Simulating, Storing and Accessing TELEMAC Simulations with AWS Cloud Computing Technologies

Julien Cousineau
National Research Council Canada
Ottawa, Canada
julien.cousineau@nrc.ca

***Abstract*— The rise of cloud computing and on-demand availability of computer system resources brings new opportunities for TELEMAC users, potentially including big data analysis, data and process automation, and machine learning. The paper presents a new application programming interface (API) module to simulate, store and access TELEMAC simulations with Amazon Web Services (AWS) cloud computing technologies. One of the main goals of the API is to create a means for users to quickly create models, solve TELEMAC simulations, automate low level processes and easily share results with a wide range of users (e.g. scientists, engineers, stakeholders, and the public); thus enabling faster and more informed decision-making.**

I. INTRODUCTION

In the simplest terms, cloud computing services, such as AWS, can provide computational resources to support the typical workflow associated with TELEMAC hydrodynamic simulation: creating a steering file, storing the input files on a local/network drive and run the simulation on the local/network drive. This can be easily achieved using the AWS EC2 service which allows users to access computing resources and control operations on one or a cluster of virtual machines. StarCluster [1] is a good toolkit for creating and managing distributed computing clusters hosted on AWS EC2. This approach, however, does not offer functionality for users to quickly create models, solve TELEMAC simulations, automate low level processes and easily share results with a wide range of users. Without a suitable API to support integration, users must have knowledge of both TELEMAC and AWS environments (i.e. installation/setup, simulation processes, storage, credentials) and this makes the process difficult with a steep learning curve.

A new API, henceforth referred as AWS TELEMAC API (ATAPI), was develop with the TELEMAC TelApy module and AWS services. ATAPI simplifies the integration of TELEMAC simulation with AWS resources, alleviating the knowledge prerequisites of low level processes that would otherwise be required. The ATAPI allows users to focus their energy on the project at hand, enabling faster and more informed decision-making. ATAPI is part of a larger framework which includes data storage, data analysis/simulation and data visualization. Fig 1 shows how a user can interact with the framework and how ATAPI is intergraded in it. Although the user can use a website or API tools (e.g. Postman) to access ATAPI through services such as API Gateway and Lambda, only the direct access route with a python interface is presented in this paper.

ATAPI is divided into three main components:

- Storing input and output files for TELEMAC simulations;
- Creating simulation cases; and,
- Simulating TELEMAC models.

Each component uses different AWS services to achieve their goal: AWS S3 is used to store and access data; AWS DynamoDB is used to store scenarios and track simulation progress; AWS Batch and AWS EC2 are used to simulate the cases. This paper describes the methodology for each of these components; an example is also presented to showcase the process and technology. Fig 2 shows the architecture diagram of ATAPI.

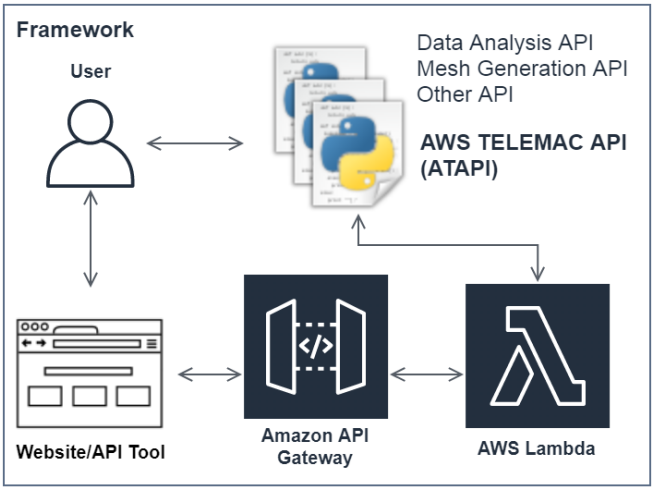


Fig 1: User interaction with ATAPI and other API services in the framework. A user can interact with ATAPI using a python interface or a website/API tool

II. STORING AND ACCESSING DATA

Storing and accessing data is an important component since decision-making is often based on data and data visualization. With the exception of steering files (.cas), all

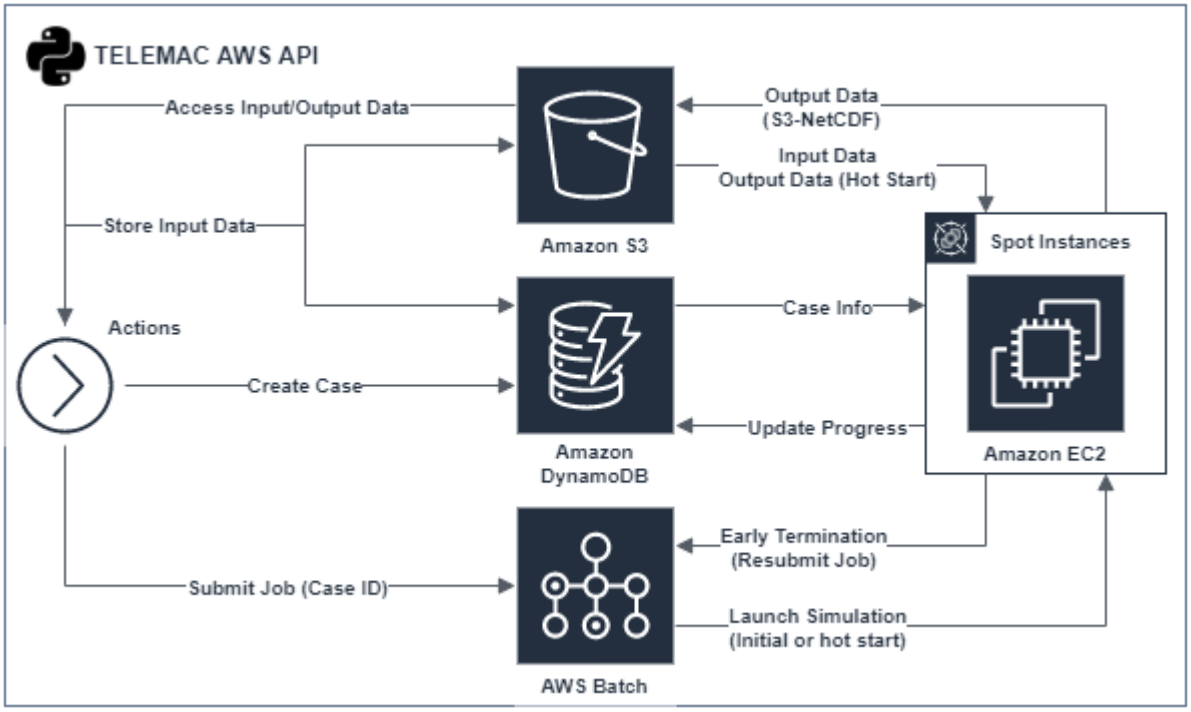


Fig 2: The architecture diagram of ATAPI. It allows storage of, and access to, data using AWS S3 and stores its metadata using AWS DynamoDB; create and store simulation cases, and track simulation progress using AWS DynamoDB; and, simulate cases using AWS Batch and AWS EC2.

input data (i.e. geometry file, boundary condition file) and result files are stored on Amazon Simple Storage Service or AWS S3: an object storage service that offers industry-leading scalability, data availability, security, and performance. Files are stored as objects in a specific S3 Bucket, which can be shared with the public or specific users. Amazon Elastic Block Store (EBS), another type of storage service, is used only temporarily during simulations with the virtual machines.

To increase efficiency in querying files in a S3 Bucket, metadata is created for each file and stored in AWS DynamoDB: a key-value and document database. The metadata consist of information such as name, file type, username, date created, date modified, and name of S3 Bucket.

Fig 3 shows an example of data and metadata stored in AWS S3 and AWS DynamoDB, respectively. The ATAPI currently saves files under the project name and under either *Input* or *Output* folders as shown in the figure. The input folder contains input files required to run TELEMAC (e.g. Geo_M2.slf, ocean.M2.cli) and the output folder contains the model output files. The only exception is the steering files (.cas). Steering information (i.e. keywords and values) is stored on AWS DynamoDB as shown in the architecture diagram (Fig 2) since metadata, such as simulation progress, date created, date modified, and created by, are also saved.

Typically, simulation results are saved in a results file: a binary Selafin file (.slf). However, most result files contain a few hundred Megabytes to Terabytes of data. Unfortunately, AWS S3 does not allow partial download of binary files and this makes accessibility and enquiries difficult. For example, extracting a time-series at one node (e.g. less than 1MB data) would require download of the entire results file from AWS S3. The solution for this is to create partition files that contain the model results using the S3-NetCDF package.

A. Model Results and S3-NetCDF

The S3-NetCDF package [2] was developed during the study to partition/split large array or TELEMAC results file into smaller file/object fragments for AWS S3. For example, instead of storing a one Terabyte Selafin file, the S3-NetCDF package can partition the file into thousands of files, each containing a slice of the data. This makes accessibility and enquires more efficient and less expensive for AWS S3.

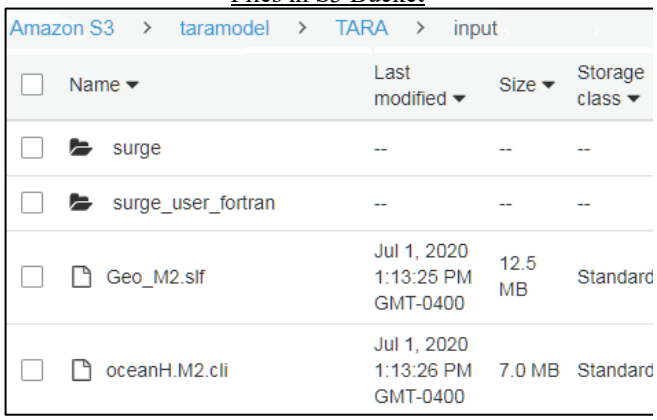
Packages such as S3-netcdf-python [3] already exist but are mainly targeted at climate data products discretized over a structured mesh. Therefore, existing packages are incompatible with TELEMAC simulations that use an unstructured mesh.

The S3-NetCDF package uses the NetCDF (Network Common Data Form) binary format file (.nc) instead of the Selafin file (.slf). NetCDF has become standardized and supported in the scientific community, and has better interoperability with other formats. Netcdf4-python [4] is a python wrapper and was used during development to create and read NetCDF files. NetCDF also has the capability of data compression, which is a big advantage, cost-wise, in storing large datasets on AWS S3.

The S3-NetCDF package works by creating a master file (.nca), also known as the header, and partition files (.nc). The master file contains the definitions of variables (i.e. name of

variable, units), dimensions (i.e. size of array), metadata, groups and partition indices. The partition file contains the data. The files are partitioned by group, by variable and then by temporal and spatial sub-domains. The size of the subdomains (# of temporal and spatial nodes) depends on the size requested by the user. By default, the size of each partition file is 10 MB.

Files in S3 Bucket



File metadata in DynamoDB

id	createdAt	name	type	updatedAt
TARA/input/Geo_M2.slf	1593623...	Ge...	slf	15936236...
TARA/input/oceanH.M2.cli	1593623...	oc...	cli	15936236...
TARA/input/surge/rcp85.GFDL-ESM2I	1593623...	rcp...	slf	15936236...
TARA/input/surge/rcp85.GFDL-ESM2I	1593656...	rcp...	slf	15936569...
TARA/input/surge_user_fortran/bord.f	1593623...	bord	f	15936236...
TARA/output/rcp85.GFDL-ESM2M.20	1593646...	rcp...	nca	15936467...
TARA/output/rcp85.GFDL-ESM2M.20	1593656...	rcp...	nca	15936569...

Steering info and metadata in DynamoDB

id	iframe	keywords	module	nframe
TARA/rcp85.(	25921	{ "AIR PRESSUR...	telemac2d	25921
TARA/rcp85.(	26065	{ "AIR PRESSUR...	telemac2d	26065
TARA/rcp85.(	25921	{ "AIR PRESSUR...	telemac2d	25921
TARA/rcp85.(	25921	{ "AIR PRESSUR...	telemac2d	25921

Fig 3: Example of file, file metadata and steering info stored on AWS

Fig 4 shows an example of model results with different partitioning options. The master file contains three dimensions: number of nodes (nnode), number of elements (nelem) and number of frames (nframe). It also contains four groups and numerous variables that contain data. Each variable under a group has the same array shape. For example, every variable under Group A contains nnode values (1D array). Variables under Group C/D contain nframe by nnode values (2D array).

The datacubes in Fig 4.1-3 illustrate different ways that the data from Group C can be partitioned between temporal and spatial domains by varying the partitioning size. The partitioning size is by the user. In this example, the first cube is partitioned with a file size of 10 MB, which contains data for one frame and for all nodes. The second cube is partitioned with a file size of 20 MB and partition files contain the two frames with all nodes. The third cube is partitioned with a file size of 5MB and partition files contain data from 1 frame and half the nodes.

Group C is perfect for querying spatially mapped variables for a given frame since it only needs to download 1 file (2 files for the third cube). However, Group C data are not conducive to extracting time-series information as this requires each partition file to be read. The solution for this is to duplicate and transpose the data in Group D, as shown in Fig 4.4. Although the data are saved twice, it is relatively cheaper to store data than to download the entire dataset from AWS S3 for every enquiry.

III. Creating Steering Case

Steering case information (i.e. keywords and values), typically saved in steering files (.cas), is stored in AWS DynamoDB as an object (under the *keywords* column). The simulation progress, type of TELEMAC module (e.g. telemac2d, telemac3d) and other metadata are saved in the database. A database is easier to maintain and update compared to writing/reading a file from AWS S3.

Fig 3 shows an example of cases in AWS DynamoDB. The simulation progress is identified by nframe and iframe, which are the total number of frames in the simulation and current frame position in the simulation, respectively. The simulation is complete when iframe is equal to nframe.

There are two ways of saving steering information: using a steering file (.cas) or using a JSON file/object that contains keywords and values. All local input files in a steering file (e.g. GEOMETRY FILE) are automatically uploaded and the local paths are replaced by the S3 paths.

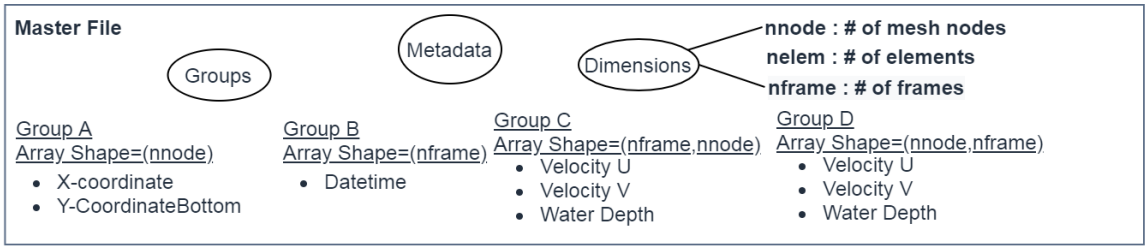
IV. Simulating Scenarios

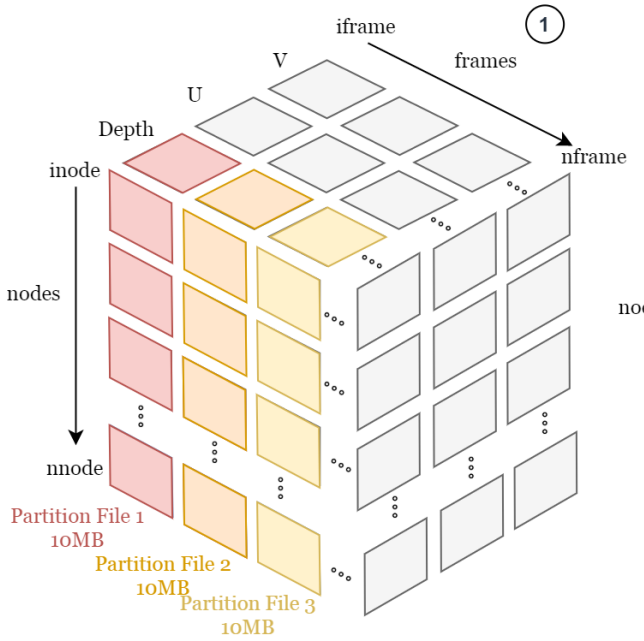
Simulations are executed using virtual machines with Amazon Elastic Compute Cloud (Amazon EC2). EC2 is an AWS service that provides secure, resizable compute capacity in the cloud. These virtual machines are used to run TELEMAC simulations and can be automatically scaled and sized based on the number of simulations using AWS Batch.

A. AWS Batch and EC2 Spot Instances

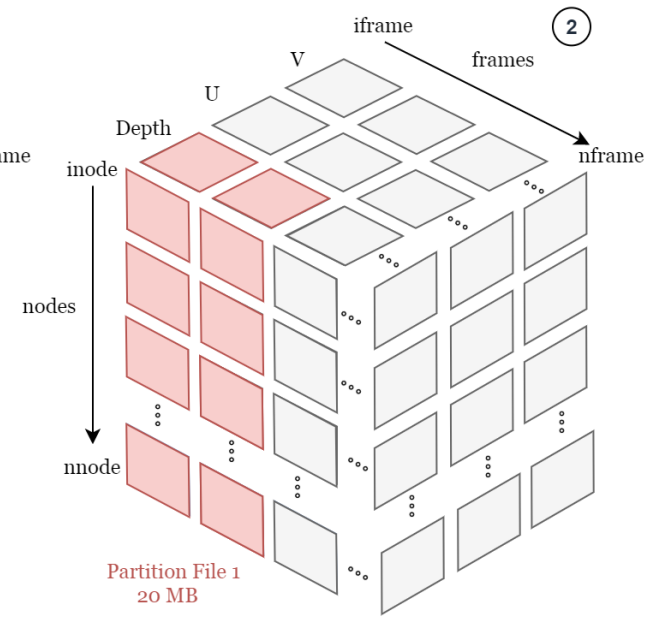
AWS Batch provides functionality to create computer environments, assign a job queue and specify the type of job. In this case, the job is running TELEMAC and saving model results to AWS S3. AWS Batch dynamically provisions the optimal quantity and type of compute resources (e.g., CPU or memory optimized instances) based on the volume and specific resource requirements of the batch jobs submitted.

During ATAPI development, the Amazon EC2 Spot Instances was used instead of EC2 On-Demand Instances. This allows the ATAPI to take full advantage of unused EC2 capacity in the AWS cloud. Spot Instances are available at up to a 90% discount compared to On-Demand prices.

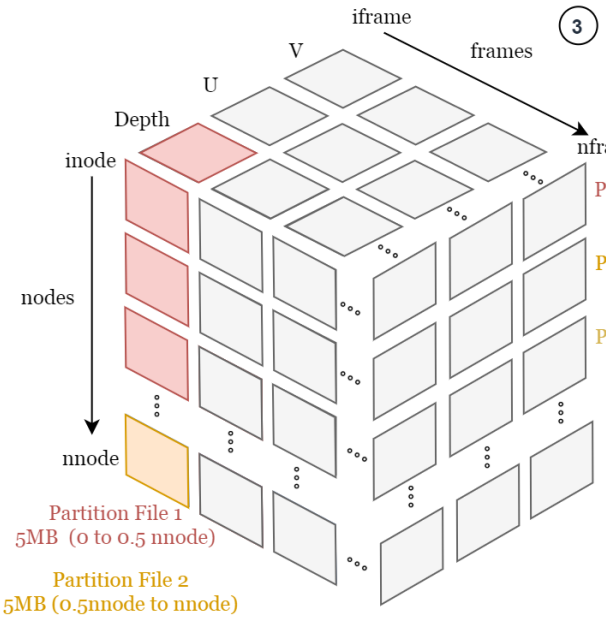




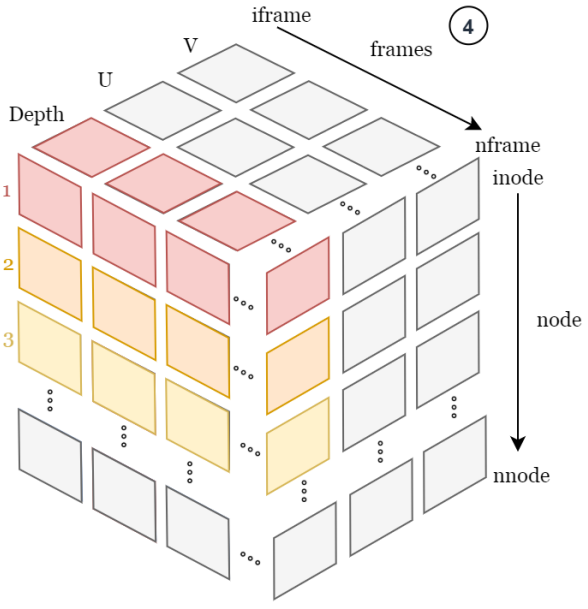
Datacube example for Group C using partition size of 10MB. This creates a partition file at each frame.



Datacube example for Group C using parition size of 20MB. This creates a partition file every 2 frames.



Datacube example for Group C using partition size of 5MB. This creates 2 partition files at each frame.



To extract time-series efficiently, the datacube from Group C is transpose and the data is stored in Group D. In this example, each partition file contains the time-series at each node.

Fig 4: Examples of different partitioning options based on the variables, temporal and spatial domains, and the size of the partitions.

The downside of using Spot Instances is early termination. Users can bid on spare Amazon EC2 instances to reduce computational costs but if the Spot price increases above the users bid price, then the Spot Instance can be terminated with two minute warning. This can be an issue for TELEMAC using the typical python launch scripts (i.e. telemac2d.py) since there's no communication between TELEMAC and AWS, and no way of determining early termination.

Amazon EC2 has different types of instances but the compute optimized series (C-series) is recommended for TELEMAC simulations since they are made for scientific computing. Most C-series use the latest Intel and AMD cores (e.g. 2nd generation Intel Xeon Scalable Processors – Cascade Lake). A cluster of virtual machines with AWS Batch for one simulation is currently not supported with ATAPI - it only runs on a single virtual machine (up to 48 cores) per simulation.

B. Job and Docker Container Image

Docker container images are used to run jobs on AWS Batch. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings. Multiple Docker container images can exist on the same virtual machine.

A Docker image was develop using ATAPI, TELEMAC V8P1R0 and its software dependencies on an Alpine 3.11 Linux template image. The Docker file, used to create the Docker image, is available in the GitHub repository [5]. When launching, the image automatically starts a simulation script from ATAPI.

It is important to note that the default Docker image container size is 22 GB and virtual machine size is 100 GB - multiple Docker containers can exist on a single virtual machine. The 22GB container size offers enough storage to contain software packages and scripts. However, large input files (>22 GB) can cause issues if the default container size is not changed.

C. Simulation Script and TELEMAC TelApy

The simulation script from ATAPI runs TELEMAC simulations with the help of TELEMAC TelApy [6] – it allows full control of the simulation while running a case. This allows users to:

- download input data from AWS
- create a new or continuing simulation steering file using case info from AWS DynamoDB
- Create a S3-netcdf output file or prepare a previous computation file
- prepare the study using TelApy
- simulate the case step-by-step
- save model results instantly with S3-NetCDF (Group C),
- save the simulation progress by changing iframe in DynamoDB; and,
- determine Spot Instances termination and resubmit the job, if necessary

The logical flow diagram of the script is shown in Fig 5.

Starting a new simulation case creates a new S3-NetCDF file automatically. By default, ATAPI creates the essential variables to continue a simulation. For example, for TELEMAC2d, it creates velocity u, velocity v and free surface elevation variables.

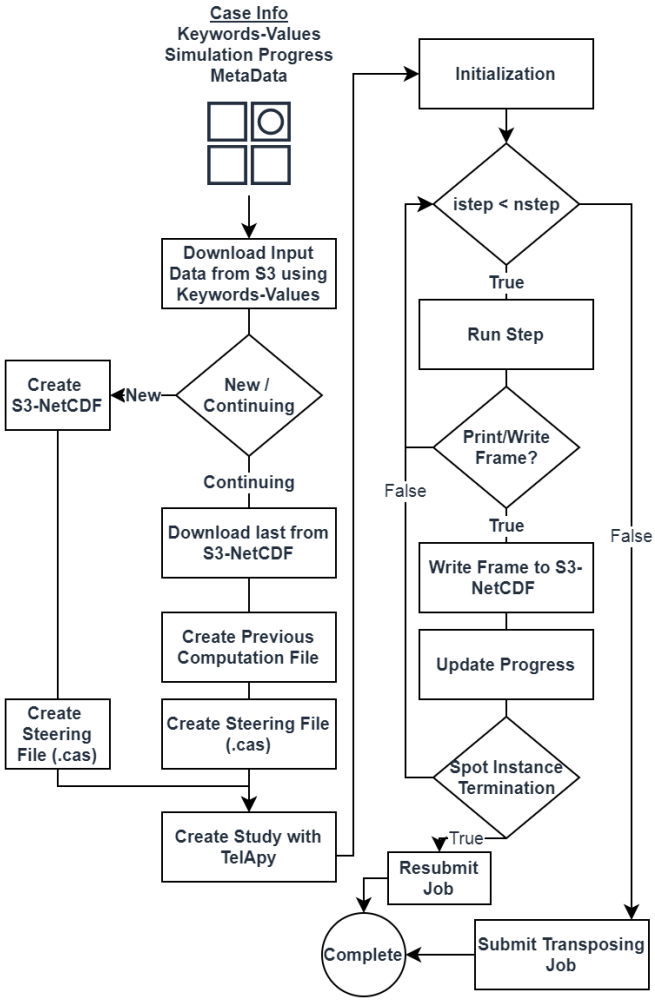


Fig 5 – Flow chart of the simulation script

The *nframe* is computed during the initialization step using the GRAPHIC PRINTOUT PERIOD keyword and the number of time steps (i.e. MODEL.NTIMESTEPS). The *iframe* value is updated every time simulation output data are saved to S3-NetCDF.

Cases with early termination are resubmitted to AWS Batch as a continuing job. Keywords such as PREVIOUS COMPUTATION FILE and COMPUTATION CONTINUED are added to the steering file. Data from the last saved frame are automatically downloaded from S3-NetCDF and are saved as the previous computation file.

To prevent simulated data from being unnecessarily written to the Docker container results file, the keyword NUMBER OF FIRST TIME STEP FOR GRAPHIC PRINTOUTS is specified equal to the number of time steps in the simulation. This prevents exceedance of the 22GB container limit.

Once the simulation is complete, it submits another job to transpose the dataset from Group C to Group D.

D. Transposing Script

ATAPI includes a short script to transpose the data array from Group C (nframe, nnode) to GroupD (nnode, nframe). This permits quick and efficient extraction of time-series information. As mentioned in the previous section, extraction of time-series information from Group C data would be computationally inefficient and expensive, as it would require download of every partition file. Fig 6 shows a flow chart diagram summarizing the script procedure. The memory-map functionality from the Numpy package was used to transpose the array. Numpy memory-map improves computational efficiency by allowing the data array to be stored as a binary file on the disk, as opposed to keeping the data in memory.

V. Application Example

ATAPI was used to simulate surge on the west coast of Canada using 3-hourly atmospheric and wind data from ECMWF ERA5. The model was used to simulate 35 scenarios from October to March from 1980 to 2015. Model and simulation characteristics are shown in Table 1.

Table 1: Model and simulation Characteristics of the example

Number of nodes (nnode)	352,464
Number of elements (nelement)	618,205
Number of scenarios	35
Number of frames (nframe)	25921 26065 - leap year
Input atmospheric and wind data @ 3hr	~ 6 GB per scenario
Output model results (U,V,FS) @ 10min	~110 GB per scenario

A. Create steering case info and storing input files

For this project, the atmospheric and wind data input files were created on a local machine. A steering file (.cas) was created for each scenario on a local machine and then uploaded using ATAPI. Each scenario case was created using the syntax shown below:

```
1.  from awstelemacapi import AwsTelemacApi
2.  atapi=AwsTelemacApi ()
3.  case1Info=atapi.upload('case1.cas')
4.  …
5.  Case35Info=atapi.upload('case35.cas')
```

AwsTelemacApi is the main class object in ATAPI. The class contains AWS information such as Dynamo Table Ids, S3 Buckets Id, local folder, number of cores during simulation,

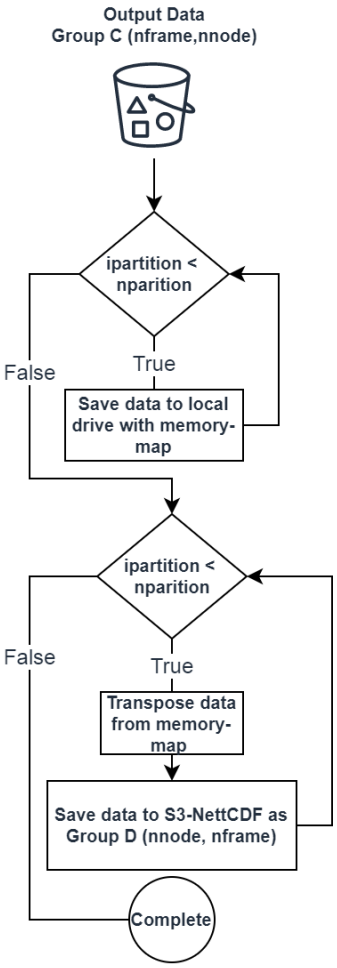


Fig 6: Flow chart of the transposing script

credentials, etc. Default values in the class can be changed by specifying keys-values as arguments (at line 2). The upload function provides functionality to read a steering file (.cas), convert the content to a steering object, check if local input files exist, upload input files to AWS S3, change the input paths in the steering object to S3 Path and upload the steering object to AWS DynamoDB.

B. Submit job and Simulate

The simulation jobs were submitted as shown below:

```
1.  atapi.submitJob("case1")
2.  …
3.  as atapi.submitJob("case35",cpu=24)
```

This syntax will send the jobs to the AWS batch queue. Once the environment is ready, it will run the simulation until completion. When calling the function, users must specify the case ID; no additional user-input is required. However, the number of CPUs employed can be changed by adding a key-value argument. The default number of CPUs is 48. If the simulation is already complete (i.e. *iframe* is equal to *nframe*), it will raise a warning.

For this example, the AWS account used during the runs had a limited access of 160 total CPUs. It took approximately 3 days to simulate all 35 scenarios with a price tag of approximately $200 CAD at a 60% discount. Had On-Demand Instances been employed, the cost would have been approximately $350CAD.

C. *Reading and Exporting*

There are multiple ways of reading simulation data from AWS S3. A few example commands to extract the data are presented below, including descriptive comments:

```
# Extract u @ frame 0 for all nodes
u=atapi.query("case1","u",frames=0)
 Extract u @ frame 1 and first 3 nodes
u=atapi.query("case1","u",frames=0,nodes=[0,1,2])
u=atapi.query("case1","u",frames=0,nodes=slice(0,3))
# Extract u time-series @ node 0
u=atapi.query("case1","u",nodes=0)
```

The code automatically selects the best group from which to extract data depending on the specifications of the enquiry. For example, the first 3 commands select data from Group C since the queries request data across a spatial range during a single time step. The fourth command selects data from Group D since the query requests data across a temporal range at a single location.

Data can be exported in a variety of formats including NetCDF, Selafin, CSV, JSON and Shapefiles. There are some limitations associated with specific formats, and these limitations will be raised by Python should they be encountered (e.g. cannot export spatial and temporal data in a CSV file). A few example commands to export data are presented below, including descriptive comments:

```
# Export to netCDf (default): frame 0 for all nodes
atapi.export("case1","u",frames=0)
# Export to Selafin
atapi.export("case1","u",frames=0,format="slf")
# Export to csv: node 0 for all frames
atapi.export("case1","u",nodes=0,format="csv")
```

VI. CONCLUDING REMARKS

This paper introduces the ATAPI, a new application programming interface that simplifies the integration of TELEMAC software and AWS resources. ATAPI is part of a framework that allows users to simulate, store and access TELEMAC simulations using AWS Cloud Computing Technologies. This capability permits automation of low level processes and allows users to easily share results with a wide range of users (e.g. scientists, engineers, stakeholders, and the public); thus enabling faster and more informed decision-making.

ATAPI was developed, built and tested on a Linux virtual machine using Python3. The API has only a few dependencies including:

- TELEMAC TelApy: Interact with TELEMAC simulations
- S3-NetCDF: read/write partition NetCDF files to S3
- SlfPy: Read/Write Selafin files
- BOTO3: Communicate with AWS services
- Numpy, Scipy and Matplolib: Scientific python packages

Documentation, installation, examples and tutorials (work in progress) are available on Github [5]. An AWS account is required to try ATAPI with the python interface.

ACKNOWLEDGEMENT

The development of ATAPI was indirectly funded through multiple projects including "A New Tool and Database of Storm Surges and Waves in British Columbia Coastal Waters for Assessing Climate Risks to Federal Transportation Infrastructure" and "Inventory and Assessment of Tidal Energy Resources near Northern Communities", financed by Transport Canada and POLAR Knowledge Canada, respectively. I thank Sean Ferguson from the National Research Council Canada for reviewing the manuscript and providing constructive feedback.

REFERENCES

[1] MIT, "StarCluster", 2020, http://web.mit.edu/stardev/cluster/.

[2] Cousineau, J, "s3-netcdf", 2020, GitHub repository https://github.com/meracan/s3-netcdf.

[3] Massey, Neil; Hassell, David; Lawrence, Bryan (2018): Semantic storage of climate data on object stores. Zenodo. Presentation. https://doi.org/10.5281/zenodo.2597531

[4] Unidata, "netcdf4-python", 2020, GitHub repository, https://unidata.github.io/netcdf4-python.

[5] Cousineau, J, "aws-telemac-api", 2020, GitHub repository https://github.com/meracan/aws-telemac-api.

[6] Goeury, Cédric; Audouin, Yoann; Zaoui, F.; Ata, Riadh; El Idrissi Essebtey, S.; Torossian, A.; Rouge, D. (2017): Interoperability applications of TELEMAC-MASCARET System. In: Dorfmann, Clemens; Zenz, Gerald (Hg.): Proceedings of the XXIVth TELEMAC-MASCARET User Conference, 17 to 20 October 2017, Graz University of Technology, Austria. Graz: Graz University of Technology. S. 57-64.