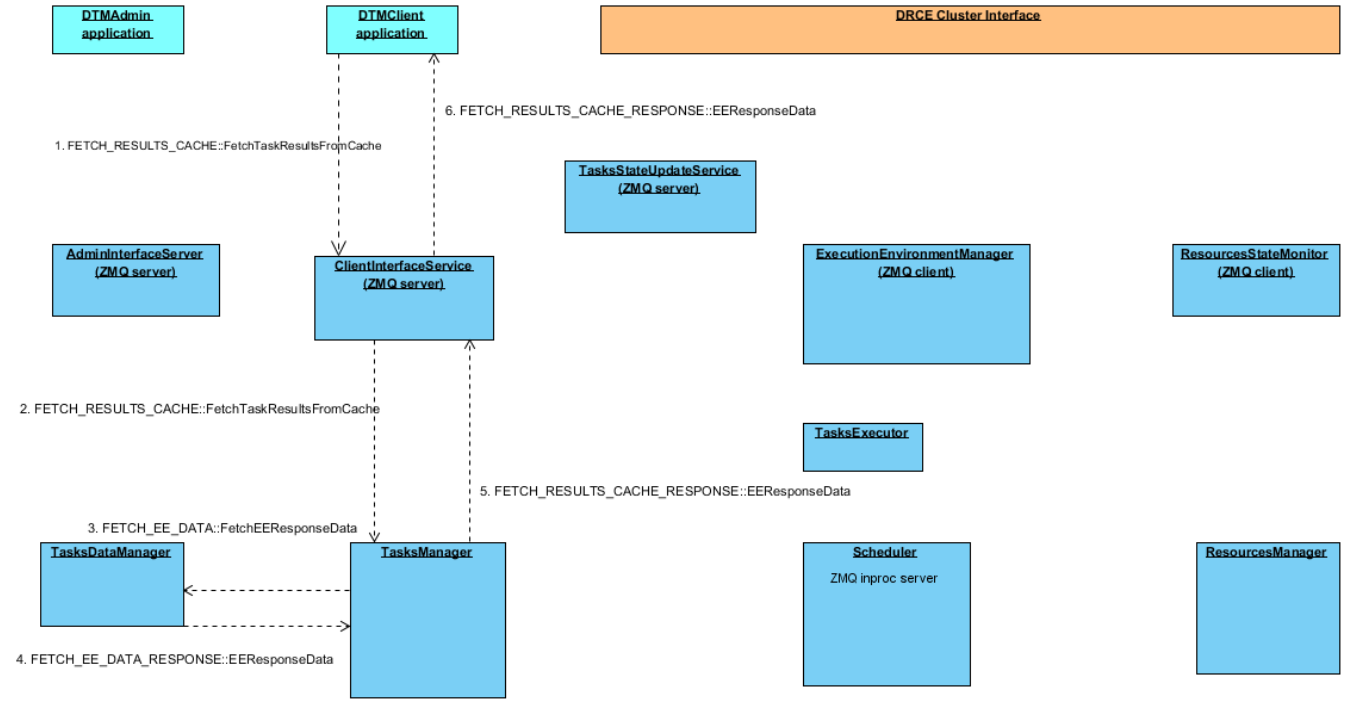
Check status



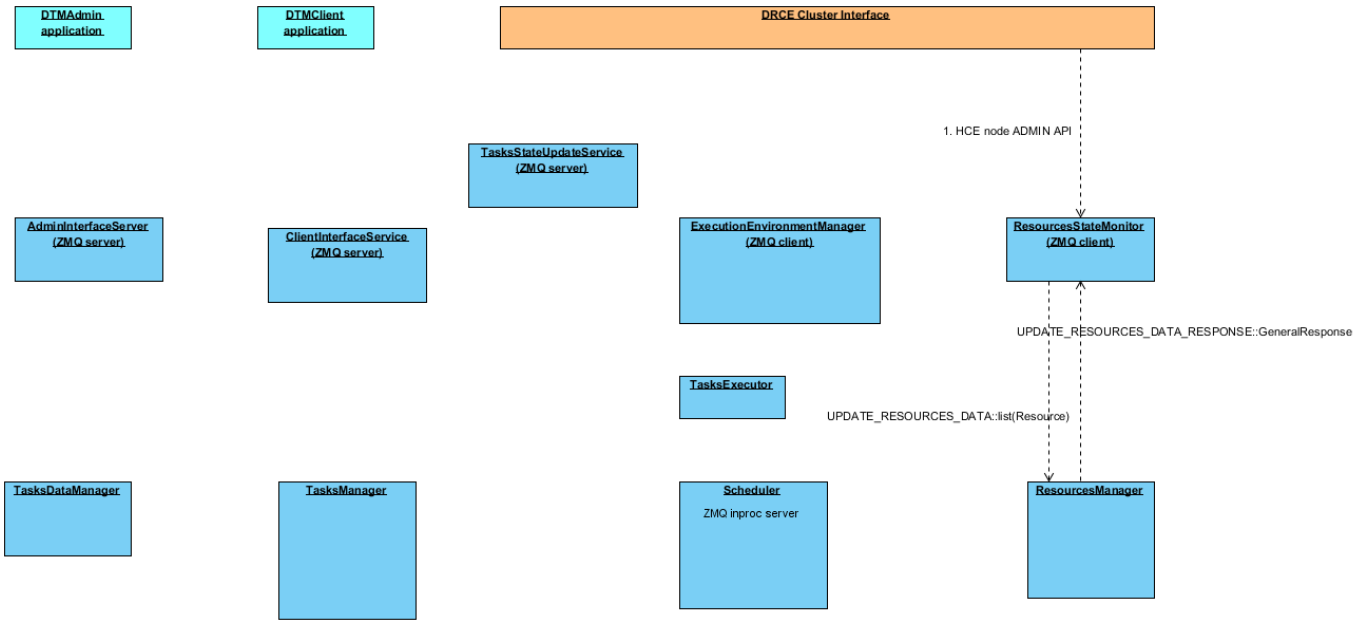
Delete task results



Fetch results from cache



Resources update



Application configuration and start

Threaded classes instantiation sequence

The threaded classes use inproc server and client connections. Inproc connections need to have server-side ready (bind and listen) to make connect operation from client side. This condition requires some instantiation sequence for threaded classes because they make connections at this time. The instantiation sequence:

1. AdminInterfaceServer.
2. TasksDataManager.
3. ResourceManager.
4. ResourceStateMonitor.
5. Scheduler.
6. TaskManager.
7. ExecutionEnvironmentManager.
8. TasksExecutor(s).
9. TasksStateUpdateService.
10. ClientInterfaceService.

Command line arguments

DTMD

- --config, -c – configuration file path name. Mandatory parameter, if application cant to load config file it exit(1) with corresponding error message on console, no demonization, no application functionality started. If omitted application tries to open file with the same name as executable module and ".ini" extension in sequence:
    - in the current directory,
    - in the /etc/<application_binary_name>/ directory,
- --help, -h – displays brief usage information, including possible command line arguments names and values and exit(0).
- --name, -n – application instance name. Optional parameter, if omitted the same name as executable module file used. Overrides the same parameter in the configuration file "Application" section.

DTMClient

- --config, -c – the same as for the "dtmd" application.
- --help, -h – the same as for the "dtmd" application.
- --task, -t – task action, specifies the requested task operation, possible values {"NEW", "CHECK", "TERMINATE", "GET", "STATUS", "CLEANUP"}. Optional if "--help" is specified, and mandatory – if not. A result of any action is file in json format. Structure corresponds with DRCE Functional Object response specification.

- --file, -f – data file for action, specified by “--task” option. Mandatory if “--task” is specified. The data file in json format, structure defined by DRCE functional object request protocol specification.
- --id – task Id, optional if “--task” has “NEW” value and mandatory for all another. If omitted, the task id is generated as md5 on the basis of current date time with milliseconds and file content specified by “--file” option. The task id is main unique task identifier for all operations and task management on the execution environment side. In case of “NEW” operation provides identifier that exists on the concrete DRCE FO node, request returned operation error. The status of other actions depends on action type. If task is not found, correspondent request error returned.

DTMAdmin

- --config, -c – the same as for the “dtmd” application.
- --help, -h – the same as for the “dtmd” application.
- --cmd – command name, specifies the requested admin management operation. Possible values are {“STAT”, “SET”, “GET”, “STOP”, “SUSPEND”}. “STOP” – lead to sequentially stop all threaded classes of application in opposite start sequence (specified in the “Server” section by the “instantiateSequence” option of configuration file) and application exit(0).
- --fields – comma separated fields list for requested operation. Mandatory, if “--cmd” is set and is not a “STOP” or “STAT” value. For “STAT” – specifies list of statistical indicators for class handler. If empty or not set – the empty list of fields used. For “GET” – specifies names of configuration options for all threaded classes and application that can be changed at runtime. For “SET” – specifies list of configuration options name:value pairs for all threaded classes and application that can be changed at runtime. All commands returned json file content. The structure depends on concrete request. It field mandatory for “SUSPEND” operation and can contains “0” - SUSPEND or “1” - RUN values only.
- --classes – comma separated list of threaded classes that will be used to execute request. Always optional for “STOP” command (always executed for all) and mandatory for all another commands. For “STOP” if omitted or empty – the list specified by the “instantiateSequence” option in “Server” section of configuration file.

The DTM application client message protocol specification

Request

All requests operate with task object in terms of DTM service, protocol message format is json.

```
{
  "type": “<operation_type>”,
  "fields": [<fields_array>]
}
```

```
operation_type – type of operation request, positive integer numeric,
                 {0 – new, 1 – update, 2 – check state, 3 – get status,
                 4 – fetch results, 5 – delete/terminate}.
fields_array – associative array of key-value fields. Concrete fields
                 list depends on operation_type value.
```

Fields list for different operations

New task

"id" – unique task identifier, positive integer numeric. If zero value – for new **operation_type** – signs that it needs to be generated. Mandatory.

“command” – command line to execute remote, string. Optional.

“input” – stdin buffer for target remote process instance, string. Optional.

“files” – array of files items, see “files array item” specification described at DRCE_Functional_object_protocol.docx document. Optional.

“session” – array of session fields items, see “session fields array” specification described at DRCE_Functional_object_protocol.docx document. Optional.

“strategy” – array of fields for the planning strategy used by the scheduler. Optional. Fields:

- **DATE** – the requested date to run task, date string “YY-m-d H:m:s,ms”. If value is empty, zero, omitted or in past this means that the task need to be scheduled to run ASAP (first nearest free time slot(s)).
- **DATE_MAX** – upper value of date that limits date range that can be used by scheduler in case of no possibility to set the task with “**date**” value. Date string, the same as “**date**”.
- **DATE_SHIFT** – maximum time shift that can be added to “**date**” value to do scheduling if “date” value time is busy. If both “**date_shift**” and “**date_max**” specified – the “**date_max**” is used, msec.
- **MAX_TASKS** – maximum tasks per time slot. Numeric, optional. Default value 0 – means no limits. Used by the scheduler to check is possible to set the task with DATE specified to some time slot.
- **CPU** – CPU resources overbooking coefficient, positive float. If zero –is not used and execution environment CPU load level state is not used as criterion when the task planned by the scheduler or candidate to run by tasks executor. If greater than zero – meant than threshold value of the rate of total task’s POSIX threads number and number of CPU cores available in the execution environment (THREADS/CORES). Positive value means that some overbooking possible. Recommended value from 1.5 to 2.5, default 2.
- **CPU_LOAD_MAX** – max value of CPU LA to run the task. Default 70.
- **IO_WAIT_MAX** – max value of CPU IO wait to run the task. Default 10.

- **RAM_FREE** – RAM resources overbooking coefficient, float. Means the % value of RAM that need to be free to run the task. If omitted – default 20% used.
- **RAM** – requested estimated minimal RAM size for the task, positive integer. If zero or omitted – means no estimation and not used during the scheduling and run.
- **DISK_FREE** – disk resources overbooking coefficient, float. Means the % value of disk space that needs to be free to run the task. If omitted – default 20% used.
- **DISK** – requested estimated minimal disk space value for the task, positive integer. If zero or omitted – means no estimation and not used during the scheduling and run.
- **TIME** – requested estimated time of work of the task, msec. Can be used to calculate number of time slots used in the schedule for this task. Default value – 1000. Optional.
- **THREADS** – requested estimated POSIX threads number that task potentially can to create. Positive integer. Zero means that task is a native shell command. Default value – 1. Optional.
- **RDELAY** – requested maximum retry delay for the re-scheduling process in case of task can't to be scheduled due time or resources cause, msec. Positive integer, zero value means that no retry used task is aborted (default).
- **SDELAY** – the same as "**RDELAY**", but for task start time.
- **RETRY** – number of reschedule and retry to run the task in case of previous is fault.
- **PRIORITY** – the task priority to be selected from the Schedule by the Tasks Executor to set in to the EE. Positive integer, higher value means higher priority.

Update task

Fields list the same as "**New task**" operation type. Depends on task state (just queued by Tasks Manager, Data stored in the storage, planned, in progress of execution inside execution environment, execution finished and so on – different task's structures and dependent properties can be updated. Updatable data and states of task will be specified later.

Check tasks state

"ids" – array of tasks Id values. Mandatory.
"filters" – array of key-value of criterions that will be used to select tasks. Optional. Fields:

- CDATE_FROM – create date from,
- CDATE_TO – create date to,
- SDATE_FROM – schedule date from,
- SDATE_TO – schedule date to,
- RDATE_FROM – run date from (run date can differ from schedule date),
- RDATE_TO – run date to,
- FDATE_FROM – finish date from,
- FDATE_TO – finish date to,
- INPROGRESS_TIME_FROM – down value of "in progress" time, msec. In progress time – the time that task is in progress state from run date to last state update.
- INPROGRESS_TIME_TO – upper value of "in progress" time, msec.
- INPROGRESS_TIME_MAX_FROM – down value of in progress time max, msec.
- INPROGRESS_TIME_MAX_TO – upper value of in progress time max, msec.
- INPROGRESS – include in progress tasks,
- SCHEDULED – include scheduled tasks,
- RUNNING – include running tasks,

- FINISHED – include finished normal way tasks,
- TERMINATED – include terminated (terminated by user request before normal way finish),
- CRASHED – include crashed (terminated by task itself of by execution environment before normal way finish)
- RRAM_FROM – amount of resource RAM min, byte
- RRAM_TO – amount of resource RAM max, byte
- VRAM_FROM – amount of virtual RAM min, byte
- VRAM_TO – amount of virtual RAM max, byte
- CPU_FROM – amount of CPU min, %
- CPU_TO – amount of CPU max, %

"strategy" – the planning strategy for the scheduler. Values the same as for new task request. Mandatory.

Get task status

"ids" – array of tasks Id values. Mandatory.
"type" – positive integer, corresponds to DRCE "check_type" field of the "Check task request". Optional.

Fetch task results

"ids" – array of tasks Id values. Mandatory.
"type" – positive integer, corresponds to DRCE "fetch_type" field of the "Get task's data request". Optional.

Delete task

"ids" – array of tasks Id values. Mandatory.
"alg" – positive integer, corresponds to DRCE "alg" field of the "Terminate task request". Optional.
"delay" – positive integer, msec, corresponds to DRCE "delay" field of the "Terminate task request". Optional.
"repeat" – positive integer, corresponds to DRCE "repeat" field of the "Terminate task request". Optional.
"signal" – positive integer, corresponds to DRCE "signal" field of the "Terminate task request". Optional.
"host" – string, HCE node host where tasks located. Optional.
"port" – string, HCE node admin port for admin connection and request query destination handler. Optional.

Response

```
{
  "error_code": <error_code>,
  "error_message": “<error_message>”,
  "time": “<time>”,
  "fields":[<fields_array>]
}
```

error_code – error state code, 0 – no errors, >0 – some error
error_message – error description text
time – internal common execution time (including request parsing and response creation), msec
fields_array – associative array of key-value fields. Concrete fields list depends on **operation_type** value.

Fields list of “fields_array” for different operations

New task

"id" – unique task identifier, positive integer greater than zero. In case of creation new task error zero Id returned. Mandatory.
"cdate" – task creation date. Optional.

Update task

"fields" – string with list of updated fields and/or properties of the task. Optional.

Check state of the task

"type" – execution environment type, positive integer. Mandatory. Possible values:

- 0 – DRCE.

"data" – jason with state information returned from execution environment (DRCE Cluster), string. Mandatory. Structure depends on execution environment engine.

Get status of the task

"type" – execution environment type, positive integer. Mandatory. Possible values:

- 0 – DRCE.

"state" – task state bit set mask, positive integer. Mandatory. Possible bit values:

- 0 – queued,
- 1 – data stored in the storage,
- 2 – scheduled,
- 3 – run,
- 4 – finished,
- 5 – results data fetched,
- 6 – results data deleted.

"cdate" – task creation date, string format “YY-m-d H:i:s”. Optional.
"sdate" – task scheduled date, string format “YY-m-d H:i:s”. Optional.

"rdate" – task run date, string format "YY-m-d H:i:s". Optional.
"fdate" – task finish date, string format "YY-m-d H:i:s". Optional.
"time" – real time that was used by task, positive integer, msec. Optional.
"tries" – real number of tries to run the task, positive integer. Optional.
"fetched" – real number of requests for results data to fetch, positive integer. Optional.
"finish" – task termination state, positive integer. Optional. Possible values:

- 0 – finished regular way,
- 1 – deleted by user request,
- 2 – crashed by itself internal error,
- 3 – killed by the execution environment.

"strategy" – fields array of strategy of planning, the same as request new task. Optional.
"data" – jason with task's results data returned from execution environment, string. Optional. Structure depends on execution environment engine.

Fetch task result data

"type" – execution environment type, positive integer. Mandatory. Possible values:

- 0 – DRCE.

"data" – jason with task's results data returned from execution environment, string. Mandatory. Structure depends on execution environment engine.

Delete/terminate task

"type" – execution environment type, positive integer. Mandatory. Possible values:

- 0 – DRCE.

"status" – status of the operation in the execution environment. Mandatory. Possible values:

- 0 – terminated.
- 1 – is not terminated.

"data" – jason with execution environment specific information, string. Optional. Structure depends on execution environment engine.

Error codes

DTM Operation	Error range	Error code	Description
NewTask		1 –	
CheckState		1 –	
CheckStatus		1 –	
FetchResultsFromEE		1 –	
FetchResultsFromCache		1 –	
DeleteTask		1 –	

DBI operation	Error range	Error code	Description
	1000 - 1100	0	Successful
		1001	Insert operation failure: key not uniq
		1002	fetch operation failure
		1003	update operation failure
		1004	delete operation failure
		1005	sql operation failure