INFORE Optimizer

Maintainer: George Stamatakis
Organization: ATHENA R.C.
Contact: giorgoshstam@gmail.com
Repository: https://bitbucket.org/infore_research_project/optimizer
Version: 1.0

Overview

This component consists of multiple services that facilitate a fast and easy way to optimize the operator placement of a workflow based on a list of available resources and rules. To perform this operation a number of files need to be submitted. To start with, a list of available clusters (also referred to as sites) needs to be provided along with the available BigData platforms (eg. Spark,Flink,...) on each site, this information must be present in a **network** file. Furthermore, a **dictionary** file is also required which contains a list of supported operators per platform and some default cost values to help ease cold starts and avoid operator placements in platforms that don't support them. A list of platform agnostic operators that describe a process can be described in a **workflow** file in which a number of RapidMIner operators and operator connections is described. All previously mentioned files can be submitted to the optimizer component at any time and they will be saved to a persistent storage service until they are requested by the optimizer service.

The goal of the optimizer component is to transform an unoptimized workflow to an optimized one which in turn requires a list of necessary rules and resources. Although the resources are collected and maintained by various optimizer services the user (eg. the RapidMiner framework) needs to provide a set of rules under which an input workflow can be processed. The set of required parameters and rules can be described in a **request** file which will trigger the start of the optimization process and will attempt to place the workflow operators to specific sites and platforms. The result of the optimization process, also known as an optimizer **response**, will be sent back to the user who initiated this process and a copy of that response will also be saved to a persistent storage. Finally, all previously mentioned JSON files that are either consumed or produced by the optimizer component are always saved to a persistent storage (currently Elasticsearch) and can be viewed by the Kibana service.

To sum up, the Optimizer component is responsible for

- Collecting, validating and storing files necessary for the optimization process.
- Aggregating and maintaining statistics in order to estimate various costs.
- Visualizing results and metrics.

File structure

The optimizer service uses a REST API and Websocket interface in order to accept and respond to user queries. All of the JSON files contain unique identification fields (see below) and the optimizer will insert or update these JSONs in a persistent key-value store using the unique fields as keys.

File	Unique ID field
Network	network
Dictionary	dictionaryName
Workflow	workflowName
Opt. Request	id
Opt. Response	id_{timestamp}

Note: The timestamp field is auto-generated by the optimizer at runtime.

- **Network**

  Information regarding the available clusters and platform topology should be placed in this file.

- **Dictionary**

  Operators-specific settings and operator implementations should be placed here.

- **Workflow**

  Abstract representation of operators, operator connections and already placed operators should be defined here.

- **Optimizer Request**

  A list of optimization instructions and previously submitted resources (e.g network file name) and requests that the optimizer service return a valid optimized workflow under certain constraints.

Algorithm	Value of request field ‘algorithm’
A* (fast but with large memory footprint)	op-A*
Exhaustive (slow but optimal)	op-ES
Greedy (Fast but approximate solutions)	op-GS
Automatic (Recommended)	auto

Note: The automatic option selection allows the optimizer to select the best suited algorithm (or a mix of algorithms) for the workflow at runtime and is therefore recommended.

- ## Optimizer Response

  An optimized workflow (with placed operators) along with relevant metadata. Since the response is produced by the optimizer at runtime a unique ID is created dynamically based on the input request ID, more specifically the current timestamp is appended after the request ID and an underscore. E.g. request1 -> request1_1596097800

The previous files can reference another file by using their unique ID field, e.g. an optimizer request can reference a previously submitted network by using its name. More specifically, the first 3 files (network, dictionary and workflow) can be submitted at any time and under multiple unique IDs and the optimizer service will only insert/update these JSON files in a persistent storage. Only an optimization request, that references 3 of these files, will trigger an optimization process which will have the optimizer retrieve the necessary files and attempt to place the workflow operators accordingly.

For security reasons an authentication layer has been added to the optimizer service. Basic authentication credentials need to be provided every time a request is made, more specifically a simple set of username/password credentials needs to be provided via an Authorization header on every request. Since the optimizer service has a public IP and allows for file uploads and downloads it’s necessary to add a form of authentication in order to prevent potential security problems. The credentials are stored in-memory (service side) and at this time no user sessionization is supported, this may change in the future. The credentials can be found at the end of this document and in the README file of the optimizer’s repository.

File submission and REST API

The optimizer service provides various services under different URLs, the routes of which are explained in this section. The Http status code of each request is always set to OK [200] unless an error occurred or a resource wasn't available.

Fille submission

Sending files to the optimizer service requires that the user simply PUTs the contents of the respective JSON file under the 'resource' path. The Content-Type header must be set to application/json since the REST API consumes JSON files. The response body value is set to 'Created' when the resource's unique ID is not present in the Component's persistent storage and to 'Updated if it already exists. (Previous submission was at {LocalDateTime of resource creation})'.

Route	**Method**	**Body**	**Response Body**	**Response Status**
/resource/network	PUT	@network	Created/Updated..	CREATED [201]
/resource/dictionary	PUT	@dictionary	Created/Updated..	CREATED [201]
/resource/workflow	PUT	@workflow	Created/Updated..	CREATED [201]

Note: The @resource refers to the contents of the JSON structure, usually a file (curl style). An exception is thrown if the JSON input is invalid (eg. wrong format, missing fields)..

File deletion

In order to permanently delete a file from the optimizer the user can use the following paths.

Route	**Method**	**Body**	**Response Body**	**Response Status**
/resource/network/{network_name}	DELETE	<empty>	<empty>	NO_CONETNT [204]
/resource/dictionary/{dictionary_name}	DELETE	<empty>	<empty>	NO_CONETNT [204]
/resource/workflow/{workflow_name}	DELETE	<empty>	<empty>	NO_CONETNT [204]
/resource/request/{request_id}	DELETE	<empty>	<empty>	NO_CONETNT [204]
/resource/response/{response_id}	DELETE	<empty>	<empty>	NO_CONETNT [204]

Note: The {unique_id} field refers to the unique ID of each JSON file (see previous section). An exception is thrown if the unique ID is invalid and a 404 error code is returned.

There is also a way to delete all files of a specific type by accessing the admin subdomain.
Use with caution!

Route	Method	Body	Response Body	Response Status
/admin/drop_networks	GET	<empty>	All networks were deleted.	OK [200]
/admin/drop_dictionaries	GET	<empty>	All dictionaries were deleted.	OK [200]
/admin/drop_workflows	GET	<empty>	All workflows were deleted.	OK [200]
/admin/drop_requests	GET	<empty>	All optimizer requests were deleted.	OK [200]
/admin/drop_results	GET	<empty>	All optimizer results were deleted.	OK [200]
/admin/drop_all	GET	<empty>	<All 5 previous messages>	OK [200]

File retrieval

In order to retrieve previously submitted files, or to simply check file contents from your browser, simply perform a GET request under the correct path. Additionally, there is also an option to retrieve specific files by providing their unique ID as a URL suffix. Note that a NOT_FOUND [404] error code is returned if a file with the provided unique ID isn't found.

Route	Method	Body	Response Body	Response Status
/info/networks	GET	<empty>	A list of all submitted networks.	OK [200]
/info/dictionaries	GET	<empty>	A list of all submitted dictionaries.	OK [200]
/info/workflows	GET	<empty>	A list of all submitted workflows.	OK [200]
/info/requests	GET	<empty>	A list of all submitted requests.	OK [200]
/info/responses	GET	<empty>	A list of all submitted responses.	OK [200]
/info/network/{unique_id}	GET	<empty>	Contents of network with {unique_id}.	OK [200]
/info/dictionary/{unique_id}	GET	<empty>	Contents of dictionary with {unique_id}.	OK [200]
/info/workflow/{unique_id}	GET	<empty>	Contents of workflow with {unique_id}.	OK [200]
/info/request/{unique_id}	GET	<empty>	Contents of request with {unique_id}.	OK [200]
/info/response/{unique_id}	GET	<empty>	Contents of response with {unique_id}.	OK [200]

STOMP Websocket interface

The STOMP protocol over Websockets is used by the Optimizer service in order to exchange messages with clients that want to submit workflows. This enables the optimizer to decouple client submissions from the optimization process and is necessary for longer running optimization processes which can take a significant amount of time. This interface also makes it easy for our service to both broadcast messages (e.g notifications) and send direct messages to clients or groups of clients. It's worth noting that messages in topics with the '/usr/queue' prefix are only sent to individual subscribers directly and are not broadcast to all subscribers. The optimizer's STOMP **endpoint** is currently '/optimizer' and can be accessed by adding that suffix to the service's IP (e.g ws://IP:PORT/optimizer)

The Optimizer offers the following **subscriptions**.

Topic	**Functionality**
/user/queue/control	Optimization process status / info.
/user/queue/echo	Echo utility, for debugging purposes.
/user/queue/plans	Optimized workflow(s).
/user/queue/errors	Error notifications and exceptions.
/topic/broadcast	Published messages are broadcast to all subscribers.

The Optimizer allows clients to send messages in the following **destinations.**

Dest. Topic	**Request payload**
/app/echo	<value>
/app/optimize	Optimization Request (JSON)
/app/cancel	Optimization Request ID

Optimization request and response flow

In order to perform an optimization request the Network, Dictionary and Workflow files need to be present in the Optimizer's persistent storage. These files can be submitted via the REST

APIs mentioned in the previous sections. In order to submit an optimization request the client needs to do the following:

1. Subscribe to '/user/queue/control' topic and monitor it's output since the Optimization Request ID and cancel status (if necessary) will be published here.
2. Subscribe to '/user/queue/plans' since all optimization results will be published there.
3. Subscribe to ' /user/queue/errors' since all errors regarding content submission and the optimization process will be published there.
4. Submit an optimization request by sending a message to the '/app/optimize' destination.
5. Monitor the control topic to get the request ID of the submitted workflow.
6. Monitor the plans topic for the optimization result.
7. If the submitted workflow falls under the continuous optimization category then the optimizer will keep publishing new optimization results in the plans topic until it's canceled. If the workflow does not fall under the continuous optimization category then the optimization process will simply publish the optimization result in the plans topic only once and then stop after this step.
8. In order to cancel a continuous optimization process simply send the request ID to the '/app/cancel' topic and check the control topic for an 'ACK' message. We are currently enforcing a hard limit of 10 continuous workflow optimizations per optimization request to avoid wasting resources.

A few notes regarding the optimization flow.

- The error topic messages are in JSON format with two fields, "error" and a "requestId".
- The optimization responses in the continuous mode have different IDs even when optimizing the same workflow. This is done to ensure that all optimization results can be mapped to a unique Elasticsearch document and ease debugging.

Service utilities

A few utility methods to test connectivity and perform health checks.

Route	**Method**	**Req. Body**	**Resp. Body**	**Response Status**
/info/echo/{value}	GET	<empty>	{value}	OK [200]
/info/logs	GET	<empty>	The entire log4j file of the service.	OK [200]
/info/time	GET	<empty>	Server time with Athen time zone.	OK [200]

Note: Log files of the optimizer are collected from a Dockerized environment and can sometimes be unavailable. An application/octet-stream object is returned.

Viewing files

To view submitted files visit the Kibana interface by clicking on the following line (replace localhost with the server's IP).

http://localhost:5601/app/kibana#/discover

The Kibana interface allows for concise and beautiful visualizations of the submitted data but unfortunately it's still a work in progress.

Host details and user credentials

A list of all deployed services that are part of the optimizer component. Since all of the following services are dockerized, port conflicts can be easily solved so feel free to contact us if another framework needs a port of ours.

Host IP: 45.10.26.123

Service	Optimizer	Elasticsearch	Logstash	Kibana
Port	8080	9200,9300	9600,5000,60001	5601

Since the optimizer service has been set up as a service and can be accessed by everyone that has its IP, a set of basic authentication credentials is required to submit and retrieve files. We should also consider user sessionization should the need for a multi-user environment arise. Although the user needs to simply include the BasicAuthentication header when making requests, the optimizer tests (as described in the next session) provide comprehensive examples and code snippets to make the authentication process easier to understand.

Service	**Username**	**Password**
Optimizer REST / WS API	infore_user	infore_pass
ELK Web UI / REST API	elastic	elastic123

Unit testing and examples

The necessary files to successfully submit and retrieve results for the life science and maritime workflows are available online in the parent folder of this document under the name demo and inside the /input path in the online bitbucket repository. There are JUnit5 tests available in the optimizer's git repository in package 'src/test/java/optimizer/OptimizationTests' which use these files to showcase optimizer's features such as file CRUD methods, authorization, expected file format and more.

Licence

The optimizer component uses the AGPLv3 license which can be found in the link below.
GNU Affero General Public License

Contact

For any questions, clarifications or bug reports related to the optimizer component feel free to open a JIRA ticket, PR or contact George directly at giorgoshstam@gmail.com.