Fundamenta Informaticae 128 (2013) 1–22
DOI 10.3233/FI-2012-881
IOS Press

Approaches to Distributed Execution of Scientific Workflows in Kepler

Marcin Płóciennik*, **Tomasz Żok**
Poznań Supercomputing and Networking Center
IChB PAS
marcinp, tzok@man.poznan.pl

David Abramson
Faculty of Information Technology
Monash University,Clayton
david.abramson@monash.edu

Marcos Lopez-Caniego, Isabel Campos Plasencia
Instituto de Fisica de Cantabria, CSIC
caniego, iscampos@ifca.unican.es

Bartek Palak, Michał Owsiak
Poznań Supercomputing and Networking Center,
IChB PAS
bartek, michalo@man.poznan.pl

Ilkay Altintas, Jianwu Wang, Daniel Crawl
San Diego Supercomputer Center
University of California San Diego
altintas,jianwu, crawl@sdsc.edu

Frederic Imbeaux, Bernard Guillerminet
CEA, IRFM
frederic.imbeaux, bernard.guillerminet @cea.fr

Wojciech Pych, Pawel Ciecieląg
Nicolaus Copernicus Astronomical Center, PAS
pych,pci@camk.edu.pl

Yann Frauel and ITM-TF contributors
CEA, IRFM
yann.frauel@cea.fr

Abstract. The Kepler scientific workflow system enables creation, execution and sharing of workflows across a broad range of scientific and engineering disciplines while also facilitating remote and distributed execution of workflows. In this paper, we present and compare different approaches to distributed execution of workflows using the Kepler environment, including a distributed data-parallel framework using Hadoop and Stratosphere, and Cloud and Grid execution using Serpens, Nimrod/K and Globus actors. We also present real-life applications in computational chemistry, bioinformatics and computational physics to demonstrate the usage of different distributed computing capabilities of Kepler in executable workflows. We further analyze the differences of each approach and provide a guidance for their applications.

Keywords: Kepler, scientific workflow, distributed execution

*Address for correspondence: Noskowskiego 12/14,Poznań, Poland

1. Introduction

Distributed execution of scientific workflows. Scientific workflow management systems are designed to compose and execute computational scientific applications and data manipulation steps. Workflows in general can be described as graphs where the nodes represent computational components and the edges represent data flow between components. Distributed execution of a workflow often needs a general framework for transporting and executing workflows and sub-workflows across a distributed set of computing nodes in order to improve execution performance through the utilization of these distributed computing nodes.

Requirements for multiple computing models for distributed execution. There are many specific requirements that are important for multiple computing models for distributed execution. Workflow systems should be able to execute jobs on different kinds of computing environments, e.g. SMP machines, homogeneous clusters or heterogeneous nodes in a distributed network like in the Grid environment. Such orchestration systems should be able to address at the same time heterogenous resources including the High Performance Computing, High Throughput Computing and Cloud Computing. In this context, the difference between HPC and HTC is that HPC resources (e.g. supercomputers, large-scale clusters) provide a good interconnection of CPUs/cores while HTC resources (PC pools, smaller clusters, etc.) do not. Ideally, workflow systems should hide from users the whole complexity, technical details, infrastructure, middleware and minimise the configuration required by users as much as possible. Such systems should be able to address transparently from a user's perspective different kinds of technologies like JINI, Web services, etc. So from the technical point of view workflows should interact with underlying middleware and system, be able to discover available resources, allocate them, make data transfers of the input and output data, handle the status of the running jobs. In case of interacting with the distributed systems, it is important to have the mechanisms for job monitoring and failure recovery. Another important challenge is the workflow provenance which is the ability to automatically record the lineage of data generated during workflow execution. The provenance of a workflow contains necessary information about the workflow run enabling scientists to easily share, reproduce, and verify scientific results.

Motivating examples and wish list. There are different kinds of use cases that can be addressed by the workflow systems using distributed resources:

- Application workflows with urgent deadline-driven requirements such as weather prediction, economic forecasting. It becomes increasingly important in respect to growing numbers of severe natural disasters such as tornadoes, floods or economical crisis situations. In case of the weather prediction, there is a need for a large scale modeling in the areas of meteorology coupled with an environment for analysis and prediction therefore it is a natural workflow requirements. Such applications usually need to compute a lot in a very short time. Besides classical computing facilities, currently they can also utilise the cloud computing resources, which are available on demand.

- Large simulations and modeling platforms, e.g. from the field of the Nuclear Fusion. The realistic simulations have three major requirements: handling very big data, intensive computing and integration of different models. In respect to the data, a normal way is to have ability to split the data in chunks and distribute them, such as using Hadoop or MPI-IO. For the intensive computation, parallelisation of the codes and execution on distributed computers is very important. Workflows

should cooperate with load balancing systems in order to try to optimise the execution. Usually these two elements are separated (using the "separation of concerns" principle) and most of the splitting is done manually by the user/developer (the chunk of data is defined by the user, the MPI jobs are also defined manually), so a global optimisation is of great importance. Such optimisation should mix data moving and code execution time.

- The bioinformatics and biomedical research that involve less computation but need access to large-scale databases. The distributed workflows should allow users to access remote databases. Scientists often needs to conduct parametric or exploratory studies that involve launching multiple parallel workflows.

- The computing science scenarios that usually use the map-reduce style of programming like the distributed rendering.

In this paper we explain several approaches available in Kepler Scientific Workflow System. The paper is organised as follows. In Section 2 we introduce and characterize Kepler, in Section 3 we analyze the state of the art, while in Section 4 we introduce the different approaches of distributed execution of workflows in Kepler. In the subsequent section we describe workflow use cases with some experimental results illustrating exploitation of each of the approaches. In the next part we compare all the approaches. We also provide directions for further work in the summary section.

2. Kepler Scientific Workflow System

In this paper we are focusing on different approaches to distributed execution of workflows using the Kepler environment[1]. Kepler [4] is an open source project for designing, executing, and sharing scientific workflows. Kepler builds upon the dataflow-oriented Ptolemy II system [2], which is a software system for modeling and simulation of real-time embedded systems. Kepler workflows model the flow of data between a series of computation steps. Kepler facilitates the execution process, data movement and gathering provenance information. The environment is composed of a Graphical User Interface (GUI) for composing the workflows, that could be run on Windows, Linux and other operating systems, and a runtime engine for workflow execution. This engine can run in a GUI or command-line mode. Kepler supports different workflow model of the computation, inherited from the Ptolemy system, including Synchronous Data Flow (SDF), Continuous Time (CT), Process Network (PN), and Dynamic Data Flow (DDF), and others. The basic Kepler workflow elements are actors and directors. Kepler separates the execution engine from the workow model, and assigns one model of computation, director, to each workow. The directors control the execution of a workflow while actors take their execution instructions from the directors and perform the computations/actions. As an example, in a workflow with a PN director, actors can be thought of as separate threads that asynchronously consume inputs and produce outputs. Actors (that are the java classes) contains parameterized actions performed on input data to produce output data. Actors communicate between themselves by sending tokens, that are the data or messages, to other actors through so called ports. Each actor can have many ports and can only send tokens to an actor connected to one of output ports. After receiving a token, actor runs the required

[1]Kepler Project: http://www.kepler-project.org http://www.kepler-project.org, 2012
[2]Ptolemy II: http://ptolemy.eecs.berkeley.edu/ptolemyII/ http://ptolemy.eecs.berkeley.edu/ptolemyII/, 2012

number of times, and it fires new tokens with the resulting data on the output port. Actors can be grouped into composite actors that are sets of actors bundled together in order to perform complex operations. Composite actors are used in workflows as a nested or sub-workflows. Each of such composite can have it's own director, different that the director used in parent workflow.

2.1. Extensibility of Kepler

Extensibility of Kepler has been a key capability to support easy collaboration. First, Kepler is modularized by defining modules and suites. Each module is for certain functionality. For instance, the hadoop module is to run Kepler workflows on Hadoop environment. Each suite contains of several modules and/or suites in order to work for a bigger usage scenario. For example, the Distributed Data-Parallel Framework (DDP) suite contains the hadoop module, stratosphere module and other modules/suites to provide different choices to execute DDP workflows. In order to get customized Kepler software for their requirements, users just need to select modules and suites, and specifying their dependencies. By this way, we can achieve easy on-demand Kepler extension. Second, Kepler can be extended at the atomic actor, composite actor or director level. New actors can be added into Kepler for new standalone functionalities by following the actor specifications. When new execution semantics requirements of existing actors and workflows appear, we can create new composite actors and directors. One advantage of this approach is that we can re-use existing workflows and actors. As will be explained in the following sections of the paper, we use composite actor and director level extension for some distributed execution requirements. Then the same workflow can execute on different environments by just switching the composite actors or directors.

2.2. Workflow Distributed Execution via Kepler

In Kepler community, there are various requirements on distributed execution in different environments, e.g., ad-hoc network, Cluster, Grid and Cloud resources. So since the very beginning, Kepler software is targeted not to a particular distributed execution requirement, but to support each scenario if it is required. The extensibility of the Kepler gives us the flexibility to choose the best approach for each requirement. The distributed execution approaches addressed in Kepler can be categorized in different levels:

- Workflow level: The whole workflow is executed in distributed environment, and all the corresponding actors of the workflow are executed on the same node. As the example there are remote workflow execution via Grid middleware (Serpens) and Cloud software (Amazon EC2, OpenNebula).

- Composite Actor level (Sub-Workflow level): In this case, actors can be executed locally or in the distributed environments. The composite actor determines where and how the sub-workflow inside of the composite actor is executed. Examples include Master-Slave Distributed Execution [25] and MapReduce Composite actor [26].

- Director level: Since director controls the execution of a workflow, we can create new directors for distributed execution semantics. Examples include the directors in Distributed Data-Parallel framework and Nimrod/K director.

- Atomic Actor level: In this case, distributed computing and data resources are utilized within one atomic actor. Examples include Serpens actors.

The advantages of supporting multiple distributed executions and environments in Kepler include: 1) Unified Interface: users can access multiple distributed environments, e.g. Cloud and Grid, within one workflow if the application needs to; 2) Actor Reuse: Most of the actors in Kepler can be reused in different distributed executions and environments; 2) Provenance: Kepler framework can capture the execution information of different distributed Kepler workflow for possible performance analysis and failure recovery[11].

2.3. Fault Tolerance via Kepler

Fault-tolerance is the method that enables a system to continue operation, even at a reduced level or with changed parameters, rather than failing completely, in case of failure of part of the system. In computer-based systems fault tolerant design allow to continue operations (with a reduction usually in response time) in the situation of some partial failure, and is the response to the problems either in the software or the hardware. Fault-tolerance system is designed in particular for high-availability systems. Distributed systems are often prone to failure caused by many reasons on different levels. In respect to the scientific workflows failure handling techniques can be divided into two different levels, namely task level and workflow level. The Kepler workflow system allows to apply and combine different methods of fault tolerance including the alternate resource on the task level and on the workflow level the user-defined exception handling and the workflow rescue. The Kepler workflow system itself has the rerun capability based on the provenance of the information collected by the system [11, 19].

3. State of Art in Distributed Scientific Workflows

There are several scientific workflow systems that provide functionality of building and executing distributed workflows. The most popular systems, besides Kepler, include Pegasus [13], Taverna [17], Triana [10], and P-GRADE [18]. Pegasus is a framework for mapping scientific workflows onto distributed resources including grid and Cloud-based systems. Pegasus has been used for extensive grid experiments and is focusing on middleware like Condor [23], Globus, or Amazon EC2. The P-GRADE Grid Portal is a web-based, service rich environment for the development, execution, and monitoring of workflows and workflow-based parameter studies on various grid platforms. Taverna is an open source and domain independent Workflow Management System. Taverna mainly targets bioscience workflows composed of Web services. Triana is a workflow-based graphical problem-solving environment independent of any particular problem domain. Triana workflows have incorporated a range of distributed components such as grid jobs and Web services. The supported grid middleware is KnowARC, and gLite. P-GRADE is focused on interoperability with Globus Toolkit 2, Globus Toolkit 4, LCG and gLite grid middleware. SHIWA is a project which aims at developing workflow systems interoperability technologies [20]. SHIWA Simulation Platform allows to execute workflows on various distributed computing infrastructures. It also enables the creation of meta-workflows composed of workflows from different workflow management systems. Workflows created in Kepler, Taverna, Triana can be either invoked or executed as embedded sub-workfows. The main difference between Kepler and other scientific workflow systems on distributed execution is extensibility. As mentioned above, the variety of distributed

execution techniques needs separate support from workflow systems. Kepler supports multiple levels of distributed execution, namely workflow level, sub-workflow level and atomic actor level. Further, starting from the 2.0 version of Kepler, a new architecture was introduced. Functionality was divided between modules, which are gathered in suites. A separate module consists of source codes for actors it publishes, external libraries, licensing information and other Kepler-related metadata. Developers can choose the best approach to extend Kepler and build corresponding suites in Kepler for each particular distributed execution requirement.

4. Different Distributed Execution Approaches in Kepler

4.1. Distributed Data-Parallel Framework in Kepler with Hadoop and Stratosphere

4.1.1. Distributed Data-Parallel Patterns

Due to the enormous growth of scientific data, applications increasingly rely on distributed data-parallel (DDP) patterns to process and analyze large data sets. These patterns, such as MapReduce [12], All-Pairs [21], Sector/Sphere [15] , and PACT [7], provide many advantages such as: (i) automatic data distribution and parallel data processing; (ii) a high-level programming model that facilitates parallelization and is easier to use than traditional programming interfaces such as MPI [14] and OpenMP [9]; (iii) reduced overhead by moving computation to data; (iv) high scalability and performance acceleration when executing on distributed resources; and (v) run-time features such as fault-tolerance and load-balancing. In many cases, applications built for execution on a single node can be executed in parallel using DDP patterns.

Each above DDP pattern represents an execution semantics type on how data can be partitioned and processed in parallel. For instance, Map treats each data element independently and process it in parallel; Reduce creates groups based the keys of data element and process the whole group together; All-Pairs and Cross in PACT have two input data sets, select one element from each set and process the two element pairs in parallel. To use the correct DDP pattern, users need to understand how these patterns work and how they can be applied to their applications. Generally, there are two different solutions applying these DDP patterns to a legacy application to make it parallelizable. The first one is to modify its internal logic and implementation based on the structure of an applicable pattern, so data partition and processing are done inside of the application. The second solution is to wrap the application through an applicable pattern. Then data partition is done outside of the application and only partial data is feed in each application instance. For our Kepler DDP framework, we use the second solution because it is easier to reuse common tasks such as data partition and pattern execution stub. Using this solution, we can quickly parallelize a large set of legacy applications.

4.1.2. Distributed Data-Parallel Actors in Kepler

The Kepler DDP framework allows users to construct workflows with data-parallel patterns. Each actor in this framework corresponds to a particular pattern: Map, Reduce, Cross, CoGroup, and Match. These patterns correspond to the input contracts in the PACT programming model [7]. Similar to other actors, the DDP actors can be linked together to express dependencies among a chain of tasks, and can be nested hierarchically as a part of larger workflows.

Each DDP actor specifies the data-parallel pattern and execution tasks. Data is split into different groups based on the type of data-parallel pattern, and the tasks can be executed in parallel for each group. For example, Reduce partitions the data into groups based on a key, but Map places each datum in a separate group.

The execution steps are applied once the data are grouped by the pattern. A user can specify these execution steps either by creating a sub-workflow in Kepler using Kepler actors and directors, or by choosing from a library of predefined functions to process or analyze the data. In the latter case, developers can write their own functions in Java and add these to the library.

The Kepler DDP framework contains actors that specify how to partition and merge data. The underlying execution engine must know how to partition the data before it can apply data-parallel patterns. Similarly, since the partitioned results are usually merged at the end of the analysis, the engine must know how to combine the data. The *DDPDataSource* actor specifies how to partition the input data and its location in the file system, and *DDPDataSink* specifies how to merge the output data and where to write it. The user can choose from a library of common partitioning and merging methods, and new methods can be imported. Currently, these actors support Hadoop Distributed File System (HDFS) [3] and the local file system.

4.1.3. Distributed Data-parallel Directors

The DDP framework provides two directors, *StratosphereDirector* and *HadoopDirector*, to execute workflows composed of DDP actors. The *StratosphereDirector* converts DDP workflows to the PACT programming model and submits the job to the Nephele execution engine. Inside each job, Kepler execution engine is called with the partitioned data and sub-workflows in the DDP actors. The director also converts the data types between Kepler and Stratosphere during their interactions.

Besides the work similar to StratosphereDirector, a big challenge for HadoopDirector is to work with Match, CoGroup and Cross DDP pattern. HadoopDirector has to transform each pattern into Map and Reduce to create Hadoop jobs. Since Match/CoGroup/Cross pattern has two inputs and Map/Reduce pattern only process one input, we employ data tagging to differentiate inputs. In Hadoop, Map jobs can have multiple inputs. So for each Match/CoGroup/Cross actor, we first use a Map job to read the two inputs of the Match/CoGroup/Cross pattern and tag their values to differentiate the data if they are from different sources. After the Map job, a Reduce job will read the data, get two data sets based on their tags and call the sub-workflow defined in the Match/CoGroup/Cross actors based on the semantics of the corresponding pattern.

We are designing a generic DDP director that will work with Stratosphere, Hadoop, and other execution engines. This director will determine which engine to use based on availability, past execution statistics, data size, and intermediate storage requirements.

The fault tolerance for workflows in the DDP framework is supported by utilizing the underlying execution engines and file systems. For instance, Hadoop monitors the execution on each slave node and automatically restart its execution on another node if one slave is not responding. Besides, HDFS provides automatic data replication to support high data availability.

[3] Hadoop Distributed File System: http://hadoop.apache.org/ http://hadoop.apache.org/, 2013.

4.2. Serpens

The Serpens suite [8] contains two modules: gLite and UNICORE modules. These module act as adapters to the corresponding grid middleware, adding additional fault tolerance mechanism on top. It provides standard activities related to remote job management: submission, monitoring, data management, etc., as a set of actors, both basic and composite. By combining a meaningful sequence of such actors, client applications can create a complete workflow in a very flexible and convenient manner. Kepler is responsible for user interaction and managing multi-site, distributed and heterogeneous workflows. The heterogeneity here implies that a single workflow, spanning multi-step activities, contains HTC and HPC executions. The gLite module contains actors and workflows integrated with the gLite middleware. It allows creating VOMS proxies, transferring data using the Globus GridFTP protocol, submitting and managing jobs. The accompanying template workflow is a full use case implementation ready to be modified according to specific needs of the grid applications. The UNICORE module is responsible for supporting the UNICORE middleware. Actors contained within are based on a UNICORE Command-line Client (UCC). A user can request information from the UNICORE registry, use available storages and sites, submit and manage jobs described by JSDL also using the BES interface. This module contains several example workflows demonstrating the usage of each separate actor and a full use case implementation that highlights how to submit a job with input files, wait until it is successfully finished and retrieve its output. Fault tolerance mechanism in addition to underlying schedulers is trying to re-submit the jobs to last known properly working sites and services, trying to re-upload input/output files basing on last known working configurations, taking into account dynamic information collected from the sites every given period. The sites and services that are failing are excluded from further usage in the workflow(for the predefined period of 1-6 hours), so that there is much more chance that the job will end up at the proposer site during further runs. In case the central grid services are down and the workflow cannot continue, it is paused for given period (in a loop for n tries) and resumed to check if the services are back again. All the operations are recorded, and job id's stored in the local and remote database so that the workflow can be stopped or crashed any time and continue from the last known state.

Both modules are used in production by many applications, some of them mentioned in the use cases section.

4.3. Nimrod/K

Nimrod/K extends Kepler by providing powerful mechanisms for exposing and managing parallelism in the workflows [2]. It leverages the Nimrod tool chain that has historically supported the use of distributed computers to perform parameter sweeps and advanced searches, across time consuming computational models. Nimrod/K provides an ideal platform for using workflows for parameter sweeps, and supports both sweeps over workflows, and workflows that contain sweeps. Nimrod/K uses a new director called TDA(Tagged-Dataflow Architecture), and implements a Tagged-Dataflow Architecture originally designed for the MIT, RMIT and Manchester data flow machines. TDA augments data tokens with a tag, or colour field. When multiple tokens of different colours are available on the inputs of an actor, multiple independent instances of the actor are invoked. Thus, by controlling the colouring and de-colouring of token, the workflow can scale to use an arbitrary number of processors. This basic mechanism supports a wide range of parallel execution templates including a variety of parallel loops, scatter and gather mechanisms. Nimrod/K includes a set of parameter search Actors that support full sweeps, experimental design

sweeps and optimization based searches. Nimrod/K uses a set of Nimrod/G services for executing codes on distributed resources. These cover a wide range of Grid platforms (such as Globus enables resources) and emerging Clouds. Importantly, Nimrod/K abstracts low level Grid mechanisms such as file transport and remote invocation, freeing the workflow designer from concern about how to launch a computation on a remote resource. In addition Nimrod/K inherits the basic restart and retry mechanisms in Nimrod/G. If a machine or network fails, Nimrod/G will retry a number of times before giving up. The TDA implementation leverages a feature of Ptolemy that clones actors, and maintains a set of queues for each actor. When multiple tokens of different colour arrive on an actor's input, TDA creates new copies of the actor using cloning. These clones are destroyed after execution, which allows the graph to dynamically expand and contract as the concurrency changes.

4.4. Cloud

There are different developments and approaches implemented in Kepler for supporting the Cloud services. The most obvious direction of most of the developers was to choose Amazon EC2 cloud but also other cloud stacks like OpenNebula have been addressed.

As an example EC2 module has been implemented which contains a set of EC2 actors [24]. These actors can be used to manage EC2 virtual instances in the Amazon Cloud environment and attach Elastic Block Store (EBS) Volumes. Users can use these actors to start, stop, run or terminate EC2 instances. Through these EC2 actors and other actors in Kepler, users can easily build workflows to run their applications on EC2 Cloud resources. One EC2 workflow can link a variety of applications residing on different platforms by connecting multiple Amazon Machine Images (AMIs), e.g., Hadoop and Linux AMIs. To run an EC2 instance, users need to provide account information including 'Access Key', 'Secret Key' and credential 'Key Pair' file. An 'Image ID' parameter indicates the AMI ID for the instance. Users can also configure 'Instance Type', 'Availability Zone', and 'Minimal Instance Number' for the instance they want to run. Using these configuration parameter settings, the *RunEC2Instance* actor interacts with Amazon EC2 Web service in order to get configured instances.

4.5. Globus

Globus actors have been introduced since Kepler 1.0, for Grid job submission, monitoring and management [28]. Several facilitating actors are also provided to building Globus workflows in Kepler, such as GridFTP actor for data movement, and MyProxy actor for user authentication. Submission is done with usage of the Globus toolkit (GT) jobs using the Grid Resource Allocation and Management (GRAM) interface.

5. Scientific use cases

5.1. A DDP Workflow in Bioinformatics

To validate the proposed framework for scientific applications, we built a workflow that runs BLAST, a tool that detects similarities between query sequence data and reference sequence data [6]. Executing BLAST can be very data-intensive since the query or reference data may have thousands to millions of sequences. To parallelize BLAST execution, a typical way is to partition input data and run the BLAST

program in parallel on each data partition. In the end, the results need to be aggregated. Partitioning could be done for the query data, reference data, or both. We have discussed how to use Map and Reduce pattern to build DDP BLAST workflows via partitioning the query data in [5] and via partitioning reference data in [27]. The execution scalability of DDP workflows has also been verified in [27]. In this paper, we build a DDP BLAST workflow that partitions both reference data and query data. Since every reference split need to be processed against every query split, the Cross pattern is employed to express this semantics.

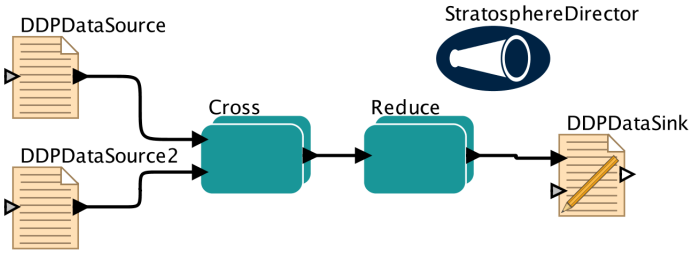
Figure 1(a) shows the overall DDP BLAST workflow. The two *DDPDataSource* actors are configured to read query sequence data and reference sequence data respectively. Each *DDPDataSource* actor generates key value pairs for the *Cross* actor. We built a customized partition method to split the query and reference data and send the data splits to *Cross* instances. The partition method follows the bioinformatics logic of the BLAST tool, so that each data partition sent into *Cross* instances has correct boundaries. It guarantees that the final results are still correct with the data partition.

Both *Cross* and *Reduce* actors have sub-workflows as shown in Figures 1(b) and 1(c), respectively. In each execution of the *Cross* and *Reduce* sub-workflow, it will read two key value pairs from its input actor, and generate key value pairs to its output actor. For each set of query and reference sequences gotten from its *CrossInput* actor, the *Cross* sub-workflow first formats the reference sequences into a binary format, and then executes BLAST against the formatted reference data and the query sequence set. The outputs from BLAST are then read by the *Reduce* sub-workflow, which sorts and merges them. Finally, the *DDPDataSink* actor writes the sorted outputs into a single file.

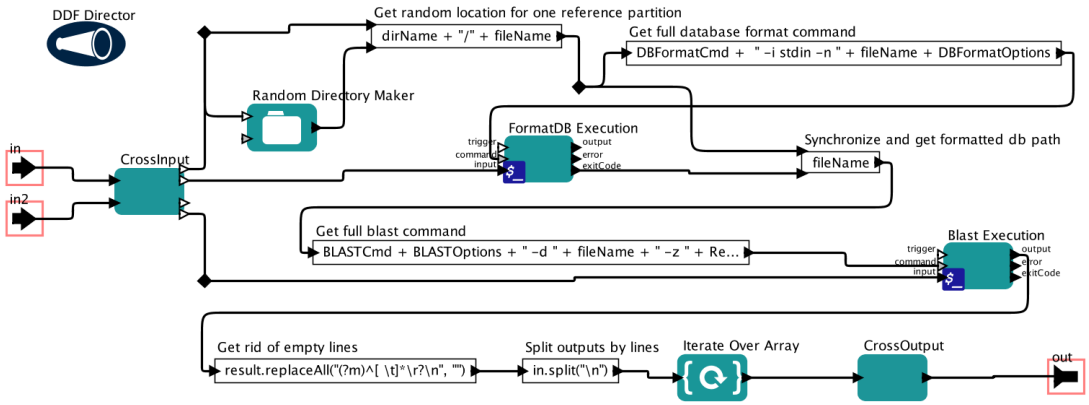
As explained in Section 4.1, the sub-workflows in Figures 1(b) and 1(c) will be executed by the Kepler execution engine inside of Hadoop/Stratosphere jobs. For each partial data, the Kepler execution engine reads them from the Hadoop/Stratosphere job, converts the data based on the Kepler data type, runs the sub-workflow built by users, convert the data and send them back to the Hadoop/Stratosphere job.

5.2. Fusion with HPC2K and Serpens

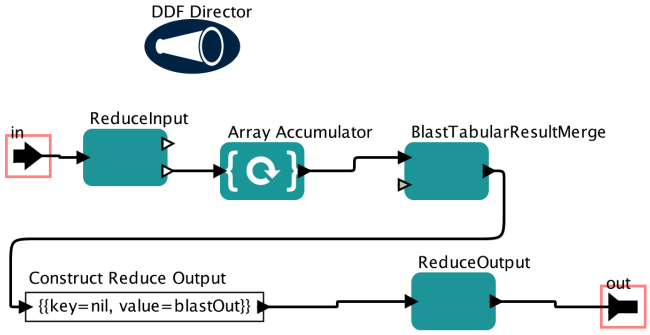
Kepler is being used also as a part of the European Fusion Development Agreement Integrated Tokamak Modeling Task Force (EFDA ITM-TF) of the software platform. The goal of the European ITM-TF is providing the fusion community with a validated suite of simulation tools for the for predictive and interpretive analyses of International Thermonuclear Experimental Reactor (ITER). ITER is intended to demonstrate the scientific and technical feasibility of fusion as a sustainable energy source for the future. Modeling activities will be an essential part of ITER plasma analysis and operation, and that is why the development of a predictive capability is of great importance. The full plasma simulation requires integrating a number of codes, each focusing on different aspect of the plasma dynamics. As part of the implementing tools to do comprehensive simulations, the ITM-TF identified the need for workflow orchestration tools and for mechanisms to transfer data between the modules that is independent from the computer language in which the modules are written, and is capable of transferring data between modules running on the same node, on the same cluster, and on widely separate compute resources including HPC and HTC resources. Kepler has been selected for the workflow platform, and has been extended with the ITM-TF actors and external tools enabling easy integration of the physics codes.



(a) The overall DDP BLAST workflow



(b) The Cross sub-workflow for parallel execution of BLAST



(c) The Reduce sub-workflow for result merge

Figure 1. A Distributed Data-Parallel BLAST workflow.

5.2.1. Example Workflow Use Case

The example use case shows the execution of GEM (Gyrofluid ElectroMagnetic) [22]: a three-dimensional two-fluid model for edge and core turbulence in tokamaks. In this use case, the turbulence code GEM is executed on an HPC (the code scales very well). Based on a basis-function approximation to the underlying kinetic model, GEM advances densities, flows, heat fluxes and temperatures as dynamical variables, coupled to an electromagnetic field response.

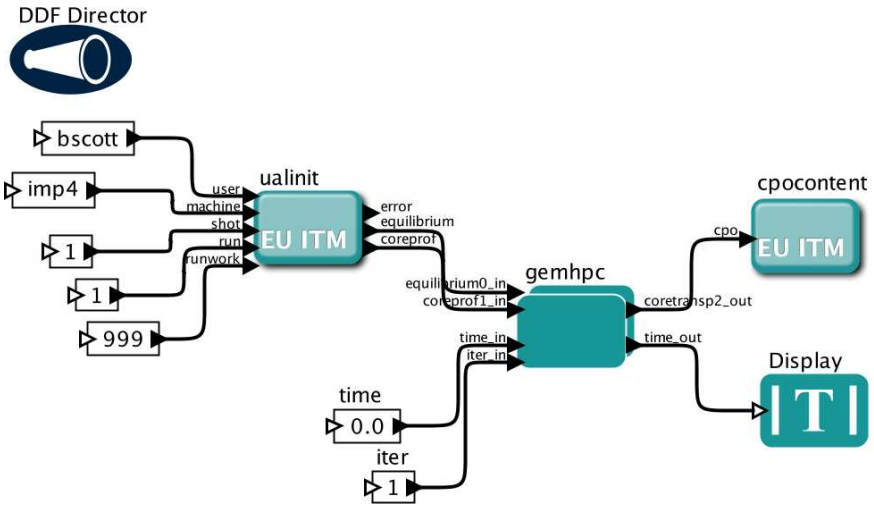


Figure 2. Basic Plasma turbulence workflow

The first actor in the workflow presented in Figure 2 , called ualinit, is used to open the database and specify the data to read. The second actor, called gemhpc, is a composite actor automatically generated automatically by tool called HPC2K. It handles submission and running of the code on the HPC machine. The last actor used in this example is cpocontent that displays the content of the output data. The MPI-based implementation allows to run the computations on multiple nodes at once. Since the target supercomputer is geographically distant to the database and because the amount of data is potentially huge (up to gigabytes), a special scheme for data transfer to all MPI nodes has to be used. A master-slave paradigm is implemented here. Only a single node (master) transfers the data through remote channel. Then all other nodes (slaves) are given copies of the data. Such approach minimize the amount of data that is transferred from remote database. The interactive communication with remote Kepler workflow and final storage of results is also coordinated and synchronized by the master node. This MPI master-slave management is automatically prepared by HPC2K tool. This use case is just a part of the complex Kepler physics workflows coupling different codes, that schedules the computations of hundred of actors to different computing resources depending on code characteristic and demands.

5.2.2. HPC2K

HPC2K [16] has been developed to integrate the physics program into Kepler workflows, and then run the programs on the remote resources. The goal of the HPC2K is to convert each physics program written in languages like Fortran or in C++ to a Kepler actor that automatically handles the jobs on the HPC or HTC resources. The program must initially be compiled as a library with a single main function that corresponds to the execution of the program. From a user point of view, HPC2K provides simple graphical user interface that allows defining parameters such as the program arguments, specific parameters, the location of the program library, the remote target system, etc. HPC2K user interface is presented in Figure 3. With two main tabs user configures the required parameters. The configurations can be serialized to XML files which can be reused later or shared among users.

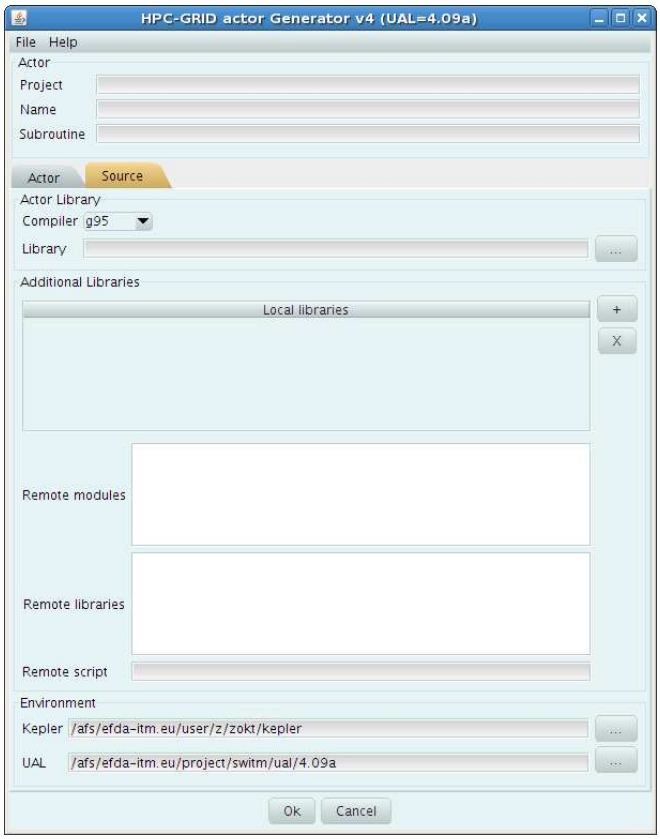


Figure 3. HPC2K user interface

HPC2K generates:

- A script that is executed on remote machine. It prepares the environment for compilation of the final executable and running it in the way specified by user.

- A wrapper program - generated in either Fortran or C++ that performs the remote data transfers in a two-way channel (input and output) between the database located on the machine where workflow is running and the program running on a remote computer. The wrapper is also responsible for administration and management of a job running on multiple cores at once.

- A Makefile to compile the wrapper and link it to the program library.

- A composite Kepler actor (i.e. a sub-workflow) that sends the above files to the execution host, prepares a description of the job and resources required for execution. After a job is executed, it monitors job's status, and collects the outputs after successful job execution. This composite actor contains components from the Serpens suite modules. These modules provide remote execution and job handling mechanism. The provenance of such actors configuration are captured as for other Kepler actors.

This composite Kepler actor generated by HPC2K allows to implement a special paradigm of program interoperation. Physics program developers are required to make the I/O of their program compliant with the standardized data model and then using HPC2K and Kepler it is possible to design complex workflows with sophisticated dependencies. The important feature of the HPC2K is the a interactive communication with running jobs that enable to pre-book the computing resources while the main workflow is running. As an example the GEM actor from the example above, if used in the loop uses the same instance and just sends new data to the job run on remote resources. HPC2K generated actors are used as a part of larger complex physics workflows, which besides including several such actors are performing computations locally (on local cluster).

5.3. Astronomy with Serpens

In this workflow the Kepler engine submits the number of jobs for different set of inputs (input images) to the grid using the Serpens suite, where the job itself is defined as a Kepler workflow. The workflow that is going to be run on the grid is used to process astronomical images. It uses the DIAPL package of the applications, that is the implementation of the Method for Optimal Image Subtraction [3], based on the DIA package implementation. The multilevel workflow is composed of a set of the C/Fortran codes that are applied to the images. There is a special tool developed allowing to generate C/C++ codes and add automatically into Kepler. The first step in the chain calculates FWHM and background level for basic image parameters like background level and shape of the stellar profiles for all images. In the next step best images are are selected (composite template image and reference image for geometric transformations). Then geometric transformation parameters and light flux ratios between each of the images and the reference one are calculated. After that the template image is subtracted from all the images within a group. Up to this point the whole procedure makes independent sub-workflow because the output data may be analysed in different ways. In particular it searches for variable stars and obtaining their light curves by means of aperture and profile photometry. This part is most time consuming and relies on C and Fortran codes. In general the input is a set of different fields on the sky and each field consists of a group of many images covering the same area on the sky (but at different times). Additionally, sometimes the field is photographed in different photometric filters so the analysis must be done for each filter separately. Such multi-level organization of input data naturally implies nested-workflows which are looped over. At the middle level, the described workflow is iterated over different filters and the highest level loop iterates over different fields in the whole data set. The workflow loops are running multiple times for a number of different variations among best images proposed for template composition. Then the user is presented with the final light curves obtained for different templates.

5.4. Astrophysics with Serpens

This use case is a parameter sweep kind of workflow where the Kepler engine submits the jobs to the grid using the Serpens suite. The workflow controls the production of realistic simulations of the anisotropies of the Cosmic Microwave Background (CMB) radiation. These simulations are used to test the performance of various algorithms designed to detect the presence or absence of non-Gaussian features in the simulated CMB maps, before they are applied to actual data as observed by Planck, a satellite from the European Space Agency. In order to test the algorithms it is required to produce large numbers of simulations. Each one of them is made of a combination of a Gaussian and non-Gaussian component plus

realistic instrumental noise that takes into account the observing strategy of the satellite. This workflow moves to the storage the necessary information to generate the simulations, then the parameter study job is submitted and each of hundreds sub-jobs generate in the actual simulation that is copied back to the storage once the job is finished. Then the postprocessing operations are applied to reluts as a part of the workflow. Since this workflow runs in the large distributed computing infrastructure, the workflow guarantees resubmission of the sub-jobs in case of the infrastructure failures. The workflow takes care in such situation about resubmitting the jobs or to the last working site, or if such site is not known, to first other site from the list of other sites that fulfills job conditions. The same situation is happening with handling the services like storages, information indexes, etc. During runs the workflow is learning the underlying infrastructure, so it can take up to date decision, if the resources choosen by grid resources brokers are failing. In addition in order to prevent situations where some of the copy threads are hanging due to bugs in grid middleware, for long running workflows (5-10 days) there is sometimes used external script killing and restarting workflow on purpose.

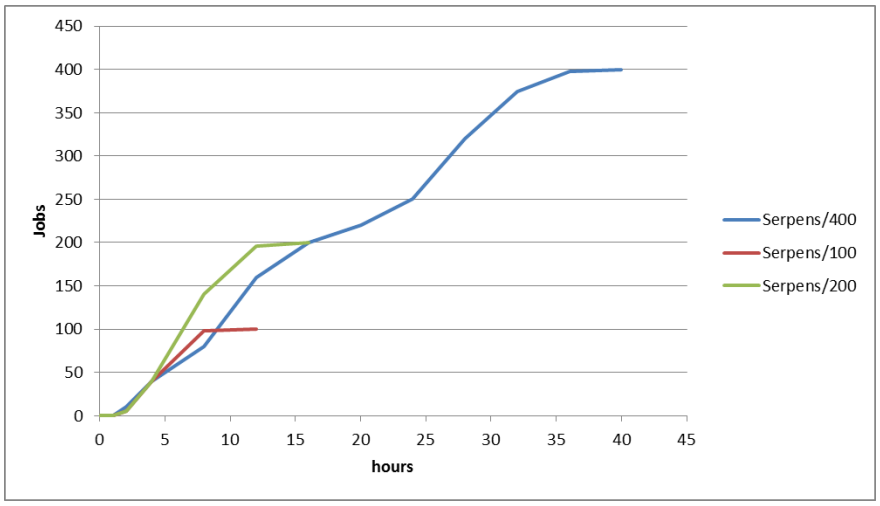


Figure 4. Astrophysics workflow runs

Figure 5.4 presents the results of runs of the same workflow with 100,200,400 independent jobs on physics VO in EGI.EU. Computing time of each job takes about 15 minutes and results download time about 2 minutes (220 MB) if done sequentially. The total time required increases since there is growing number of the resubmission required. The most important factor related with resubmissions was the accessibility of chosen storages and issues with some sites (that were empty beacuse of the issues from one hand, but because of that reason many jobs were summited there). Also the last 1-5 remaining jobs usually or stack in schedule state on some sites, or are resubmitted due to different reasons. Tests performed on the production heterogenous grid infrastructures (Physics VO in EGI.EU, about 100 runs of the whole use case) with this workflow showed usefulness of the failover feature, minimising from 90% to 99,9% the level of the successfully completed jobs. It is very good example of the production usage where the Kepler takes care of the high level fault tolerance mechanism.

5.5. Computational Chemistry with Nimrod/K

The workflow shown in Figure 5 depicts the computation, which involves the execution of the GAMESS quantum chemistry package a number of times, across a search space defined by 4 parameters (A, B, C and D). Here parameters A, B, C and D are to be determined that will provide the best fit between the real atomic system and target properties of a pseudo atom. The experiment used the results of the very large sweep over the A, B, C, D parameter space. This workflow use case has been published and discussed in details in [1]

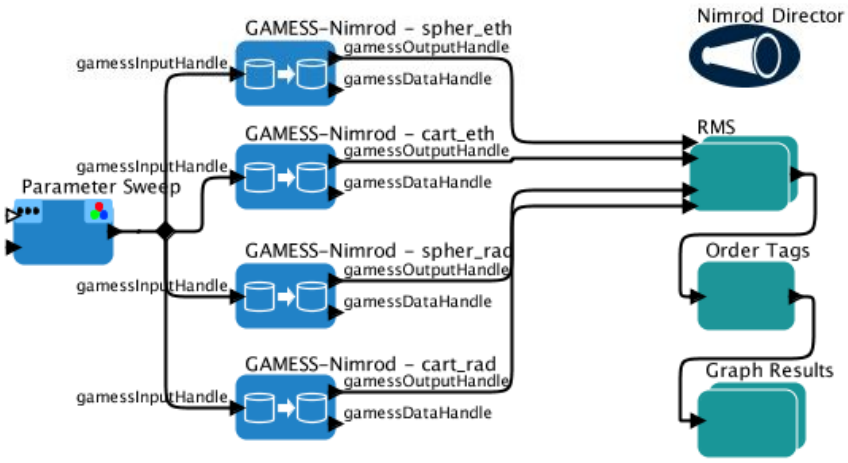


Figure 5. GAMESS Workflow

The workflow shows 4 separate executions of GAMESS for each parameter set. Each execution of GAMESS is shown as a separate GAMESS actor in Figure 5. The Parameter Sweep actor computes all combinations of the A, B, C and D, and builds a record for each of these combinations. They are used as inputs to the GAMESS actors, and the outputs are sent into an actor that computes the Root Mean Square error of the cost function. This is a measure of how well a particular pseudo potential surface fits calculations of a real molecule. The results are re-ordered and plotted using a Graph actor. In the experiment we executed the workflow with three different Directors–SDF, PN and TDA. We highlight the simplicity of changing the execution semantics–all we had to do to change them was to swap out one director on the Vergil canvass and replace it with another one. Figure 6 and 7 show the performance of the workflow under different Directors running on the testbed. All compute resources have little communication latency (<1ms) and the data sets were very small and such had a negligible affect on the experiment time. The SDF director only executed one job at a time, as expected, and took a total of 15:34:20. The PN director was able to exploit the static parallelism in the workflow, and executed up to 4 jobs at a time, taking 05:18:38. The TDA director was able to execute as many jobs together as there were processors available, and took a total of 00:9:45. Thus, the PN ran approximately 3 times faster than the SDF, and the TDA ran nearly 96 times faster than the SDF. The reason the PN times are not exactly 4 times faster is because the director waits on all 4 GAMESS-Nimrods to finish before submitting the next 4, and since each of them takes a different amount of time, time is lost in synchronizing them. The graphs in Figure 6 and 7 also show the number of jobs running at any time. The top trace shows that the TDA Director peaks at 352 jobs, which is the maximum number of available processors on the testbed

at the time. The variation in the number of processors is because there is a variation in job execution times. Thus, whilst the shorter jobs had completed by approximately 4 minutes, a smaller number of longer jobs continued to run for the remainder of the time. This behavior meant that the speedup was less that the peak number of processors. The bottom trace shows the execution profile for the SDF and PN directors. It is clear that the SDF director can only utilize one processor, and the PN uses 4. There were 4 different parameters in the pseudo-potential expression, and by explicitly specifying 4 the PN director can extract this low level parallelism. This allowed us to demonstrate that SDF cannot extract any parallelism even when it is statically specified in the workflow, PN can extract parallelism up to the amount coded statically, and the TDA Directory can extract across data sets.

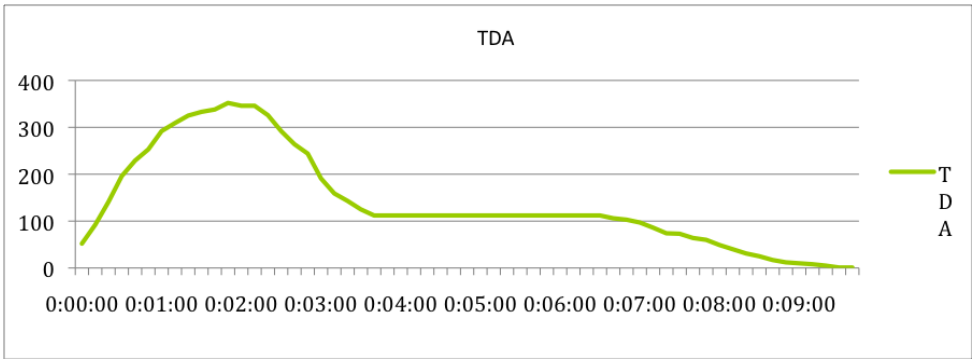


Figure 6. Performance results - TDA

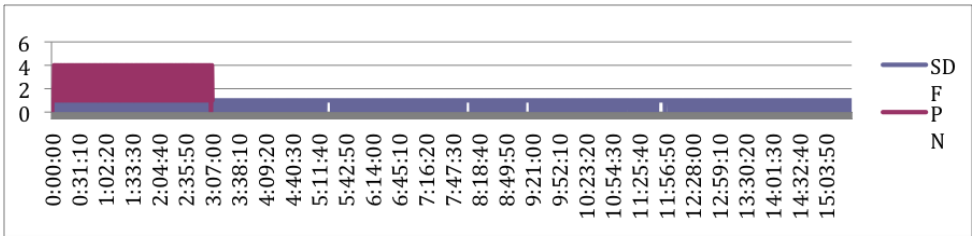


Figure 7. Performance results - PN/SDF

6. Discussion and Comparison of the Approaches

As discussed in the above two sections, these approaches address different requirements for distributed scientific workflows. All provide abstractions to run applications in distributed environments on the production level. In many cases they use and extend the capabilities of used middleware, not being just a wrappers around. Concretely, Serpens and HPC2K use grid middleware including gLite and UNICORE; DDP requires Stratosphere or Hadoop; Nimrod requires Globus middleware; Cloud modules and actors require OpenNebula or Amazon EC2; Globus modules require Globus Toolkit. These approaches are also different in respect to the computation and parallelisation. DDP groups data by patterns, and each

group can be processed independently. Nimrod has combinations of parameters in parameter sweep, and each combination is independent. Serpens and HPC2K run Fortran and C++ codes that use MPI, which allows for application-specific parallelization. Serpens uses also the parameter sweep gLite possibility, submitting one job with the parameter range that divides into many sub-jobs on the middleware level where each of the job is independent. As for the specific implementation level in Kepler (directors/actors/tasks), Nimrod and DDP use directors for job management, whereas Serpens, HPC2K, Globus, Cloud uses specialized actors for distributed execution. HPC2K automatically creates actors to run specific Fortran and C++ applications in distributed environments. DDP uses pattern actors to specify data grouping. All of the solutions enable to run workflows on heterogeneous resources. Table 1 presents further comparison of the approaches including scalability, adaptability, data parallelization.

Table 1. The comparison of the different approaches

	Approach	Scalability	Adaptability	Data parallelization	User requirement
DDP	Director and composite actor	Through DDP execution engine	The same workflow works on both local machine and distributed environment	Support by automatic data partition	Understand the DDP patterns and how to apply them
Serpens	Actor and composite actor	Through gLite and UNICORE middleware	Only works on Grid distributed environments	Partial support by partitioning data and parallel jobs with partitioned data	Provide applications to be submitted via gLite or UNICORE
Nimrod/K	Director	Through Nimrod middleware	The same workflow works on both local machine and distributed environment	Support by automatic data tagging	Understand the data tagging and provide applications to be executed via Nimrod
Cloud	Actor	Through Cloud services	Only works on Cloud distributed environments	Partial support by partitioning data and parallel jobs with partitioned data	Provide applications to be executed via Cloud actors
Globus	Actor	Through Globus middleware	Only works on Grid distributed environments	Partial support by partitioning data and parallel jobs with partitioned data	Provide applications to be executed via Globus actors

In respect to handling of the data, the most specialised is the DDP that is good for large data sizes since it reduces data movement to remove overhead. Serpens (and HPC2K) and Nimrod are using the

solutions provided by corresponding middleware mostly for data movement. Nimrod has no specific actors for data transfer, DDP uses I/O actors to stage in/out data. Serpens has data transfer actors, e.g., GridFTP, and HPC2K wrappers transfer data between the database and remote resources. Cloud module use case uses scp/sftp/gridftp and Globus modules use gridftp. Thanks to the common Kepler platform, all of the approaches can use workflow provenance feature. Also each of the solution addresses the issue of the failure handling, usually using the local databases for saving the check-pointed state of the workflow or of the data.

All these solutions try to hide the complexity, technical details of the middleware and minimize the configuration required by users as much as possible. In this respect, the most user-driven and intuitive solution is the HPC2K that generates whole workflows basing on the application specific parameters, hiding whole the technology part. Also all of the approaches have been validated not only from the pure technical point of view, but also by several application use cases. They are exploited by different scale applications with large computing (thousands of computing jobs per run) and data requirements Approaches like Serpens of Nimrod, shows advantages of using Kepler as a client for the distributed applications, handling failures and gathering additional dynamic informations on underlying infrastructure that are used during submission of jobs.

All these solutions can be installed easily as the Kepler modules. All of the presented methods of distributed execution in Kepler can be used in the same workflow. The overall workflow can be composed of many sub-workflows each of which implements one of these techniques. Additionally, we are building a *ExecutionChoice* actor allows the workflow developer to specify several alternative choices to perform a task, where each choice is presented as a sub-workflow. For example, it may be better to use DDP in some situations, but Serpens in others. The *ExecutionChoice* actor provides a container for these alternative sub-workflows, and the user or the director can decide which sub-workflow to execute. Most of the workflows can be implemented in other workflow systems. But all the actors and workflows would have to be rebuilt based on the target libraries and model specifications. In addition DDP and Nimrod/K create new directors to the corresponding distributed execution. Since director is unique in Kepler, it would be hard to support the same capability and flexibly in non-Kepler systems.

7. Summary and Conclusions

This paper presents different approaches to distributed execution of scientific workflows in Kepler system. This comparison covers the distributed data-parallel framework with Hadoop and Stratosphere, cloud and grid execution with the Serpens, HPC2K, Nimrod/K and Globus actors. The presented technical features and capabilities show that Kepler is a very extensible and flexible platform. Kepler supports a variety of distributed execution techniques on multiple levels and means. Basic application requirements like dealing with the big data or data intensive parallel computing, handling heterogeneous infrastructures, scalability, fault tolerance are addressed by the presented solutions. Some of the solutions focus on specific requirements like DDP that is good for applications with large data size, other take horizontal approaches on supporting multiply middleware stacks on the production level like Nimrod/K and Serpens. In Kepler users can easily choose the best approach to use corresponding solution in Kepler for their particular distributed execution requirements, having also a possibility of mixing multiple solutions at the same time. As an example, thanks to the modularity of the Kepler architecture, all the solutions could be reused in one larger workflow, which is suitable for addressing large modelling plat-

forms. The approaches are illustrated with corresponding scientific application use cases from different scientific fields: bioinformatics, nuclear fusion, astronomy, astrophysics, computational chemistry. We also discussed how to combine different approaches into one workflow in Section 6.

In respect to the future directions, we plan continuing the work addressing requirements of next scientific use cases and maintaining in longer term the technical feasibilities so far. We will also investigate new computing paradigms and middleware could be supported in Kepler, including exascale applications, new cloud "like" computing extensions.

Acknowledgments

The research leading to these results has received funding from different projects and funding schema, including: the European Community's Seventh Framework Programme under grant agreement RI-261323 (EGI-InsPIRE), the European Communities under the contracts of Association between EURATOM and CEA, IPPLM, carried out within the framework of the Task Force on Integrated Tokamak Modeling of the EFDA, the Polish project PLGrid Plus under the contract POIG 02.03.00-00-096/10, NSF ABI Award DBI-1062565 for bioKepler, partial financial support from the Spanish Ministerio de Economia y Competitividad AYA 2010-21766-C03-01 and Consolider Ingenio 2010 CSD2010-00064 projects, and from the Juan de la Cierva programme.

References

[1] Abramson, D., Bethwaite, B., Enticott, C., Garic, S., Peachey, T.: Parameter Exploration in Science and Engineering Using Many-Task Computing, *IEEE Transactions on Parallel and Distributed Systems*, **22**, 2011, 960–973, ISSN 1045-9219.

[2] Abramson, D., Enticott, C., Altinas, I.: Nimrod/K: towards massively parallel dynamic grid workflows, *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, IEEE Press, Piscataway, NJ, USA, 2008.

[3] Alard, C., Lupton, R. H.: A Method for Optimal Image Subtraction, *The Astrophysical Journal*, **503**(1), 1998, 325–331.

[4] Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludaescher, B., Mock, S.: Kepler: An Extensible System for Design and Execution of Scientific Workflows, *IN SSDBM*, 2004.

[5] Altintas, I., Wang, J., Crawl, D., Li, W.: Challenges and approaches for distributed workflow-driven analysis of large-scale biological data, *Proceedings of the 2012 Joint EDBT/ICDT Workshops*, ACM, 2012.

[6] Altschul, S. F., Gish, W., Miller, W., Myers, E. W., Lipman, D. J.: Basic Local Alignment Search Tool, *Journal of Molecular Biology*, **215**(3), 1990, 403 – 410, ISSN 0022-2836.

[7] Battré, D., Ewen, S., Hueske, F., Kao, O., Markl, V., Warneke, D.: Nephele/PACTs: A Programming Model and Execution Framework for Web-Scale Analytical Processing, *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, ACM, New York, NY, USA, 2010, ISBN 978-1-4503-0036-0.

[8] Cabellos, L., Campos, I., del Castillo, E. F., Owsiak, M., Palak, B., Płóciennik, M.: Scientific workflow orchestration interoperating HTC and HPC resources, *Computer Physics Communications*, **182**(4), 2011, 890 – 897, ISSN 0010-4655.

[9] Chapman, B., Jost, G., van der Pas, R., Kuck, D.: *Using OpenMP: Portable Shared Memory Parallel Programming*, The MIT Press, Cambridge, MA, USA, 2007.

[10] Churches, D., Gombas, G., Harrison, A., Maassen, J., Robinson, C., Shields, M., Taylor, I., Wang, I.: Programming scientific and distributed workflow with Triana services: Research Articles, *Concurr. Comput. : Pract. Exper.*, **18**, August 2006, 1021–1037, ISSN 1532-0626.

[11] Crawl, D., Altintas, I.: A Provenance-Based Fault Tolerance Mechanism for Scientific Workflows, *Provenance and Annotation of Data and Processes (IPAW 2008, Revised Selected Papers)* (J. Freire, D. Koop, L. Moreau, Eds.), 5272, Springer, 2008.

[12] Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters, *Communications of the ACM*, **51**(1), 2008, 107–113.

[13] Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Patil, S., Su, M.-H., Vahi, K., Livny, M.: Pegasus: Mapping Scientific Workflows onto the Grid, in: *Grid Computing* (M. Dikaiakos, Ed.), vol. 3165 of *Lecture Notes in Computer Science*, chapter 2, Springer Berlin / Heidelberg, Berlin, Heidelberg, 2004, ISBN 978-3-540-22888-2, 131–140.

[14] Gropp, W., Lusk, E., Skjellum, A.: *Using MPI: Portable Parallel Programming with the Message Passing Interface*, Scientific And Engineering Computation Series, 2nd edition edition, MIT Press, Cambridge, MA, USA, 1999.

[15] Gu, Y., Grossman, R.: Sector and Sphere: The Design and Implementation of a High Performance Data Cloud, *Philosophical Transactions of the Royal Society A*, **367**(1897), June 2009, 2429–2445.

[16] Guillerminet, B., Plasencia, I. C., Haefele, M., Iannone, F., Jackson, A., Manduchi, G., Plociennik, M., Sonnendrucker, E., Strand, P., Owsiak, M.: High Performance Computing tools for the Integrated Tokamak Modelling project, *Fusion Engineering and Design*, **85**(34), 2010, 388 – 393, ISSN 0920-3796, Proceedings of the 7th IAEA Technical Meeting on Control, Data Acquisition, and Remote Participation for Fusion Research.

[17] Hull, D., Wolstencroft, K., Stevens, R., Goble, C., Pocock, M. R., Li, P., Oinn, T.: Taverna: a tool for building and running workflows of services, *Nucleic Acids Research*, **34**(suppl 2), 1 July 2006, W729–W732.

[18] Kacsuk, P., Sipos, G.: Multi-Grid, Multi-User Workflows in the P-GRADE Grid Portal, *Journal of Grid Computing*, **3**(3), September 2005, 221–238, ISSN 1570-7873.

[19] Köhler, S., Riddle, S., Zinn, D., McPhillips, T., Ludäscher, B.: Improving workflow fault tolerance through provenance-based recovery, *Proceedings of the 23rd international conference on Scientific and statistical database management*, SSDBM'11, Springer-Verlag, Berlin, Heidelberg, 2011, ISBN 978-3-642-22350-1.

[20] Kozlovszky, M., Karoczkai, K., Marton, I., Balasko, A., Marosi, A., Kacsuk, P.: ENABLING GENERIC DISTRIBUTED COMPUTING INFRASTRUCTURE COMPATIBILITY FOR WORKFLOW MANAGEMENT SYSTEMS, *Computer Science*, **13**(3), 2012, 61–78.

[21] Moretti, C., Bui, H., Hollingsworth, K., Rich, B., Flynn, P., Thain, D.: All-Pairs: An Abstraction for Data-Intensive Computing on Campus Grids, *IEEE Transactions on Parallel and Distributed Systems*, **21**, 2010, 33–46, ISSN 1045-9219.

[22] Scott, B. D.: Free-energy conservation in local gyrofluid models, *Physics of Plasmas*, **12**(10), 2005, 102307.

[23] Thain, D., Tannenbaum, T., Livny, M.: Distributed computing in practice: The Condor experience, *Concurrency Computation Practice and Experience*, **17**(2-4), 2005, 323–356, Cited By (since 1996) 440.

[24] Wang, J., Altintas, I.: Early Cloud Experiences with the Kepler Scientific Workflow System, *Procedia Computer Science*, **9**, 2012, 1630–1634.

[25] Wang, J., Altintas, I., Hosseini, P. R., Barseghian, D., Crawl, D., Berkley, C., Jones, M. B.: Accelerating Parameter Sweep Workflows by Utilizing Ad-Hoc Network Computing Resources: An Ecological Example, *IEEE Congress on Services*, IEEE Computer Society, 2009.

[26] Wang, J., Crawl, D., Altintas, I.: Kepler + Hadoop: A General Architecture Facilitating Data-Intensive Applications in Scientific Workflow Systems, *Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science*, WORKS '09, ACM New York, NY, USA, Portland, Oregon, 2009.

[27] Wang, J., Crawl, D., Altintas, I.: A framework for distributed data-parallel execution in the Kepler scientific workflow system, *Procedia Computer Science*, **9**, 2012, 1620–1629.

[28] Wang, J., Korambath, P., Kim, S., Johnson, S., Jin, K., Crawl, D., Altintas, I., Smallen, S., Labate, B., Houk, K.: Facilitating e-Science Discovery Using Scientific Workflows on the Grid, *Guide to e-Science: Next Generation Scientific Research and Discovery*, 2011, 353–382.