Ar2GeMS - HPC

by Péricles Lopes Machado (UFRGS)

“There are three great virtues of a programmer; **Laziness, Impatience** and **Hubris**

Laziness: The quality that makes you go to great effort to reduce overall energy expenditure. It makes you write labor-saving programs that other people will find useful and document what you wrote so you don't have to answer so many questions about it.

Impatience: The anger you feel when the computer is being lazy. This makes you write programs that don't just react to your needs, but actually anticipate them. Or at least pretend to.

Hubris: The quality that makes you write (and maintain) programs that other people won't want to say bad things about.”

- Larry Wall, creator of the Perl Language

The problem

Intro

- How to develop an API language-independent?
- How to expose algorithms (services) in a network (cloud)?
- How to avoid write complex codes related to network protocols, security and load balance?
- How to implement a robust and efficient HPC system?
- How to reuse an existing base code and improve the system performance without great changes?

The solution

Be lazy is cool

RESTAPI = WEB-SERVER + FASTCGI + JSON = Ar2GeMS HPC

[1] or other web server...

What’s REST API?

- REST API is an API developed using an architecture REST (Representational state transfer) to execute the specified operations.
- REST is architecture developed originally for web (a system massively distributed) to provide hypermedia content (like video stream, facebook comments, etc...).
- REST is a key concept when we want to develop a system based on microservices.
- REST API examples: google maps, facebook api, oAuth (used to authentication using third-party accounts), dropbox, google drive, etc…

What’s REST API?

This architecture is based on some constraints:

- **Uniform interface** : REST API is a function that receive a string (byte vector) and return a string using the same representation. For example:
  The client send a ‘{“cmd” : “sum”, “args”:{“a”:3,”b”:4} }’ to a “sum” server that implement a “sum” REST API. In this case, the server must to return the result using the same format (JSON): ‘{“result” : 7}’
- **Stateless**: the client request has all necessary information to the task execution. For example: Let suppose the user wants to compute the variogram of a property “x”, the request sent must be like ‘{“cmd”: “variogram”, “args”:{“property”: “property_name”, “params” : {params}}} ‘

What’s REST API?

- **Cacheable (be lazy is cool)**: The server or client or other intermediary can cache some results to avoid unnecessary computations. For example: If the user is asking the variogram to a simulated grid property that hadn’t any modification since the last request, then the server can use the cached varmap (or some intermediary representation) to generate the variogram or just sent a variogram previously generated.
- **Layered system**: The client doesn’t need know who is the end-point that will realize the real request computation. Generally, REST API are implemented using industrial web servers like nginx, apache, lighthttpd etc. to handle the requests (in this case, load balance, security, package, tcp, sockets, integrity and work redirecting are managed in the server layer). A third-party server can be used as entry-point to the REST API implementation. This is cool, because the API developer doesn’t need to worry about tricky problems related to robust and efficient servers development.

What’s REST API?

- **Code on demand**: The client can delegate to the server some responsibilities. For example, resume statistics generation can be computed in server side and the client just receive a processed information (the client can view the variogram cloud of a 1000 realizations of a grid with 1B nodes in a personal computer, cell phone or tablet).
- **Identification of resources**: The resources demanded are provided in the client request. For example,
  {“cmd”: “sgsim” , “params”: params, “results”: [“variogram_cloud”, “histogram_cloud”, “statistics”, “save_result”]}
  In this request ask to the server compute a simulations using the parameters provided by “params” and process the results using three functions “variogram_cloud”, “histogram_cloud”, “statistics”, after this save the results. This request contains all necessary data to execute the task.

What’s REST API?

- **Code on demand**: The client can delegate to the server some responsibilities. For example, resume statistics generation can be computed in server side and the client just receive a processed information (the client can view the variogram cloud of a 1000 realizations of a grid with 1B nodes in a personal computer, cell phone or tablet).
- **Identification of resources**: The resources demanded are provided in the client request. For example,
  {“cmd”: “sgsim” , “params”: params, “results”: [“variogram_cloud”, “histogram_cloud”, “statistics”, “save_result”]}
  In this request ask to the server compute a simulations using the parameters provided by “params” and process the results using three functions “variogram_cloud”, “histogram_cloud”, “statistics”, after this save the results. This request contains all necessary data to execute the task.

What’s REST API?

- Generally, REST APIs are implemented using HTTP protocol (but this is optional).
- A resource in a REST API can be identified via URIs (Uniform Resource Identifiers). For example: The user can access directly a variogram api using an URL like http://ar2gems.com/api/variogram, and pass the arguments via http POST (using an ajax json request for example). Or, using the curl command line, the user can do:

  ```
  $ curl -data ‘{“params” : {params}}’  http://ar2gems.com/api/variogram
  ```

- The REST API can generate virtual files or cache result in image format, text format and provide access to this resource via an URI like http://ar2gems.com/api/variogram/result/jd7wrqr34.png or http//ar2gems.com/api/variogram/result/djiq3wj35r.csv.
- The REST API can use database manager to store results, input data, cache intermediary data, share information between slave nodes, etc.

What’s REST API?

- The entry-point of the API can be just a proxy server like nginx that delegate the responsibility to compute the result to a slave node. For example, we can keep an entry-point in http://ar2gems.com that just redirect the request to a server with low workload or just create a ticket and enqueue the request in a job queue. The user can receive like request result just a ticket to verify the request execution status or retrieve the resultant data when the process is finished.
- NGINX, APACHE or other web server are responsables by network and http protocol details. The developer just need to worry in implementing the api end-points.
- The API end-point is a function **format_data execute_request(format_data args)**, i.e., a function that receive the arguments in a json format and returns a json with the result. Where format_data is a data representation scheme like JSON, PROTOBUFFER, XML or some binary format (but, it’s important to remember that using JSON we can access our REST from almost anywhere easily).

What’s JSON?

- JSON is an acronym to JavaScript Object Notation. This format has been developed originally to communication between web pages and server via AJAX (asynchronous JavaScript and XML). But now, JSON is replacing XML in many situations because is a data representation format more easy to be parsed. (it’s very simple to write a JSON parser just using a stack and a map, for example)
- There are many very efficient parser available in all main programming languages.
- The JSON generally generate a hash map with the json content, this is very useful when we want to reduce the request processing time.
- The size of package can be reduced significantly.
- It’s more readable than XML. Also, JSON is a natural representation to many languages like python and javascript.

What’s JSON?

- JSON example:

```
{
    “cmd” : “sgsim”,
    “args”:  {
        “params” : {
            "variogram" : variogram_obj,
            "n_simulations" : 1000,
            "grid" : {"name" : "simulations", "size": [10000, 10000, 10000], "type" : "cartesian_grid" }
            "data" : {"hard_data" : {"grid" : "grid_1", "property" : "new_prop"}, "soft_data" : {}}
        }
    },
    “results” : { "compute_variogram", "compute_histogram", "statistics"},
    "save_as" : { "base_name" : "my_simulation", "database" : "my_databse", "type" : "MySQL"}
}
```

What’s JSON?

- Using JSON in a cpp code:

```
#include <jsoncpp/json/json.h>
#include <iostream>
#include <string>

int main()
{
        std::string in;
        std::string line;

        while (std::getline(std::cin, line)) {
                in += std::move(line);
        }

        Json::Value json(in); // convert JSON to a hash map is a very efficient operation

        std::cout << "cmd" << json["cmd"] << "\n";
        std::cout << "args" << json["args"].toStyledString() << "\n";
        std::cout << "variogram" << json["args"]["variogram"].toStyledString() << "\n";

        return 0;
}
```

Why JSON?

- Tags aren't cool and are hard to parse. JSON uses just a pair of '{}', '()' or '[]' to store the values and built the object hierarchy, this is very fast and easy to process.
- JSON has a semantic very close to C++, C, Java and other c-like languages.
- It's more easy write a json in a curl command line…
- We can use ajax javascript to test our server end-point (We don't to worry in writing a client and lead to network details)
- Modern languages like javascript, Go, python, have native serialization methods to serialize data using JSON.

Why JSON?

“XML is crap. Really. There are no excuses. XML is nasty to parse for humans, and it's a disaster to parse even for computers. There's just no reason for that horrible crap to exist. “

- Torvalds, Linus.

What’s FastCGI?

- CGI (Common Gateway Interface) is a protocol to interface external application with web servers.
- Using CGI, a web server receive data packages via http, https or other protocol and redirect the input data to the standard input of the external application.
- In other words, using CGI, we just need to process the data received via stdin (in c++)! :-) We don’t need to worry about sockets and other tedious web server details! We just have to parse the data in stdin (generally, JSON or XML format data).
- The request result must be write via stdout.
- CGI is used by many web programming language to handle web request.
- FastCGI is an implementation of the CGI protocol that avoid initialize the external application every time.

What’s FastCGI?

- Essentially, a FastCGI library will redirect all POST data to the stdin of the external application, and the GET data, PUT and headers are storage in hash_maps. The library provide methods to access easily these informations.
- This is a “hello world” using CGI (the user just need to provide a header describing the content):

```
#include <stdio.h>
int main(void) {
  printf("Content-Type: text/plain;charset=us-ascii\n\n");
  printf("Hello world\n\n");
  return 0;
}
```

Why FastCGI?

- FastCGI is perfect to implement REST API end-points because we just need to worry in process the data provided via stdin or via GET hash_map (in plain CGI these extra infos are provide via argc and argv).
- All popular production web server provide fastcgi interface (NGINX, for example, has a very efficient implementation).
- It’s more easy to write robust applications servers when we delegate critical and complex jobs (like security management, load balance, protocol implementation,etc) to more mature tools like production web server.
- Be lazy is a virtue. :-)

Why FastCGI?

```
while (true) {
    accept_lock.lock();
    // FCGI accept isn't a thread safe operation
    bool ok = (FCGX_Accept_r(&request) == 0); // wait a new request (sleep while input stream is empty)
    accept_lock.unlock();

    if (!ok) break;

    fcgi_ostream out(request.out);
    fcgi_ostream err(request.err);
    fcgi_istream in(request.in);

    size_t clen = gstdin(in, err, &request, content); // read input request data from input stream

    ++count;

    json json_status;

    if (clen < STDIN_MAX && clen > 0) {
        json r = generate_request(content); // process the raw_data and generate the json request
        json_status = add_request(r); // execute the request and return an error, a ticket or a result
        json_status.update()["nreqs"] = count;
        json_status.update()["PID"] = static_cast<Json::Value::UInt64>(pid);
    } else {
        json_status.update()["error"] = "TRUNCATED";
        json_status.update()["max_size"] = static_cast<Json::Value::UInt64>(STDIN_MAX);
    }
    print_header(out); // print header to be sent to client
    print(out, json_status); //print json with the request result
    FCGX_Finish_r(&request);
}
```

Why to use FastCGI + JSON + Web Server?

- With few lines of code we can develop a robust and highly efficient distributed application. :-)
- Essentially, using FastCGI, we just need to worry about built an interface between ar2gems core and the REST API resources.
- The most important step in the design of a REST API is to define an expressive, simple and uniform query standard to all ar2gems core resources.
- Using this stack, we just need to write a set of functions that receive json requests and return json requests. The server doesn't need to worry about implementation details.
- Ar2gems HPC now is completely language and platform-independent.
- Microservices are elegants, robust, easily scalable and lean softwares (this saves computational resources and $$$ in cloud computing infrastructures).
- Using REST API we can split ar2gems hpc in many specialized microservices like

Why to use FastCGI + JSON + Web Server?

- Using REST API we can split ar2gems hpc in many specialized microservices like variogram services, statistics services, MPS services, estimation services, simulation services, etc.
- Microservices can be unified via REST API (uniform resource access).
- We can divide the workload along our cluster and cloud infrastructure in a transparent way (the user don’t need to about where the data is being processed).
- Details about database can be hided via a data source microservice.
- Slaves and masters server can use the same REST API to split the workload between themselves.
- Via REST API is very easy to write applications with the ar2gems core embeded.
- It’s more easy to create interface between data generated via GSLIB, ISATIS, DATAMINE and AR2GEMS.

Results and implementations

What did I implement?

- A new high level interface to the ar2gems_core called ar2gems_process:

```
class ar2gems_process {
public:
.
.
.
/* Other implementation details*/
.
.
.
    // These functions are used to run scripts and sgems commands
    // using directly the ar2gems core code inside a standalone application
    // In a first moment, I'm using theses function to redirect commands from
    // REST API to the ar2gems system.
    json run_script(const string& script);
    json run_script_from_file(const string& script);
    json run_cmd(const string& command);
    json run_cmd_from_file(const string& cmd);

.
.
.
/* Other implementation details*/
.
.
.

};
```

What did I implement?

- Python interfaces to the ar2gems_api and ar2gems_client (a rest api client without dependencies to the ar2gems core).
- A fastcgi entry-point to handle the requests.
- A process management to redirect the command to the appropriated handler.
- The ar2gems_server, the application responsible for waiting packages provided by web server (nginx, etc).
- An utilities library to handle json, base64, data compression and other details related to REST API input processing.
- Some unity tests.
- Configuration script to nignx.

How am I implementing?

- I’m using a design pattern based on new features from C++11 like move semantics, shared pointers, std::lambdas, std::threads and std::mutexes. Also, I’m trying to use only immutable and moveable objects (this saved a lot of memory and execution time). Using this methodology is possible to write a exception free code, lock-less and highly efficient.
- The process management is asynchronous, based on tickets and JSON-RPC protocol.
- This first implementation of the ar2gems_client uses long polling to check if there are available results.
- I’m trying don’t change the ar2gems main code in a first moment.
- I’m implementing all main primitives needed to handle ar2gems process, like grid management (create_grid, load_grid, save_grid, delete_grid), property management, algorithmics primitives (kriging, simulation algos, pre or post-processing algos, etc.), remote command or script invoking.

Ar2gems API examples

```
from pyar2gems import *


#THIS SCRIPT ISN'T thread-safe

#Don't use this script inside sgems!
#pyar2gems.Ar2gemsProcess.instance(True)

print "TESTING RUN COMMAND"
r1 = run_command('LoadProject  /home/gogo40/git/ar2gems-hpc/data/WalkerLake.prj')
r2 = run_command('RunGeostatAlgorithm  sgsim::/GeostatParamUtils/XML::<parameters>  <a
print "TESTING RUN COMMAND [OK]"

print "TESTING LOAD COMMAND FILE"
r3 = load_command_file('/home/gogo40/git/ar2gems-hpc/sgsim.sgems')
print "TESTING LOAD COMMAND FILE [OK]"

print "TESTING LOAD SCRIPT"
r4 = load_script('execute_cmd.py')
print "TESTING LOAD SCRIPT [OK]"

print "TESTING RUN SCRIPT"
script = get_file_as_string('execute_cmd.py')
r5 = run_script(script)
print "TESTING RUN SCRIPT[OK]"

print "ok!"
```

Ar2gems API examples

```
from ar2gems_hpc import *

from multiprocessing import Process
from multiprocessing import Queue
from multiprocessing import cpu_count

___N_RUNS___ = 100

def load_remote_project(host, min_sleep_time):
        # create ar2gems_client
        c = ar2gems_client(host);

        c.set_min_sleep_time(min_sleep_time)

        # Load project, this operation isn't thread safe
        r = c.run_remote_command('LoadProject  /home/gogo40/git/ar2gems-hpc/data/WalkerLake.prj')
```

Ar2gems API examples

- Also, there are C++ examples in these two links: https://github.com/ar2tech/ar2gems-hpc/blob/master/client/test_ar2gems_client.cpp and https://github.com/ar2tech/ar2gems-hpc/blob/master/ar2gems_api/test_ar2gems_api.cpp
- The ar2gems_api is a library used to implement the api end-points (the functions that will really execute the task)
- The end-point must be functions with the same signature (json foo(json)). All details to access the ar2gems resources or to implement the algorithm must be hidden.

TO-DO LIST

TO-DO LIST

- To improve the quality of the rest api, we need an high level api for ar2gems reources.
- For example, functions like geostat_grid create_grid(grid_name, prop), geostat_grid load_grid (grid_name, data_source), simulation_result sequential_simulation(params), etc.. would be very useful to write I high quality ar2gems code.
- We need to remove singleton dependencies in the ar2gems_core.
- We have to separate GUI related code and ar2gems primitives (like distribution generation, variogram calculation, neighborhood generation, etc..).
- Remove or overload functions that receive xml as input!
- Create input parameters class to algorithms and primitives.
- Currently, we are very tied with the xml param inputs, this isn't cool. We need a more "protocol agnostic api". For example

TO-DO LIST

- Currently, we are very tied to the xml param inputs, this isn't cool. We need a more "protocol agnostic api". For example, classes like this one would be very useful:

```
class variogram_param {
public:
        variogram_param(const xml& _xml) { init(_xml); }
        variogram_param(const json& _json) { init(_json); }
        bool is_valid();

        void set_structures(std::vector<structures>);

        json serialize();
        xml serialize_xml();
private:
        void init(const xml _xml);
        void init(const json _json);
};


...

json compute_variogram(json input) {
        function_2d result = std::move(compute_variogram(std::move(variogram_param(input))));
        return std::move(result.serialize());
}
...
```

TO-DO LIST

- A good idea is to add a new initialize method in geostat actions and algorithms to receive json objects provided by ar2gems_server. Also, if there was functions where we could to pass directly the needed params, this would improve the performance significantly, because we avoid the overhead to translate a protocol to other.
- An high level standalone api to ar2gems core would be very useful to write the REST API end-points and standalone applications.
- In a near future we can remove all processing from the ar2gems GUI! This would be very nice to the system stability, because a problematic plugin wouldn't turn down the GUI.
- To Implement all ar2gems primitives as microservices highly optimized (using cuda to algebraic operations, state-of-art algorithms etc).
- Remove all raw pointer from the code base and modernize the code to C++11.

TO-DO LIST

- Use more functional features like moveable objects, immutable objects, lambdas, std::threads.
- Remove depencies to Qt in ar2gems core… This is an unnecessary overhead. Boost can replace ellegantly Qt in many features.
- Separate completely the GUI from the core.
- Reimplemented the main algorithms to explore fully all computational provided by ar2gems hpc (via ar2gems_client, ar2gems_server and ar2gems_api).

Questions?

Thank You!