Grid Computing API

Marcelo da Silva Reis [1]
Rafael José Peres Correia [1]

[1] Instituto de Matemática e Estatística, Universidade de São Paulo.
`marcelo.reis@gmail.com`, `rafaeljpc@gmail.com`

March 27th, 2012

1 Abstract

This article describes Grid computing API that provides access to the InteGrade middleware developed by the Distributed Systems Research Group at Institute of Mathematics and Statistics, University of São Paulo [1].

2 Application features

2.1 Submitting an application

If an application is successfully created, then its respective ID is returned at the response status.

HTTP Method	*URI*	*Request Content Sample*	*Responses*

POST	/applications	`<application>`
		 `<applicationName>`
		  `Blast_N`
		 `</applicationName>`
		 `<basePath> dir </basePath>`
		`</application>`
			`201 CREATED`
			
			`500 ERROR`

2.2 Submitting an application binary

This operation uploads an application binary, given an application ID and a system architecture (*Linux_i686* for 32-bits Linux environment, *Linux_x86_64* for 64-bits Linux environment, and *Darwin_Power* Macintosh for Power Mac environment).

HTTP Method	*URI*	*Request Content Sample*	*Responses*

POST	/application/{id}/binaries	`<binary>`
		 `<architecture>`
		  `Linux_i686`
		 `</architecture>`
		 `<file>`
		  `(application binary)`
		 `</file>`
		`</binary>`
			`200 OK`
			
			`401 NOT FOUND`
			
			`500 ERROR`

2.3 Getting application details

This method works almost like returning what was sent to create the application, except that it is included the architectures of the binaries uploaded by the user.

HTTP Method	*URI*	*Request Content Sample*	*Responses*
GET	/application/{id}	-	`200 OK` `<application>` `<applicationId>` `7` `</applicationId>` `<applicationName>` `Blast_N` `</applicationName>` `<basePath> dir </basePath>` `<numberOfBinaries>` `1` `</numberOfBinaries>` `<binaries>` `<architecture>` `Linux_i686` `</architecture>` `...` `</binaries>` `</application>` `404 NOT FOUND` `500 ERROR`

2.4 Listing existing applications

This operation lists applications related to the term specified by the “query” value. If “query” is empty or missing, all applications are listed.

HTTP Method	*URI*	*Request Content Sample*	*Responses*
GET	/applications?query=Blast	-	`200 OK` `<applications>` `<application>` `<applicationId>` `7` `</applicationId>` `<applicationName>` `Blast_N` `</applicationName>` `<basePath> dir </basePath>` `<numberOfBinaries>` `1` `</numberOfBinaries>` `<binaries>` `<architecture>` `Linux_i686` `</architecture>` `...` `</binaries>` `</application>` `<application>` `...` `</application>` `</applications>` `500 ERROR`

2.5 Delete an existing application

HTTP Method	*URI*	*Request Content Sample*	*Responses*
DELETE	/application/{id}	-	`200 OK` `500 ERROR`

2.6 Execute an application

An application execution may be either sequential or parametric. In both of them “constraints”, “arguments”, and “preferences” fields are optional. The “storeStdout” and “storeStderr” fields are optional and have “False” as default value. The application type is also optional and has “Sequential” as default value. If an execution is successfully created, then its respective ID is returned at the response status.

Sequential application execution

HTTP Method	*URI*	*Request Content Sample*	*Responses*

POST	/executions	`<execution>`
		` <applicationId>`
		`  7`
		` </applicationId>`
		` <constraints>`
		`  freeRAM >= 1024`
		` </constraints>`
		` <preferences>`
		`  freeCPU >= 30`
		` </preferences>`
		` <applicationType>`
		`  Sequential`
		` </applicationType>`
		` <arguments>`
		`   -p blastn -i genes.ffn`
		` </arguments>`
		` <storeStdout>`
		`  False`
		` </storeStdout>`
		` <storeStderr>`
		`  True`
		` </storeStderr>`
		` <inputFiles>`
		`  <executionFile>`
		`   <fileName>`
		`    genes.ffn`
		`   </fileName>`
		`   <file>`
		`    (binary of input file)`
		`   </file>`
		`  <executionFile>`
		` </inputFiles>`
		` <outputFiles>`
		`  <executionFile>`
		`   <fileName>`
		`    genes.blastn`
		`   </fileName>`
		`  </executionFile>`
		` </outputFiles>`
		`</execution>`
			`201 CREATED`
			
			`500 ERROR`

Parametric application execution

The "numberOfCopies" field specifies how many copies has the parametric execution. For each parametric copy, it must be provided individual execution information within the "parametricCopies" field.

HTTP Method	*URI*	*Request Content Sample*	*Responses*

POST	/executions	`<execution>`
		` <applicationId>`
		`  1`
		` </applicationId>`
		` <applicationType>`
		`  Parametric`
		` </applicationType>`
		` <numberOfCopies>`
		`  7`
		` </numberOfCopies>`
		` <forceDifferentNodes>`
		`  True`
		` </forceDifferentNodes>`
		` <parametricCopies>`
		`  <parametricCopy>`
		`   <inputFileNames>`
		`    <executionFile>`
		`     <fileName>`
		`      gene1.fasta`
		`     </fileName>`
		`    </executionFile>`
		`   </inputFileNames>`
		`   <outputFileNames>`
		`    <executionFile>`
		`     <fileName>`
		`      gene1.blastn`
		`     </fileName>`
		`    </executionFile>`
		`   </outputFileNames>`
		`   <arguments>`
		`    -p blastn -i gen...`
		`   </arguments>`
		`  </parametricCopy>`
		`  <parametricCopy>`
		`     ....`
		`  </parametricCopy>`
		` </parametricCopies>`
		` <inputFiles>`
		`  <executionFile>`
		`   <fileName>`
		`    gene1.fasta`
		`   </fileName>`
		`   <file>`
		`    (binary of input file)`
		`   </file>`
		`  </executionFile>`
		`  <executionFile>`
		`     ....`
		`  <executionFile>`
		` </inputFiles>`
		`</execution>`
			`201 CREATED`
			
			`500 ERROR`

2.7 Getting execution details

All the information given in the execution creation will be returned, except binaries and there is also the status tag to inform if execution was finished, is still running, was refused, etc.

HTTP Method	*URI*	*Request Content Sample*	*Responses*

GET	/execution/{id}	`-`	`200 OK`
			`<execution>`
			` <applicationId>`
			`  ...`
			` <constraints>`
			`  ...`
			` <status>`
			`  FINISHED`
			` </status>`
			`</execution>`
			
			`404 NOT FOUND`
			
			`500 ERROR`

2.8 Listing existing executions

Existing executions can be retrieved in two different ways: one by listing executions of a specific application and another by listing all executions (if "applicationId" is not specified).

HTTP Method	*URI*	*Request Content Sample*	*Responses*

GET	/executions?query={appId}	`-`	`200 OK`
			`<executions>`
			` <execution>`
			`  <applicationId>`
			`   {appId}`
			`  </applicationId>`
			`  <executionId>`
			`   3`
			`  </executionId>`
			`  <status>`
			`   EXECUTING`
			`  </status>`
			` </execution>`
			` <execution>`
			`    ...`
			` </execution>`
			`</executions>`
			
			`404 NOT FOUND`
			
			`500 ERROR`

2.9 Getting execution results

HTTP Method	*URI*	*Request Content Sample*	*Responses*
GET	/execution/{id}/result	-	`200 OK` `<execution>` `<executionFile>` `<fileName>` `stdout` `</fileName>` `<file>` `(binary contents)` `</file>` `</executionFile>` `<executionFile>` `<fileName>` `result.txt` `</fileName>` `<file>` `(binary contents)` `</file>` `</executionFile>` `<executionFile>` `...` `</executionFile>` `</execution>` `404 NOT FOUND` `500 ERROR`

2.10 Cancel an execution

HTTP Method	*URI*	*Request Content Sample*	*Responses*
DELETE	/execution/{id}	-	`200 OK` `500 ERROR`

References

[1] Distributed Systems Research Group at IME-USP. The InteGrade middleware. `http://www.integrade.org.br`, 2012.