Prozium — A workflow manager dedicated for distributed and redundant computational infrastructure

***M.B Kursa*[1], *J.M. Kopeć*[1][2]**

The computational workflow management is an important issue especially in the cases where computations are time critical, and include many complex tasks. The excellent example of such computations is numerical weather prediction (NWP), which involves coordination of many tasks using variety of infrastructures, including major HPC. Moreover, it requires timely and unfailing delivery of results, what often requires handling of complex failover mechanisms.

Most software used for managing the NWP process requires user to define a graph of dependencies and a set of time constraints. Instead, we propose the Prozium system: lightweight execution platform for rich tasks, for which scheduling and coordination is handled by tasks no different than computational ones. We argue that such solution is more manageable, flexible and robust, still providing deep insight in the workflow state.

Keywords: workflow management, numerical weather prediction, HPC workflow.

Introduction

Numerical weather prediction (NWP) poses an interesting challenge in managing the computations workflow. Usually the process consists of various tasks that range from simple data interpolation to resource intensive NWP models or data assimilation programs. The complexity of the process itself is very relevant. However, the other aspect is that NWP is a process that provides essential data for multitude of end users. These data have to be delivered repeatedly in a timely and reliable manner since any failure to do so may put infrastructure or even human lives at risk. Therefore NWP is an interesting case of HPC computation workflow where both time constraints and reliability of the process are very essential.

This results in the need to properly manage the process in order to optimize both the use of the computational resources and manage the workflow reliability. Since many institutions around the world facing this challenge every day a variety of solutions have been implemented [3]. Some of the most notable ones are the ecFlow [2] used both in ECMWF and NCEP, Cylc used in UK Met Office and many other organizations and Rocoto used e.g., in NCEP. Other more custom solutions are also considered (e.g., [6]).

Despite the availability of multitude of solutions some unique features of the Interdisciplinary Centre for Mathematical and Numerical Modelling of the University of Warsaw (ICM UW) NWP division motivated us to develop new original software. First of all we are working with 3 very different NWP modelling software environments: COAMPS, Unified Model (UM) and WRF, all in both production and experimental setups. This variety calls for a very universal software that can manage all of those in a similar manner. Second, ICM UW is not an institution solely dedicated to forecast delivery, therefore both human and computational resources must be carefully managed, which calls for possibly automatic solutions. Finally, we aim to provide as high reliability as possible, mostly by utilising wide range of heterogeneous resources shared with other ICM enterprises for NWP failover.

The majority of available software dedicated for workflow management of NWP use two standard paradigms: the task dependency graph is managed directly by the software and their

[1]Interdisciplinary Centre of Mathematical and Computational Modelling, University of Warsaw
[2]Institute of Geophysics, Faculty of Physics, University of Warsaw

architecture is based on client-server (master-slave) assumption. We find both of them limiting; explicit dependency graph quickly gets incomprehensibly complex and so is hard to be maintained, while central server is an obvious single point of failure.

In the present paper we will introduce the Prozium software, with which we aim to introduce a new paradigm into the NWP workflow management; instead of managing trivial tasks with elaborate metadata in a complex framework, we propose simple distributed execution platform for arbitrary complex tasks, on its own providing only basic compatibility layer as well as anti-entropy and monitoring services.

1. Architecture

The architecture of Prozium is distributed by design allowing it to manage many computational resources simultaneously; those can be both individual machines or clusters managed by their internal job managers. Each one runs a Prozium daemon called agent; agents are the only software elements of Prozium, and are all homogeneous, there is no central coordinator. The configuration and the behaviour of the cluster is defined by the contents of a control repository, in our case, a simple Git repository. Each agent maintains as recent copy of the repository as possible; because of this, failure of the central repository only prevent reconfiguration of a cluster, but does not impact already defined tasks. Agents communicate using gossip protocol [4, 7]; this is because availability is more important that consistency in the meteo forecasting case — calculating certain forecast twice only costs computational time, while missing forecast causes severe disturbances to the services depending on it.

```
await First(
 run("A"),
 All(
  run("B"),
  run("C")))
run("D")
run("E")
await run("F")
await run("G")
```

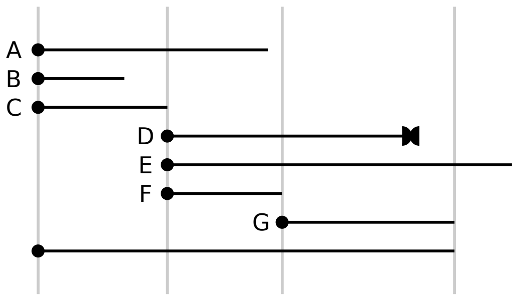

Figure 1. Example pseudocode of a task and its execution in time. Note that task D ended with an error, while task E continued execution after the completion of the master task.

The cluster works as follows. Agent may *run* a certain *task*; to this end it is given: a *reference* to a Git tree, i.e., a branch name or a commit id; *taks name*, a name of a file in the referenced tree holding code which will be executed as the task; *atom*, a string id of the task; *parameters*, a key-value object. Agent first consults the gossip state for a fact of a task of the same atom already running, either locally or within the cluster; if it is found, run fails. Otherwise, agent executes the code from the Git tree with given parameters and gossips this fact to the cluster. Most tasks are run from other tasks; on its own, agent only constantly re-runs task *main* from the *master* branch of the control repository, under random, globally unique atom and with an empty parameter object. Tasks can be also manually run by the operator, but this is discouraged.

Because the executed task code can only come form the control repository, user can easily identify not only which code produced certain artefact, but also the whole hierarchy of codes which lead to its execution. In the NWP nomenclature such organisation of workflow is called *suite*, and has been highlighted as an important feature of a reliable NWP system [3].

In Prozium, tasks are implemented in the ECMA-262 JavaScript, extended with async/await mechanism [1]. We believe this is a superior solution to using shell, as it is a more capable language and allows for easy and clear asynchronous programming, see fig. 1.

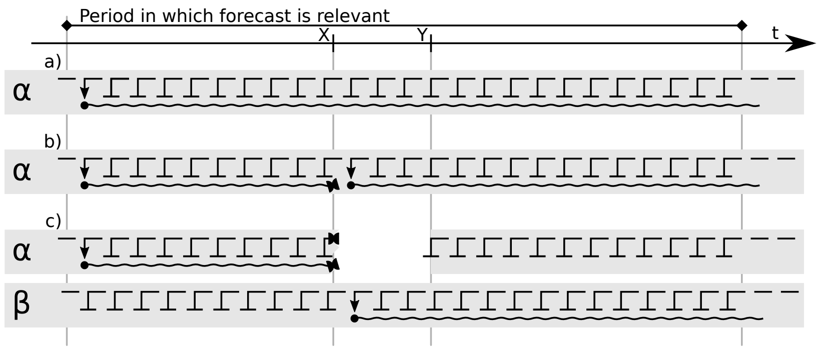

Figure 2. Example of execution of a simple forecast. a) When executed during the *a priori* defined time range, schedule task (solid line) tries to run the forecast task (weavy line) on each execution; yet this only succeeds once because only one task with a given atom can run at one time. b) In case the forecast task gets interrupted, it is re-run on the next run of a schedule task. While tasks are idempotent, actual forecast job will be continued rather than started from scratch. c) The atom uniqueness is enforced over the whole cluster, allowing for automatic failover between two agents, α and β.

The philosophy of Prozium requires tasks to be idempotent, so that they can be successfully re-executed in case of execution failure or agent failure, and partially re-entrant so that same task could run simultaneously under different atoms. As mentioned earlier, Prozium has no scheduling capabilities, and executing tasks is also a job of a task. An example of such set up is illustrated in fig. 2 where a dedicated schedule task is run repeatedly, for instance as a aforementioned master task. Whenever it is run, it checks the current time and consults the forecast running schedule implemented inside it for a list of forecast instances which should either be executing or have their results available for this time. Then, it simply runs a task for each of them, with an atom unique to the instance. Because agents enforce uniqueness of atoms of running jobs, this will end only in a single execution of a given forecast instance, or a re-run in case it fails.

One should not that even though the dependecy graph is not stated explicitly nor can be easily inferred from the tasks code, it can be constructed at run-time from the observation of task runs. This is why Prozium can report it via the monitoring module, similar to graph-based workflow managers.

2. Configuration of tasks for a simple ICM NWP suite

For the sake of simplicity we will present the simplest configuration of regional model, using the WRF-ARW model software [5]. One can divide its run into five steps: input gathering, pre-processing, model run, post-processing and results locking. We also have the operational time frame in which a model run is relevant, namely from the approximate time when the input data from external sources start to become available, till the end of the period for which forecast is calculated.

The input gathering task generates a list of files required by the model run and attempts to sync them from external locations; it also generates configuration files. Latter three steps make use of a generic task for idempotent executing program as a cluster job under slurm, which is used on most ICM UW clusters. In the pre-processing step three programs are run: geogrid.exe, ungrib.exe and metgrid.exe, first two in parallel. Certain possible errors are interpreted as a malformations of input files and delete them before failing the whole task. The following forecast run is realized by two consecutively run programs: real.exe and wrf.exe. Post-processing task starts in parallel with wrf.exe, hence can digest results in 1-hour chunks, as generated by the model. The final step simply sleeps till the end of the forecast relevancy period.

For robustness, the whole process is driven by a pair of Prozium agents running on two HPC clusters located in indpendent data centers.

3. Conclusions

We believe that the proposed Prozium workflow management system offers certain unique features useful for operational NWP. Rich tasks and control repository allows for increased flexibility and better reproducibility. The use of extended JavaScript for tasks' code makes it more comprehansible and easier to maintain. Finally the gossip-based clustering of agents allows workflow to be orchestrated over several redundant computational resources in a highly available manner.

References

1. *ECMAScript Async Functions proposal.* https://tc39.github.io/ecmascript-asyncawait/.

2. Avi Bahra. Managing work flows with ecFlow. *ECMWF Newsletter*, 129:30–32, 2011.

3. Ligia Bernardet and Laurie Carson. NITE - Numerical Weather Prediction Information Technology Environment. http://www.dtcenter.org/eval/NITE/NITE-report-AOP2014.pdf.

4. N. Hayashibara, X. Défago, R. Yared, and T. Katayama. The ϕ accrual failure detector. In *Proceedings of the IEEE Symposium on Reliable Distributed Systems*, pages 66–78, 2004.

5. J. Michalakes, J. Dudhia, D. Gill, T. Henderson, J. Klemp, W. Skamarock, and W. Wang. The weather reseach and forecast model: Software architecture and performance. In George Mozdzynski and Walter Zwieflhofer, editors, *Proceedings of the Eleventh ECMWF Workshop on the Use of High Performance Computing in Meteorology*, pages 156–168. World Scientific.

6. Samuel Trahan, T. Brown, S. Hsiao, B. Thomas, C. Holt, L. Bernardet, V. Tallapragada, H. L. Tolman, C. C. Magee, B. Kyger, and W. Lapenta. Modernizing the Operational Workflow and Automation of the NCEP Hurricane Weather Research and Forecast (HWRF) Modeling System using Python and Rocoto. In *Fifth Symposium on Advances in Modeling and Analysis Using Python*, 2015.

7. R. Van Renesse, D. Dumitriu, V. Gough, and C. Thomas. Efficient reconciliation and flow control for anti-entropy protocols. In *ACM International Conference Proceeding Series*, volume 341, 2009.