

Persistent Locality Management of Scientific Application Workflows

Lamine Aouad
University of Limerick
Centre for Next Generation Localisation
Ireland
lamine.aouad@ul.ie

Tahar Kechadi
University College Dublin
Ireland
tahar.kechadi@ucd.ie

Serge Petiton
University of Science and Technology of Lille
France
serge.petiton@lfl.fr

Abstract

The huge data requirements of large nowadays applications in science and engineering make optimised and scalable data placement mechanisms an essential need. For this purpose, we propose a scheduling scheme based on an efficient data locality management for data-intensive workflows. Transfer and placement decisions are made based on constructions in the workflow, representing inter-relationships between inputs and outputs at its different levels. When running large applications, most of the input data would not be shipped, keeping the data close to the jobs, and resulting on much less communication and transfer overheads. We have implemented these techniques for the YML workflow system. This paper presents results showing a substantial improvement in the performance of many interdependent multi-level workflows through these data placement optimisations.

1 Introduction

A wide range of applications in science, industry, and business are taking advantage of large distributed computing and storage facilities available over the networks; grids or classic datacentres, resources federations, or more recently clouds. In the last few decades, a tremendous amount of work has been carried out at different levels of distributed system architectures ranging from network protocols and middleware, to programming paradigms and models. Significant work has then been conducted in self-management of applications on large distributed systems. These include the

workflow concept, and process or task management. The latter has played a key role in application development, independently of the targeted execution systems, using high-level representations.

Nowadays applications often involve huge amount of data being processed. To support the scale of these applications, not only very large sets of resources are needed, but also different optimisations on how to efficiently map the application onto the resources. Indeed, running and managing the execution of large-scale applications have many challenges. These include jobs' control and management, scheduling and data mapping, specific characteristics of the data management requirements such as consistency, reliability, space allocation and management, access restrictions, the communication protocols, etc. These issues are very important and have a direct impact on the system performance. Recently, the cloud has emerged as a viable option for grid computing, offering interesting solutions to some of these issues, and providing an ideal infrastructure for large scale applications of resource- and data-intensive nature.

On the other hand, the data composition and placement are still widely considered as a side effect of the computation. In this paper, we describe automated data placement policies to deal with issues in running data-intensive workflows. We provide a framework for deploying scalable grid scientific applications on the grid, which could also leverage the cloud service capabilities, in queuing, storage, and elastic computing, for efficient data persistent implementations.

Motivations & contributions

The high-level workflow representations are usually mapped onto the underlying middleware taking into account only some information about the computational jobs. The choice and allocation of the processing locations are, indeed, a very complex task leading to NP-completeness for data distribution and task scheduling in most cases, even with simple applications and very restrictive hypotheses. What we are targeting in this paper is a mapping that takes into account data placement and locality management as a primary concern. The traditional feasibility constraint may also be taken into account depending on the platform, i.e. access to the remote site / node as well as hardware resources availability. This will imply different constraints in the case of different distributed arrangements. The main contributions of this paper are:

- An extended YvetteML¹ (7) language with new scheduling policies,
- A compiler implementing the additional placement policies, which uses an XML-based modelling similar to CoG², and
- A persistent locality management execution provider.

This paper extends our previous work presented in (1) for these three aspects in relation to the YML system. Note that in (1) we used a simple translator to CoG, from a predefined workflow specification language similar to YML. The jobs definition formalism was similar to the one used in Condor's DAGMan, and the resulting representation was executed on a grid with a Globus installation.

The paper is organised as follows. In the next section, we present some existing workflow frameworks. Section 3 presents the YML architecture and the additional persistent locality management mechanisms newly implemented. Section 4 presents details about the new system components and implementation. Case studies and some results are presented in section 5. Finally, section 6 gives concluding remarks along with future work directions.

2 Related Work

Significant research work has already been conducted on workflow, and process/task arrangement systems and frameworks for scientific computing. This includes a large number of tools, with different mapping

¹YvetteML is the language used in the YML workflow management system.

²Commodity Grid, used within the Globus Toolkit.

capabilities, such as the DAGMan meta-scheduler in Condor (14), Askalon (8), CoG (15), YML (7), GridAnt (11), among many others. Many description languages for declaring the jobs composition have also been proposed, and many of them have an XML-based modelling. Their architectures are usually composed of a user interface or language tools and a workflow execution engine, which controls the execution of the jobs.

Although almost all the existing workflow approaches are only dealing with the assignment of jobs to nodes independently of their data requirements, few data-oriented schedulers have been proposed. These include the Stork framework (12). Stork considers data placement independently of the computational jobs and is exclusively dedicated to the data transfer task. Higher level planners such as Pegasus (5) or Chimera (10) can interact with Stork and DAGMan at the same time (for data scheduling and jobs scheduling respectively), but they still separate computational tasks and data placement processes by sending them either to the DAGMan queue or to the Stork queue. Note that the Condor batch scheduler does not preserve the data locality. There are other systems, which provide distributed processing and storage management of very large datasets, but they do not integrate the concept of the application workflow. These include DataCutter (3), OceanStore (4), or the DataGrid project (13). In this paper, we are more focused on what we call *a locality-based mapping*.

3 YML

YML (6) (7) is a workflow management framework. It has a language called *YvetteML*, and implements different middleware providers; currently supporting XtremWeb and OmniRPC. In comparison to other workflow systems, YML offers an additional level of abstraction by implementing a component-oriented mechanism. The components and catalogs model bring more flexibility in applications design and graphs description. Reusability is also a very important feature of this model. YML uses directed general graph (DGG) description model which expresses loops, iterations and branching, in comparison to the directed acyclic graph (DAG) description used in DAGMan or GridAnt for instance.

YML intends to be middleware-independent and proposes useful generic abstractions for workflow representation on large-scale grids. YML forms the basis of this study. Its general architecture is presented in Figure 1. Our aim is not to define a yet another specification language and workflow management framework,

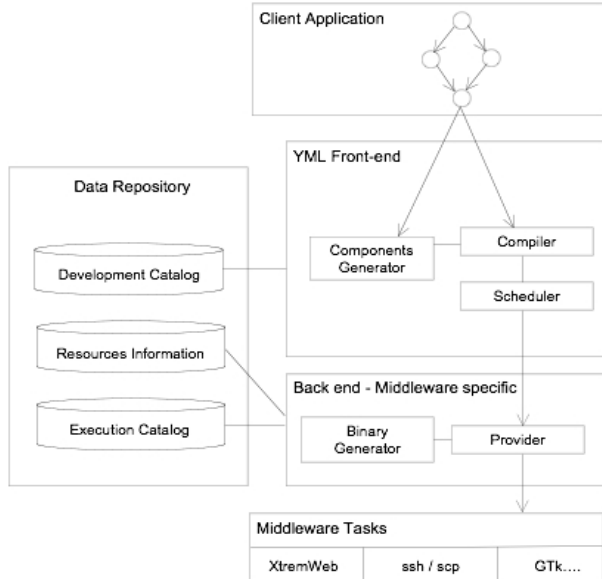


Figure 1. YML architecture

but to take advantage of existing formalisms to propose a simple and flexible representation taking into account a set of specific data scheduling policies. Our contribution is then to improve data locality management for applications developed using YML, and to enhance the existing language and system with additional jobs and data placement features. Data placement capabilities have been added to the specification language describing the client application. In addition, a new compiler automatically generating persistent placement decisions based on predefined patterns, and a provider supporting locality management, have been implemented.

3.1 Persistent locality management

In the literature, various definitions of data persistency can be found. In Java for instance, a persistent object refers to the information retrieval, even after the end of an application. In the OceanStore system mentioned earlier (4), the persistent storage refers to the consistency, high-availability, and durability of storage through an infrastructure comprising of untrusted servers, such as in desktop grids or volunteer computing. This notion usually refers either to the reliability or services continuity in some systems, or *data survival* in others.

In this paper, what we refer to as persistent locality management is related to data placement techniques. These currently include migration anticipation and data labelling (or data nailing). Jobs clustering and replications techniques will also be considered.

This is intended to maximise the migration anticipation rate since the data locality scheme, and then the labelling scheme, is predetermined for a given application, i.e. is known at specification time. For each application, the persistent storage corresponds to some data distribution and placement optimisation in order to minimise the communications. Tasks scheduling is then carried out according to this placement and the data flow.

On the other hand, the implementation of the persistent locality management can be done implicitly or explicitly. In the latter case, application programmers have to manage data locality explicitly, using handles or functions for instance. The implicit case can be related to a global file system or a persistent scheduler, as the one presented here, which is based on interdependencies patterns in the workflow. An example is given below.

Example:

Running distributed applications requires moving inputs data, which could be outputs of other jobs, from / to the processing nodes at various levels of the execution workflow. Consider one of the tasks 3 in Figure 2 (representing the Gauss-Jordan matrix inversion use case). A given task 3 in the workflow has 3 input blocks; the outputs of 2 tasks at the previous level and another independent input block (not an output of previous execution levels). In these cases, one of the execution pattern would be to execute the task 3 in question on either of the locations used by the dependent tasks (at the previous / 2^{nd} level). In addition, the independent input will also be moved to that location beforehand. This latter pattern is called migration anticipation or pre-deployment depending on the context. The former is referred to as data labelling or referencing. The use cases below give more details about this application, and an astronomy application workflow.

4 Components & implementation

The persistent version of YML is developed in C++ and uses the same set of libraries and tools³. The current persistent locality management provider uses ssh/scp to transfer data and execute jobs.

4.1 Specification language

The YvetteML specification has been extended by adding new scheduling policies as features of the language. Two new constructs were added: *deploy* and

³Autotools, Yacc&Bison, and the libutil library.

transfer. These are processed by the new compiler as the existing *compute* construct. The *deploy* construct expresses jobs mapping on specific locations. The *transfer* construct expresses specific data placement requirements. It transfers data from any two locations on the system. Note that the placement decisions are generated automatically based on predefined constructions, which represent inter-relationships patterns between inputs and outputs at different levels of the workflow as described above. The new added language constructs, *deploy* and *transfer*, are only used to deal with information about the location of the data referenced by a given job, or the location or characteristics of the environment, software, libraries, etc., that a user may have.

4.2 Compiler

The extended compiler uses a new class `YvetteXmlWriter` derived from the original `YMLTaskGenerator`. The class searches the application tree, registers new tasks, and stores the relevant informations into two additional XML files, using the `XMLSerializer` class (class for XML file writing). The first new file is the application described in a CoG-derived XML representation. It uses several elements from the XML CoG syntax (15). There are several common elements: `project`, `set`, `sequential`, `parallel`, `string`, and `execute`. The `project` element describes the global application element. The `set` element describes the variable assignment element. The `sequential` & `parallel` elements describe synchronisations. The `string` element describes the character string type. Several other elements are introduced with the new XML representation: `integer`, `real`, and `argument`. The `integer` and `real` elements replace the `number` element, as the numerical type is always indicated in the YvetteML language so we can avoid any ambiguity. The `argument` element is given as a child of the `execute` element.

The compiler generates a new file describing the tasks dependencies. It consists of one main `tasksinfo` element containing several `task` elements, each one of them containing its own inputs, outputs and conditions information. This is the basic representation allowing the persistent placement. Indeed, the placement patterns are represented by specific successions of interdependent `task` elements. The `parameter` element contains information about input or output parameters. The `name` element describes a parameter name. The `mode` element tells us whether a parameter is used as input, output or both. The `type` element describes the parameter's type. The `conditions` element de-

scribes, using boolean elements `and`, `or` and `xor`, the current task's precondition. In addition to the output described above, the compiler finalises its process by copying all the required components binaries into the application's destination folder.

4.3 Provider

A persistent ssh/scp provider has been implemented. The purpose of the provider is to execute the XML application files generated by the extended compiler. It translates the features of scheduling and placement decisions into ssh/scp jobs. The main class `Provider` uses the `ConsoleApplicationAdaptor` class derived from the `ApplicationAdaptor` class (enabling the quick creation of applications and the efficient management of their input and output). The provider implements an XML parser within the `XmlAppHandler` class based on the libutil `XMLParser` and `XMLHandler` classes. The class `VarPool` manages variable assignments by registering all declared variables in a map structure with their names and types, and then creates the file containing the corresponding data via the libutil `Pack` class.

5 Case studies

We present two case studies. The first one is a basic application in numerical analysis, and the second one represents a scientific application in the astronomy area. These two applications have fine-grained and data-intensive properties. Both applications are divided into several interdependent jobs. Each level of the workflow represents a component. The workflow are expressed using the extended YvetteML language. These specifications are compiled into the CoG-derived XML-based workflow representation, and the tasks elements description used by the provider. Note that both representations are aimed at implementing the placement techniques automatically. The former representation can also be considered to be fed to a grid with a Globus installation (9).

5.1 Gauss-Jordan elimination workflow

The Gauss-Jordan elimination is a particularly interesting case study as it has a uniform workload throughout the factorisation phase. This makes the evaluation, as well as the impact of the data placement management techniques, much more noticeable. We will test block-based matrix inversion using the Gauss-Jordan elimination. Figure 2 shows the workflow. There are three components; inversion of the

pivot block in 1, blocks product in 2, and blocks triadic in 3.

If the matrix is partitioned into $n \times n$ blocks, the workflow is made up of n steps. Each step is composed of a block inversion, $2(n - 1)$ blocks products, and finally $(n - 1)^2$ blocks triadics. We will take into account only intra-step dependencies. Indeed, inter-steps dependencies can also be considered. More details about this algorithm can be found in (2). For this algorithm, the persistent locality management may save $(2n - 1)$ block transfers at the first level, and $2(n - 1)^2$ block transfer at the second level (see Fig. 2) using the proposed mechanisms.

5.2 Montage workflow

The Montage application deals with constructing custom science-grade astronomical image mosaics on demand. Figure 3 shows a small Montage workflow. The workflow contains 7 different jobs corresponding to each level in the workflow graph. This is an interesting case study since it presents the same number of jobs at the 1st and 5th levels, and larger number of jobs at the second level. If the number of jobs at the 1st level is N , the workflow may save at least $2N$ data outputs' transfers (at the 2nd and 5th levels) using the proposed persistent locality management.

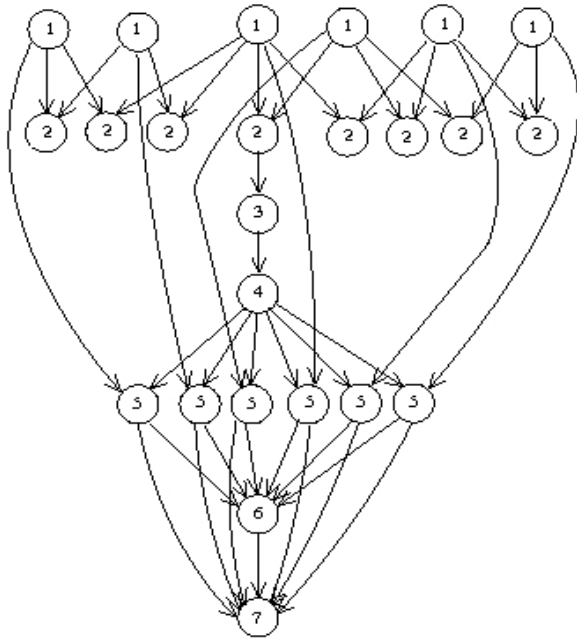


Figure 3. A small Montage workflow

5.3 Performance analysis & discussion

This section presents the testbeds and setup of both case studies. The computing platform is a set of 10 dual-core CPUs (2.40GHz and 4GB memory), interconnected by a local Ethernet network. For the Gauss-Jordan inversion, we consider a homogeneous data distribution among the processing nodes, with a block size of 3000, and matrix size of 9000 ($n = 3$). For the Montage application, we consider an initial input of about 30MB for the input files (images), with N equal to 180, and up to 1254. Recall that N is the number of jobs marked 1 in Figure 3.

In these experiments, we consider only I/O requirements within each level of the workflows. Indeed, the computing platform is not dedicated, and its state during computation may alter the evaluation. This aims at minimising its effect on the measurement of the persistent placement impact. Note that other additional overheads related to tasks preparation and scheduling for instance also occur. Indeed, the placement of jobs is explicit for the persistent placement version, which is not the case otherwise, while the mapping of the initial version does not take into account the data location. These overheads can also be substantial and may badly affect the applications performance.

Now, the additional transfer overheads for the Gauss-Jordan inversion is about 21 minutes. The persistent placement version needs about 30 minutes compared to 51 for the traditional version, which represents more than 40% decrease in the data transfer time. Note that we expect this to be more important while considering longer processing times, since this will allow more important overlapping of pre-deployment (which represents $2n$ transfers for this algorithm) and the computations. The same applies for larger partitioning (n with a higher value). In the montage workflow, the additional overhead was from 16 minutes for N equal to 180, up to 2 hours for N equal to 1254. These results highlight the utmost importance and relevance of persistent placement for data-intensive applications. The same experience (montage workflow) reported in (1) shows much less transfer overheads on a dedicated and closed high performance computing platform using a Gigabit Ethernet connexion. Indeed, this is highly dependent on the execution platform, the workflow system and underlying middleware / provider, the size of data files, and the number of levels of the workflow. However, the gain has shown to be substantial using different setups and applications, in this paper as well as in our previous work in (1).

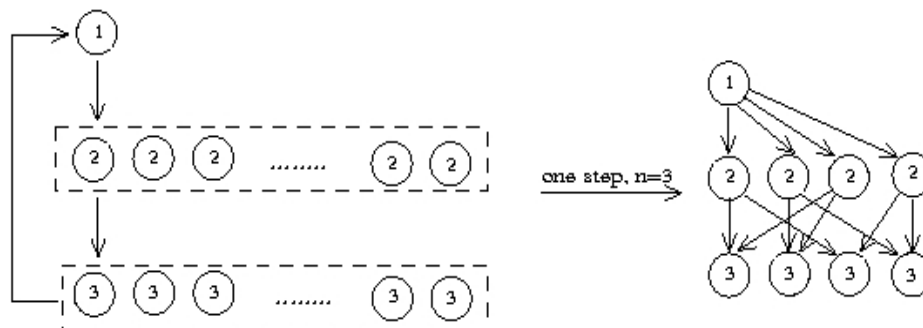


Figure 2. The Gauss-Jordan matrix inversion workflow & intra-step dependencies for $n = 3$.

6 Conclusion

In this paper, we presented a persistent data locality management system for scientific application workflows. We tackled the implementation of additional persistence capabilities within the YML workflow management system. This work also highlights the need of more efficient and persistent placement and storage for large-scale and data-intensive applications. The results show a substantial decrease in the overheads related to data transfer operations. Considering more analysis of the hierarchy of the transfer overheads, and additional providers are some of the future work. Also, the implementation of a complete cloud-based YML solution is also of interest. This involves interfacing YML with basic components in current clouds in terms of storage, queuing, allocating new resources, in a comprehensive framework interconnecting these components and ensuring proper messaging, and job and data flow.

References

- [1] L. M. Aouad, N. A. L. Khac, and M.-T. Kechadi. Persistent workflow on the grid. In *The 2008 IEEE Asia-Pacific Services Computing Conference, IEEE APSCC 2008*.
- [2] L. M. Aouad and S. Petiton. Grid-based programming paradigm for distributed linear algebra applications. *Multiagent and Grid Systems. To appear*.
- [3] M. Beynon, R. Ferreira, T. M. Kurc, A. Sussman, and J. H. Saltz. Datacutter: Middleware for filtering very large scientific datasets on archival storage systems.
- [4] D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, B. Zhao, and J. Kubiawicz. Oceanstore: An extremely wide-area storage system. Technical report, Berkeley, CA, USA, 2002.
- [5] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Sci. Program.*, 13(3):219–237, 2005.
- [6] O. Delannoy, N. Emad, and S. Petiton. Workflow global computing with yml. In *GRID '06: Proceedings of the 7th IEEE/ACM International Conference on Grid Computing*, pages 25–32, 2006.
- [7] O. Delannoy and S. Petiton. A peer to peer computing framework: Design and performance evaluation of yml. In *HeteroPar 2004, the third Int. Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks*, 2004.
- [8] T. Fahringer, A. Jugravu, S. Pllana, R. Prodan, J. Clovis Seragiotto, and H.-L. Truong. *Concurr. Comput. : Pract. Exper.*, (2-4):143–169.
- [9] I. Foster. Globus toolkit version 4: Software for service-oriented systems. *Journal of Computer Science and Technology*, 21(4), 2006.
- [10] I. Foster, J. Voeckler, M. Wilde, and Y. Zhao. Chimera: A virtual data system for representing, querying, and automating data derivation. In *The 14th Conference on Scientific and Statistical Database Management, 2002*.
- [11] k. Amin, G. von Laszewski, M. Hategan, N. J. Zaluzec, S. Hampton, and A. Rossi. Grid. In *Proceedings of the 37th Annual Hawaii International Conference on System Science*.
- [12] T. Kosar and M. Livny. A framework for reliable and efficient data placement in distributed computing systems. *J. Parallel Distrib. Comput.*, 65(10):1146–1157, 2005.
- [13] H. Stockinger, F. Donno, E. Laure, S. Muzaffar, and P. Kunszt. Grid data management in action: Experience in running and supporting data management services in the eu datagrid project. In *Computing in High Energy and Nuclear Physics, 24-28 March 2003, La Jolla, California, 2003*.
- [14] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: The condor experience. *Concurrency and Computation: Practice and Experience*, 2004.
- [15] G. von Laszewski, I. Foster, J. Gawor, and P. Lane. A java commodity grid kit. *Concurrency and Computation: Practice and Experience*, 13(8-9):645–662, 2001.