EUROFUSION WPISA-PR(16) 16561

M Owsiak et al.

Running simultaneous Kepler sessions for the parallelization of parametric scans and optimization studies applied to complex workflows

Preprint of Paper to be submitted for publication in
Journal of Scientific Computing

This work has been carried out within the framework of the EUROfusion Consortium and has received funding from the Euratom research and training programme 2014-2018 under grant agreement No 633053. The views and opinions expressed herein do not necessarily reflect those of the European Commission.

Running simultaneous Kepler sessions for the parallelization of parametric scans and optimization studies applied to complex workflows

Michal Owsiak[a,*], Marcin Plociennik[a], Bartek Palak[a], Tomasz Zok[a], Cedric Reux[b], Luc Di Gallo[b], Denis Kalupin[c], Thomas Johnson[d], Mireille Schneider[e]

[a] *Poznań Supercomputing and Networking Center IBCh PAS, Poznan, Poland*
[b] *CEA, IRFM, F-13108 Saint-Paul-lez-Durance, France*
[c] *EUROfusion Programme Management Unit, Boltzmannstr. 2, 85748 Garching, Germany*
[d] *Fusion Plasma Physics, EES, KTH, SE-10044 Stockholm, Sweden*
[e] *ITER Organization, St. Paul Lez Durance Cedex, France*

Abstract

In this paper we present an approach taken to run multiple Kepler sessions at the same time. This kind of execution is one of the requirements for Integrated Tokamak Modelling platform developed by the Nuclear Fusion community within the context of EUROFusion project[1]. The platform is unique and original: it entails the development of a comprehensive and completely generic tokamak simulator including both the physics and the machine, which can be applied for any fusion device. All components are linked inside workflows. This approach allows complex coupling of various algorithms while at the same time provides consistency. Workflows are composed of Kepler and Ptolemy II elements as well as set of the native libraries written in various languages (Fortran, C, C++). In addition to that, there are Python based components that are used for visualization of results as well as for pre/post processing. At the bottom of all these components there is a database layer that may vary between software releases, and require different version of access libraries. The community

*Corresponding author
Email addresses: michalo@man.poznan.pl (Michal Owsiak), marcinp@man.poznan.pl (Marcin Plociennik), bartek@man.poznan.pl (Bartek Palak), tzok@man.poznan.pl (Tomasz Zok), cedric.reux@cea.fr (Cedric Reux), luc.digallo@cea.fr (Luc Di Gallo), denis.kalupin@euro-fusion.org (Denis Kalupin), johnso@kth.se (Thomas Johnson), mireille.mchneider@iter.org (Mireille Schneider)

is using shared virtual research environment to prepare and execute workflows. All these constraints make running multiple Kepler sessions really challenging. However, ability to run numerous sessions in parallel is a must – to reduce computation time and to make it possible to run released codes while working with new software at the same time. In this paper we present our approach to solve this issue and examples that show its correctness.

Keywords: Kepler Project, workflows, parallel execution, Docker

Acknowledgement

This work has been carried out within the framework of the EUROfusion Consortium and has received funding from the Euratom research and training programme 2014-2018 under grant agreement No 633053. The views and opinions expressed herein do not necessarily reflect those of the European Commission.

1. Introduction

Integrated modelling efforts for ITER[2] experiment focuse currently on the assessment on the validation of comprehensive models against present facilities results. In view of supporting ITER operation, modelling tools are necessary, both for pulse validation and plasma control. The Nuclear Fusion community has developed for this purpose the software infrastructure framework for integrated modelling activities as well as a validated suite of simulation codes, platform that is currently operated under EUROFusion[1] project.

EUROfusion strategy includes the integration of the most advanced EU fusion codes into a centrally maintained suite of Integrated Modelling tools. Kepler plays major role in this development as it serves as a basis for integration of all the components and codes developed as part of the project.[2] The work

[1] `https://www.euro-fusion.org`

[2] `http://portal.efda-itm.eu/itm/portal/`

follow up the Integrated Tokamak Modelling Task Force that operated under EFDA from 2004 until 2013.

Within EUROFusion WPCD, Kepler is used as a basis for linking various components that are used in numerical computations. Each of these components can be developed in programming language that is supported by ITM (Integrated Tokamak Modelling) platform: C/C++, Fortran, Java, Matlab, Python. All these components (developed separately) are linked together to form workflows performing numerical computations. Different workflows focus on different aspects of plasma simulation. Thanks to using Kepler, each component (numerical code) can be wrapped by Java code and exposed as Kepler actor. This way, regardless of the workflow type, each actor (numerical code) can be easily reused without too much effort.

After workflows are released, there are two main ways of using them. First one, is to use released workflow for actual simulations, second one is to optimize it. These two actions require two, different, installations of Kepler. Both should be able to run at the same time. Apart from that, there is another requirement related to running multiple Kepler sessions at the same time, that is batch execution[3]. In order to reduce computation time one typically runs numerous Kepler sessions running the same workflow with different parameters (see Figure 1).

This way, it is possible to run multiple simulations at the same time. However, this is not an easy task to achieve when we talk about Kepler being run at batch nodes in multi user environment.

Current developments are influenced by numerous factors that affect execution of Kepler in parallel. These factors are either result of project's specifics or are based on internal Kepler's limitations. We will discuss these factors in next section.

We will also take a look at solution based on recent advances in virtualization, in particular Operating-system-level. Active development of such platforms, e.g. like Docker[4] container, made them an interesting solution for encapsulating applications. We will show benefits comming from this approach

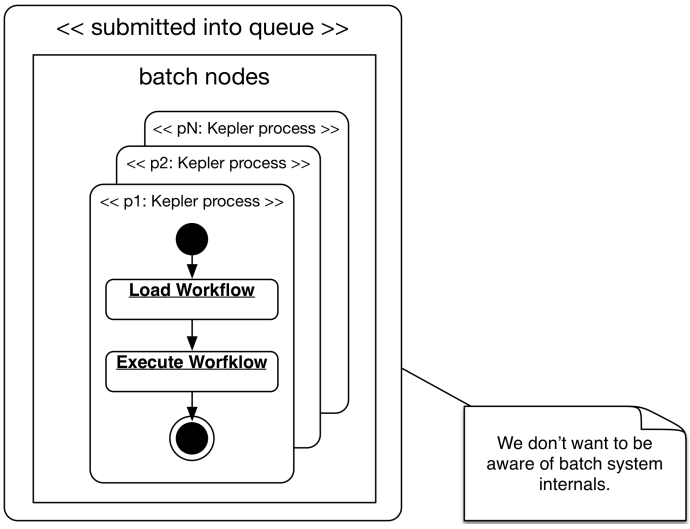

Figure 1: Running multiple Keplers in parallel

as well as some limitations. In our research we have focused on Docker based solution.

2. Limitations of Kepler's mechanisms

Kepler itself provides solutions for running multiple Kepler instances at the same time. However, these solutions are not fully applicable in case of EUROfusion based developments. We are dealing with very specific architecture where numerous components have their impact on workflow execution. In addition to that we work in multi-user and multi-configuration environment (different Kepler versions and different set of components executed by workflows).

There are four, most important factors having influence on parallel execution of Kepler:

- multi-user environment,
- different Kepler installations,
- version of modules being used,

- size of the workflows.

These factors are mostly related to Kepler's internal cache. Kepler's cache improves management of workflows and allows to reduce loading time of the workflow itself. Cached data are stored in the internal database that is stored for each user as a separate database file located inside user's home directory. Running multiple Kepler sessions at the same time means that each Kepler session should have its own means of accessing database file (cache). In case of HSQLDB[3] database (used by Kepler), it is not possible unless it is started in, so called, server mode[4]. This way, it is possible to run one database server and allow different Kepler instances access the data. It is fairly simple to use Kepler is this kind of execution mode, however, it is not suitable for multi user environments.

We will discuss these limitations in four, different contextes.

2.1. Multi user environment

In case of EUROfusion IM platform, all users work at the same front end machines of, so called, Gateway. It means, physically, all applications are run at the same machines. If we want to use HSQLDB in server mode[5,6], it means, each user has to have dedicated range of ports just for him.

This approach can be treated as a solution, however it is hard to maintain. It requires handling proper values of ports for all users (we want each user to have his own cache). In addition to that, it might be impossible to use this approach at some batch systems where security policy puts hard constraints on port ranges available for users. Eventually, we cannot allow sharing cache database due to number of workflows and configurations available. Users may use actors with different version numbers, they may use actors that use different database

[3] `http://hsqldb.org`

[4] `http://hsqldb.org/doc/2.0/guide/running-chapt.html#rgc_server_modes`

[5] `https://kepler-project.org/developers/reference/what-happens-when-kepler-starts-up`

[6] `https://kepler-project.org/developers/reference/accessing-hsql`

releases (with different structure) and, eventually, they may use completely different set of actors (different workflows). There is yet another issue with this approach. When started in server mode, it is not possible to run multiple instances of Kepler by single user. And that is exactly what we want to do.

2.2. Different Kepler installations

If we decide to use file based cache system, HSQLDB database is run in standalone mode, we do not have to allocate TCP ports for users. However, this approach triggers some issues as well. Development of workflows is a process. It means part of codes are ready for release (and can be used for simulations) while at the same time new codes are developed and added to platform. Users want to use Kepler in two flavours. Released version – for installations that contain stable codes, development version – that is still unstable and has to be tested. This leads to issues related to internal HSQLDB locking – see Figure 2.

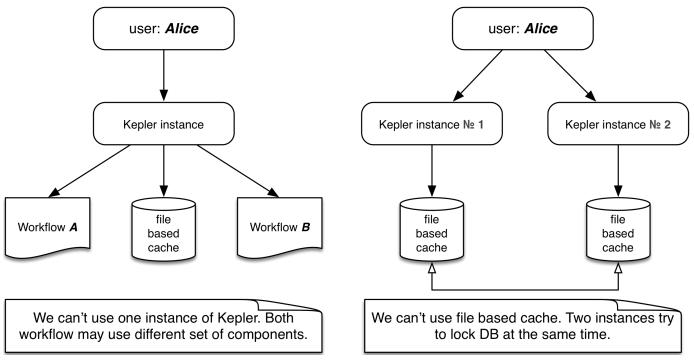

Figure 2: Different schemas for workflow execution

In this scenario, each Kepler instance tries to get exclusive access to cache. Clearly, only one instance will be able to achieve that. All other Kepler sessions will fail.

2.3. Different modules' versions

Yet another approach is to use single Kepler installation and load multiple workflows. In this scenario, there is a single Kepler instance that loads multiple

workflows. This scenario is depicted on Figure 2. However, this solution is not suitable for batch execution where Kepler is started in non-gui based mode. Another issue here is that in case of ETS (European Transport Solver) workflows (described in Section 4.1) we may run this workflow with different set of components.

2.4. Size of the workflow

It is possible to disable internal Kepler's cache at all (*-runwf -nocache -nogui*). This way, user is not forced to mangle Kepler's locations each time it is required to run Kepler in batch mode. However, there is yet another problem – size of the workflow. Workflow developed within EUROfusion are quite huge in size. They consist of thousands of components (actors, composite actors, connections, etc.). Typical ETS workflow consists of 460 composite actors and is nine levels deep. Parsing workflow file itself is quite time consuming task. In case Kepler is started without cache, we face huge increase of loading time (see Table 1)

ETS workflow (simplified)	**cache**	**no cache**
execution time [s]	120	420
loading time [s]	50	320

Table 1: Difference in execution/loading time with/without Kepler's cache

The reason for the difference here lays in parsing mechanism of workflow content. In case of executing Kepler without cache, it spends most of its time (while loading workflow) inside ***ptolemy.moml.MoMLParser._loadFileInContext()*** – calculations . In case of long lasting workflows this is not such an issue, in case of short computations, this time may heavily impact execution time.

3. Solution

To solve issues mentioned in section 2.1 we have decided to create artificial ***$HOME*** structure for each Kepler being run in batch queue. This solution is

depicted on Figure 3. This way, all required directories are replicated for each and every Kepler instance. These directories are: *.kepler*, *.ptolemyII*, *KeplerData*, *redirect* (used for redirecting non GUI based output). We also link *kepler* to installed Kepler version of user and we provide additional system information generated during job execution (*environment* – for all environment related information, *kepler.log* – logs generated by Kepler). For our specific case we also create directory *public*, it is used for data input/output. All workflow related data are stored inside this directory and are shared over all Kepler instances.

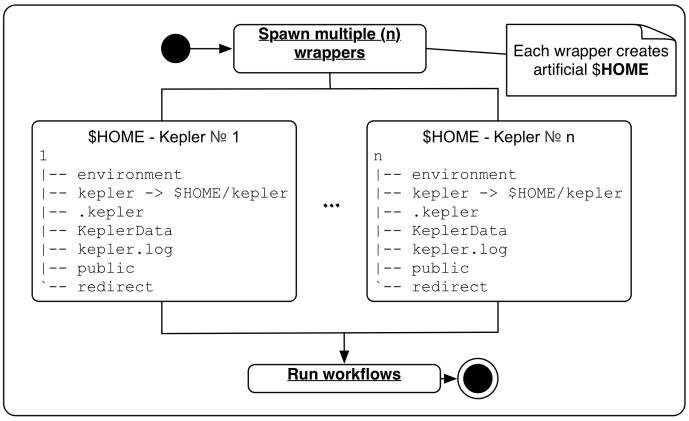

Figure 3: Each Kepler is executed inside dedicated $HOME directory

After structures are ready, all Kepler processes are started as serial jobs via ***mpirun*** command. This way, we get propper distribution of all processes over requested resources (let it be 4, 16, 32, 64, 128 CPUs). We have successfully scaled to 128 CPUs while running production ready workflows (e.g. SYCOMORE).

In order to pass the location of new ***$HOME*** structure we use two solutions. First one is based on system settings. We modify ***$HOME*** variable before running code (Kepler). Another approach is to pass *user.home* property directly into JVM. This way, only JVM is altered by settings.

Process of submission is divided into two parts: generation of input script (this script depends on the Resource Manager being used), execution of task at

batch nodes (submission of prepared script). At the time of writing, we have been able to use this approach for two, different Resource Managers: Load Leveler and SUN Grid Engine. Both cases require adaptation to proper submission format, however, thanks to modular architecture of the solution this is not a drawback. All we have to do to adapt script for a given submission system is to prepare templates that will be altered by user's parameters (e.g. name of the queue, system requirements, nodes reservation, etc.). Schema of the job submission is depicted on Figure 4.

3.1. Task submission

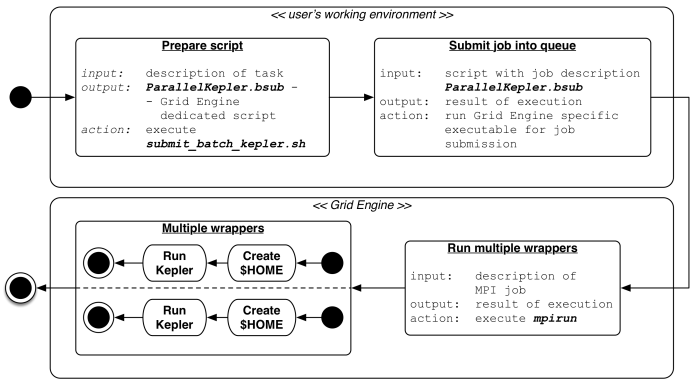

Figure 4: Steps taken to run Kepler inside batch queue

Whole process takes part in two, separate areas: user space and execution environment. User space is the place where user can alter parameters of batch job. It is possible to set predefined parameters, or even change submission script completely. It depends on specifics of the job. Most, typical, workflows can be run using default, predefined set of scripts. However, in case of specific requirements, it is possible to alter each part of the execution chain. Jobs can be submitted in two flavours: synchronous, asynchronous. In case of synchronous jobs, submission scripts executed within user's space wait until the job is done, in case of asynchronous jobs, submission scripts simply start job and quit immediately.

3.2. Limitations

The solution we have described above has some limitations. It makes strong assumption on what is available inside ***$HOME*** during Kepler execution. It means that before running workflow we have to make sure all locations required during execution are provided. Providing new working space for ***$HOME*** directory has some advantages as well as disadvantages. An advantage is application wide setting that is visible in each and every part of the code (let it be executable called from JVM, code called via JNI or call to *System.get("HOME")*). Disadvantage of this solution is related to internals of legacy code. In case there is a strong assumption in the code, in terms of location of files, we have to make sure they are properly linked in the artificial ***$HOME*** structure. There is yet another issue that can be solved using modified ***$HOME*** variable. In some batch queues (with shared *HOME* volumes) it might be that there are dedicated environment variables that refer to ***$HOME*** in case we are at frontnodes or batchnodes. In that case, if one wants to have universal binary that works fine in both environments it is required to alter *HOME* variable.

As we already mentioned, another approach is to change *HOME* location directly inside Java VM. This is possible thanks to ***user.home*** property that can be passed to JVM via ***-D*** argument.

```
java -Duser.home=$NEW_HOME_LOCATION JavaApplication
```

This approach is also suitable in many cases, however, it may fail in case there are references to *$HOME* variable directly in the code. This might be the case for legacy code. Of course, we cannot prevent any possible misusage of environment variables and we cannot assure that user's code will not base its execution on *user.home* or HOME *variables*.

3.3. mpirun based execution and its limitations

In solution provided we use *mpirun* in order to properly distribute serial applications over reserved nodes. This approach, however, has some limitations. In case of workflows that run *mpirun* themselves, this solution will not work

correctly. This is related to doubled initialization of ***MPIWORLD***. It is not possible to run *mpirun* from within *mpirun*.

3.4. Working with various grid engines

Current solution is somehow bound to grid engine it operates on. This is dictated by a number of factors: command line applications used for job management, submission script layout, environment variables being used by a given grid engine. In general, we have to pay attention to its specifics. These elements play a role in the solution and we cannot use current approach as a general solution for all cases. Before porting it to new grid engine, we have to make sure what elements are available for us and how should we alter current solution to get it fully functional in a new environment. Fortunately, thanks to modular approach of the solution this is just a matter of replacing parts of the scripts with proper code. We have been able to port this solution, initially developed for SUN Grid Engine, to Load Leveler without much effort.

4. Applications

This approach was successfully applied to various workflows within various environments. We have been able to adapt this solution for EUROfusion[7] developments (workflows): ETS, IMP5HCD (Heating, Current Drive and Fast Particles), SYCOMORE. In addition to that we have been able to port the solution into JET infrastructure where we have been able to use this approach to submit ETS workflow into queue system deployed at JET (Joint European Torus).

4.1. ETS

Development of the operating scenario for fusion reactor requires integrated modelling addressing the critical reactor issues: plasma heating and fuelling,

[7] `https://www.euro-fusion.org`

radiation from impurities, MHD stability, etc. Since all these issues are interlinked, to demonstrate a successful operational scenario, the relevant physics need to be included in a single simulation. Thus, the environment used for scenario modelling should allow for the integration of multiple codes and physics modules into a single scientific workflow. The European Transport Simulator (ETS) [5] is an outstanding example of such an integrated workflow. The ETS workflow couples individual physics modules e.g. calculating the plasma magnetic equilibrium, deposition (by auxiliary heating systems) transport of energy and, impurity radiation and MHD. It also offers several options of different fidelity for each physics component. Previously, ETS was verified against state-of-the-art transport codes and used to analyze data from existing tokamaks. In this work, it has been applied to study the possibility to control the plasma density in a reactor size machine by means of multiple injections of frozen pellets composed of 50/50 mix deuterium-tritium.

Simulations were performed for five different poloidal angles of injection position, each with several pellet sizes and velocities.[6] To obtain results for various configurations and perform calculations in parallel, each workflow was preset with given configuration and executed in parallel with other workflows.

4.2. Sycomore

SYCOMORE is a modular system code for fusion reactor design. All the physics and technology calculations are handled by a Kepler workflow[7]. Every run of the workflow gives a reactor design point. The workflow is coupled to an external optimisation framework called Uranie. This framework is used to sample a chosen set of input variables over a chosen range to assess the sensitivity of the designs to these particular variables. Uranie can also be used to find optimum design points: in this case, optimal input variables are searched following a figure of merit and constraints. A typical example of an optimisation problem is the following: find the smallest possible reactor with a minimum of 500 MW net electric power. A genetic algorithm is used to carry out such an optimisation process. In either case (sampling or optimisation), the SYCOMORE workflow

has to be iterated a large number of times: from a few dozens iterations for simple sampling runs to several 10^5 or even 10^6 iterations for multi-criterion optimisation runs. Since every iteration of the workflow takes between 2s and 10s, it is therefore mandatory to be able to run several of them in parallel to achieve reasonable run times.

SYCOMORE and Uranie use genetic algorithms to carry out optimisations. Therefore, within a generation of the population generated by the algorithm, every iteration of the workflow is independent. As a consequence, parallel scaling is very efficient. More details on a scaling test can be found in the following paper: *Coupling between a multi-physics workflow engine and an optimisation framework*[8].

No of CPUs	**Wall clock time**	**Teval**
256	4 min 56	8.4 s
128	7 min 53 s	6.0 s
96	11 min 09 s	5.3 s
64	13 min 26 s	5.1 s
48	17 min 30 s	4.9 s
32	23 min 27 s	4.4 s
28	53 min 20 s	4.3 s
24	1 h 02 min 37 s	4.2 s
20	1 h 15 min 47 s	4.2 s
16	48 min 36 s	4.4 s
12	2 h 10 min 55 s	4.0 s
8	1 h 38 min 32 s	4.1 s
4	3 h 58 min 52 s	4.1 s

Table 2: Computation time. Process with rank 0 handles only data transfer. Teval is an averaged time spent for one SYCOMORE evaluation per session.

4.3. IMP5HCD

Applying parallel based solution into IMP5HCD workflow allows to reduce computation time – several, different cases, are run together. This is particularly useful in case one wants to verify various combinations of actors being used in workflow. In case of IMP5HCD, parallel execution was used for testing all the possible combinations between deposition codes and Fokker-Planck codes. This calculations were done while benchmarking NBI within one of EUROfusion's activities. IMP5HCD workflow is partially based on Monte Carlo approach. This way, scalability is quite efficient – 128 CPUs provides 128 faster computations [9, 10, 11].

4.4. Using Docker for Kepler encapsulation

With Docker containers, encapsulation of software became easier to achieve. Treating Docker as universal solution for problems of all types is unwise, however, it can help when all we want to achieve is to encapsulate a single application or service. Docker containers wrap a piece of software in a complete filesystem that contains everything needed to run: code, runtime, system tools, system libraries - anything that can be installed on a server. Docker image created this way will always contain a stable snapshot of the environment for the software to run [4]. This approach is very well suited for tasks we are focusing on while developing solutions based on Kepler[8].

Running Kepler in a multi-user environment, can be improved with Docker based approach. In case users want to execute multiple workflows at the same time, they can run multiple Docker images containing Kepler and workflow itself (Figure 5).

This way, application is fully separated from regular user's environment. Docker based image contains all the required components – Kepler, libraries

[8]Part of this work has been co-funded by the Horizon 2020 Framework Programme through the INDIGO-DataCloud Project, RIA-653549.

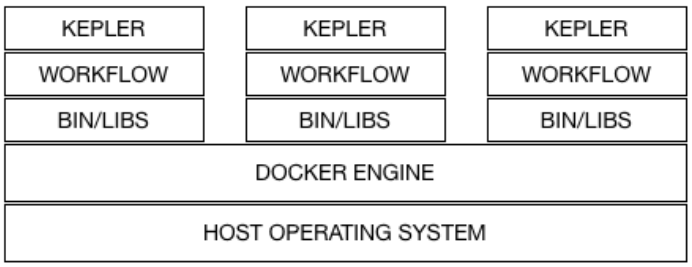

Figure 5: Running multiple Kepler instances using Docker based approach

and actors that are not distributed with Kepler. Only varying element is workflow and its execution parameters. In fact, this approach provides numerous advantages comparing to originally presented solution.

If we discuss aprroach where Kepler is executed as an MPI proccess we have to keep in mind that no actor executed inside workflow can be based on MPI. It is impossible to run MPI process from another MPI process. This might be a limitation in case someone wants to run workflow that spawns local, MPI based, computations. If we use Docker based approach, we do not face this issue. In that case, Kepler is executed as serial process and there are no conflicts with underlying processes.

In case of ***$HOME*** variable manipulation we have to juggle multiple locations of Kepler, depending on environment we want to provide (e.g. set of actors to be included inside Kepler). In addition to that, we interfere with user based settings for the environment. If we encapsulate Kepler, actors, and workflow inside Docker image all we have to do is to make sure each image contains set of components required for execution. All the images can be started in parallel and there will be no clash between them. This approach is even better if we want to share whole Kepler with other users. If we want to provide users with a whole execution environment, we can simply share Docker image with everything that is needed to run simulation.

Even though it seems like a perfect solution for everybody, we have to keep in mind that Docker based solution may not be well suited for all scenarios.

First of all, we have to remember that Docker images are required to contain everything that is needed for execution. In case of complex workflow where underlying layers require various external libraries it might be a challenge. For systems such as the ITM Gateway, i.e. a dedicated host to develop both codes and their Java wrappers in form of Kepler actors, a working Docker image may as well duplicate almost the whole system itself just to run a simple workflow.

Additionally, Docker based solution has one more issue that is related to security policy. Currently, Docker daemon is required to be run as the `root` user and it is strongly emphasised to give access to Docker management only to a limited set of trusted users. In some cases of highly restricted environent, this distinguished group may contain only administrators and not regular users of Kepler workflows.

4.5. Executing Kepler inside Docker

Docker allows to manipulate existing images (e.g. upload files into image), it allows to mount external file system locations (e.g. it is possible to mount local file system inside running Docker image) and it provides ways to run applications as soon as Docker image is up and running. This way, it is possible to build very efficient execution schema where each and every Docker instance with Kepler can perform different calculations.

One can run very simple use case with Docker executing external calculation by running "Hello world" sample[9].

```
docker run ubuntu /bin/echo 'Hello world'
```

With this template it is possible to build more complex use-case where Docker image is executed together with workflow

In presented approach, user prepares one, common image. Image itself contains Kepler with all the required libraries, actors, etc. Whenever user wants

[9] `https://docs.docker.com/engine/tutorials/dockerizing/#hello-world-in-a-container`

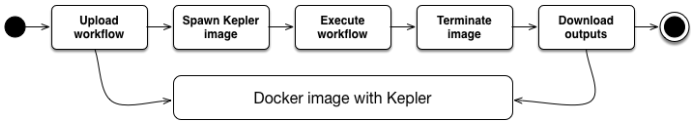

Figure 6: Execution schema with Docker based approach

to run different workflow he just specifies what workflow will be used during execution. It is possible to make the solution even better. By encapsulating VNC server inside Docker image one can start Docker image and run Kepler in both GUI/non GUI modes without problems. In fact, this kind of Docker based container is already implemented, as part of INDIGO-DataCloud project[10]. We have succesfully executed multiple instances of this image both using OS X and Linux based host operating systems. In addition to that we could run workflows inside the container without any issues.

4.6. Running Kepler in cloud infrastructure

Running multiple Kepler instances locally might be interesting task in case user wants to perform some basic tests. There is one issue here. In case of typical hardware, users will reach limits of available resources (memory/CPUs) quite quickly. Running multiple Kepler instances locally, for large scale computations makes not much sense. In case of using Kepler inside Docker users gain another way of running Kepler with ease – cloud computing. As we can encapsulate everything that is needed for computations in single package, users can easily run Kepler in cloud based environment. This way they are not bound to HPC/GRID policy. In fact, users are free to choose best computation plans by using Amazon EC2, Docker based solutions for cloud computing, IBM Bluemix or previously mentioned INDIGO-DataCloud based infrastructure (INDIGO-DataCloud relies on Apache Mesos[12] and Kubernetes[13] solutions) – just to name few that support Docker based containers.

[10] `https://github.com/indigo-dc/ansible-role-kepler/tree/master/docker-kepler`

5. Conclusions

Running applications in parallel can save reasonable amount of time. In our opinion, presented solution elevates Kepler usage to the next level. We no longer use it as a workflow design and serial execution tool, but rather, as a platform for execution of numerous simulations at the same time. We are sure that suggested solution provides users with new abilities when it comes to Kepler's utilisation. There are two, main areas where we find proposed solution to be very efficient:

- parametric scan,
- task decomposition.

In case of parametric scan, workflows executed in Kepler perform computations using the same input data set, but they use different parameters set. In here, we have multiple Keplers running multiple workflows over the same data. This scenario is useful for all simulations where we do some parameters based optimization (Figure 7). Second approach is useful in case we are dealing with large input data set and we want to reduce computation time for this given input. In this case, initial problem is sliced and each Kepler runs exactly the same workflow but we provide different input data. Each workflow performs computations for a single slice (Figure 7).

Parametric scans are based on running exactly the same workflow (in terms of its architecture) but we change values of parameters. At the moment, we use two approaches here: altering XML file, passing parameters via command line. Both solutions have their limitations (e.g. potential risk of breaking XML structure). In future we want to focus on Workflow Manager module. This way, we hope to provide better support in terms of maintaining workflows as well as allowing users to alter workflows with ease.

When it comes to running the same workflow over different data, situation is very similar to parametric scan based approach. In this case we still have to alter workflow a little bit. We have to (at least) point to input and output that

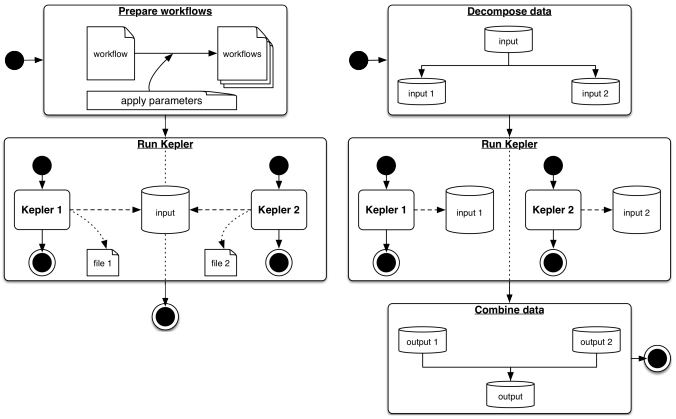

Figure 7: Parametric scan and data decomposition based scenarios

will be processed by each and every workflow. In this case we also look forward for new releases of Workflow Manager.

Docker based execution of Kepler is another place for conducting further research. There are still areas of interest that have not been fully covered by our research but seems to be very interesting. We are willing to take a closer look at the MPI based computations triggered by Docker based installations of Kepler. We are also looking for more user friendly way of supplying running Docker images with different workflows to reduce startup overhead.

References

[1] EUROfusion, 2016, `https://www.euro-fusion.org`.

[2] iter, 2016, `https://www.iter.org`.

[3] M. P. et al, Approaches to distributed execution of scientific workflows in kepler, in: Fundamenta Informaticae 128, 2013, pp. 1–22.

[4] Docker, 2016, `https://www.docker.com`.

[5] D. K. et al., Numerical analysis of jet discharges with the european transport simulator, in: Nucl. Fusion 53, 2013.

[6] D. K. et al, Predictive simulations of reactor-scale plasmas fuelled with multiple pellets with the european transport simulator, in: Proc. EPS Lisbon, 2015.

[7] C. R. et al, Demo reactor design using the new modular system code sycomore, in: Nuclear Fusion 55 vol 7, 2015.

[8] L. D. Gallo, C. R. et al, Coupling between a multiphysics workflow engine and an optimization framework, in: Computer Physics Communications 00, 2015, pp. 1–19.

[9] O. A. et al, Nbi benchmark on the european integrated modelling (eu-im) framework, in: 14th ITPA on Energetic Particle Physics, 2015.

[10] M. S. et al, Benchmarking neutral beam injection codes within the european integrated modelling framework, 2015.

[11] M. S. et al, Nbi benchmarks in integrated modelling frameworks, in: 15th ITPA on Energetic Particle Physics, 2015.

[12] Apache mesos, 2016, `http://mesos.apache.org`.

[13] Kubernetes, 2016, `http://kubernetes.io`.