QUERY PROCESSING OVER THE GRID: THE ROLE OF WORKFLOW MANAGEMENT

E. Floros, G. Kakaletris, P. Polydoras and Y. Ioannidis
National and Kapodistrian University of Athens
Department of Informatics and Telecommunications,
Athens, GREECE
floros, gkakas, p.polydoras, yannis@di.uoa.gr

Abstract gCube Information Retrieval Engine differentiates from federated search. Data and services are scattered over the infrastructure instead of being contained in confined sub-sections, made accessible via narrow interfaces. Grid, where gCube runs, is not meant for interactive work, but offers a vast pool of resources for processing large amounts of information. In such domain it comes as natural consequence the preference of machinery that can employ the afforementioned resources for offering a different range of services, over a traditional search facility.

Keywords: Optimisation, Process Scheduling, Information Retrieval, Grid Computing, Workflow Execution

1. Introduction

In traditional *Information Retrieval (IR)*, managed or federated, systems exploit domain-specific constructs targeting the needs of a particular application scenario. Query execution on these systems generally assumes that the way to exploit resources is predefined, while the optimal roadmap to obtain results is roughly known a-priori to the system (in contrast to RDBMSs), with minimal potential deviations driven from cardinalities and resource availability - in face of failures and load.

These, otherwise stable assumptions, do not hold in case of *Virtual Research Environments (VREs)* [2]. In these environments, the size and type of information managed and the ways it can be exploited, might vary significantly due to different user / application needs, a challenge which grows under the perspective of hosting them on a dynamic, uncontrolled vast environment, such as a computational grid.

Under the herein described *gCube* framework [1], the aforementioned challenges are handled through the innovative approach of dynamic composition and execution of workflows of services that, step-by-step, carry out the individual tasks implied by an information retrieval request, in a near optimal manner.

The rest of this paper is structured as follows: in section 2 we present the rationale behind the complex Information Retrieval mechanism of gCube and background information on the framework's fundamental concepts and technologies. We describe in more details the gCube framework, in section 3, and we unveil the details of how a simple user query is transformed into a standard-based graph of service invocations and subsequently gets executed. The system evaluation is presented in section 4 along with intentions for future work and system enhancements.

2. Background Information

2.1 gCube

gCube is a middleware for the realisation of Virtual Research Environments on top of a grid-enabled infrastructure. Being multidisciplinary in nature, spreads over domains which, among others include: Knowledge Management / Information Retrieval, Data Management / Data Processing, Distributed Computing, Resource Management, Service Semantics Definition and Service Orchestration. In the context of this paper we introduce Workflows as a mechanism that enables Information Retrieval on the Grid Computing domain:

In **Grid Computing**, a Computational Grid [4] represents a vast pool of resources, interconnected via networks and protocols, that form the substrate where storage and computational demands can be satisfied at large, in a cross-organisational scope. The enabling software (middleware) that brings the infrastructure together and its capabilities can vary significantly, yet, it is generally expected that it offers mechanisms to allow infrastructure and security management, such as information services, authentication mechanisms etc.

Information Retrieval is an empowering concept [5], of Knowledge Management, satisfied nowadays mainly through two architectural models: autonomous systems and federated ones, the latter being a model which fits the SOA paradigm and leaves space for independent realisations of local services. Despite the hard to beat performance of dedicated systems, be it autonomous, federated, network (web) or desktop based, bringing Information Retrieval to the Grid is quite attractive because, on this new joint-domain, VREs can exploit:

- shared, generic resources, with significant lower cost than dedicated infrastructures, for hosting their Knowledge Banks
- large capacities for on demand processing of information beyond the typical mechanisms of the IR domain

- opportunities for exploiting highly demanding IR techniques over the hosted content, such as, but not limited to, Feature Extraction and Query By Example over multimedia content

Under these assumptions, the provision of a standards-based, open system that allows arbitrary realisation of Information Processing scenarios, utilising resources not known a-priori, seems to be fundamental for exposing the benefits of IR over the Grid.

gCube middleware attempts to capture the above-mentioned requirements via a set of specifications and an entire mechanism that transforms user-queries into workflows of service invocations and subsequently manages all the details of communication and execution. It builds on top of OGSA, WS-* and WSRF specifications and exploits the Globus Toolkit 4 provided WS-Core implementation of WSRF. It offers the means for building, managing and running an infrastructure that hosts VREs and a complete set of tools for hosting data and information and exploiting them efficiently in the Knowledge Management domain [3].

Among its core constructs and contents, the Information System, is the glue that keeps the infrastructure together by offering the system-wide registry of gCube and the machinery to interact with it. The *gHN (gCube Hosting Node)*, corresponds to the storage / computing resource of the infrastructure, being correspondent to a container in a physical machine. Registring, exploring, monitoring and running elements on the infrastructure, all pass through these two tightly collaborating elements.

On top of gHNs live the Resources, which can be fairly diverse in nature. Services, Web Service Resources, software components, or even "files" can be resources that may be published and consumed. Every publishable entity exposes a profile in the IS, which renders the set of information upon which it is discovered by its potential consumers. Due to the aforementioned heterogeneity of resources in this infrastructure, the profiles are classified in several subclasses, while more generic ones exist for arbitrary usage. Among these profiles we distinguish the Running Instance and the WebResource profiles, which describe entities that contain "executable logic", i.e. web services, under the WS Resource perspective.

2.2 Services and Resources

gCube is inherently Service Oriented. Service Oriented Architecture (SOA) [6]) is a model ideal for the realisation of large scale systems, essentially distributed. Composing individual entities (services) that encapsulate state, physical resources and logic behind narrow interfaces, is achieved via the numerous protocols on which service interaction is based. The publisher / subscriber model, for declaring the availability and the requests for consumption of Services, is essential for rendering a Service Oriented System able to modify its internal flow of information and control and take advantage of the composition in a manner other than statically binding pieces of logic together.

Service Oriented conceptualisation has received wide acceptance and a wide proliferation after the emergence of XML based technologies of Web Services / HTTP / SOAP stack ([8]). The grid computing community, through its official standardisation body, the *Open Grid Forum* (OGF), having early recognised that computational grids should be build on service oriented foundations, has proposed the *Open Grid Services Architecture* (OGSA) [10] as the blueprint for building and deploying grid tools and infrastructures. The cornerstone of this architecture is the Web Services Resource Framework (WSRF) [9].

Web Services in principle are stateless. They avoid formalising the handling of state among interactions, letting the designer apply home-grown techniques for offering feature rich systems, in a similar way this is handled in the stateless HTTP world. This gap in specification is filled by the WSRF, a set of concepts ([7]), specifications, practices and tools, which do not specify just a formal way for stateful service interactions but defines the Web Service Resource entity as an undividable, identifiable, discoverable and utilisable composition of logic, soft-state and physical resources with a life-time. WSRF builds upon Web Service specifications, like WSDL, WS-Notifications , WS-Addressing , and adds new ones consequence of its new concepts (XML infoset , WS-ResourceProperties , WS-Resource Lifetime, ERP etc) ([8]), [9]). These are currently incarnated in reference implementations such as WS-Core (included with the Globus Toolkit 4 [11]) that provides the basic tooling for building Web Service grids. WS Resources are integral part of the gCube architecture and raise a number of challenges for the workflow composition and execution engine.

2.3 Workflows on Grids

As will be shortly shown, IR queries in gCube are transformed into workflows for execution on the Grid. Naturally Workflow management and processing have attracted tremendous interest in the context of computational grids and scientific computing [14], being employed by almost every non-trivial computation / data-intensive application. Under this observation, reuse is quite desirable, thus tools and abstractions are needed to define, execute and monitor such workflows.

In the business world, WS-BPEL (Business Process Execution Language for WebServices) [12] has become the standard notation for defining workflows (or "processes") of web services. The standard is supported by many commercial and open-source platforms providing tooling for programming, deploying and executing BPEL processes. Yet, in the context of scientific computing and computational grids, BPEL has witnessed limited proliferation till now. This is mainly due to the fact that the standard unit of execution currently on the large grid infrastructures, remains the job, rather than the service.

A job encapsulates an application and its dependencies, that is autonomously being executed on a cloud of physical grid resources. The execution of such workflows typically is based in the implicit flow of data between jobs, which can have sequential or parallel control dependencies.

This model of workflow definition is provided by almost all of the popular workflow tools. For instance Condor [13], one of the most well known middleware for setting up campus-wide and corporate-wide grids, provides DAGMan (Directed Acyclic Graph Manager). The same approach has been adopted by gLite middleware [18] developed in the context of EGEE project [17], and by P-Grade which builds above the previous tools and provides a portal and workflow execution engine for inter-grid workflows.

On the other hand there exist high level tools that break the barriers of DAGs and go beyond jobs, either by supporting pure web service integration or hybrid solutions, where web services wrap local applications, remote applications or complete jobs. Nevertheless, the majority of these tools don't use any standardised workflow representation and provide capabilities for defining and running only static workflows. Two good examples of generic engines are the *Taverna* and *MOTEUR* which both use *SCUFL* (Simple Conceptual Unified Flow Language) as the workflow description language. Other examples of traditional scientific workflow tools, with web service extensions, include *Triana*, *Kepler* and *Karajan*. For more details on the above tools/activities see [14].*K-Wf* [15] uses stateful WSRF services as the main unit of execution and supports knowledge based execution in which the workflow enactment is based on stored ontologies of the subsystems involved. It also uses its own workflow definition language based on Petri-nets.

OGSA-DAI (OGSA-Data Access and Integration) [16], can be also considered as data-centric workflow execution framework. It captures three aspects of distributed data-management: acquisition / delivery, processing (transformations) and transportation. Beside the built-in constructs, extending the framework allows custom application logic to be invoked in all sections of the workflow.

2.4 Service Composability

The composability of the various web services into a meaningful workflow is an essential yet not easily tackled, multifaceted issue:

Service semantics: The roles to be undertaken by an entity in a workflow is an integral property of the entity. In typical SOA, consumers of services are a-priori explicit on their requirements. Yet dynamic service composition sce-

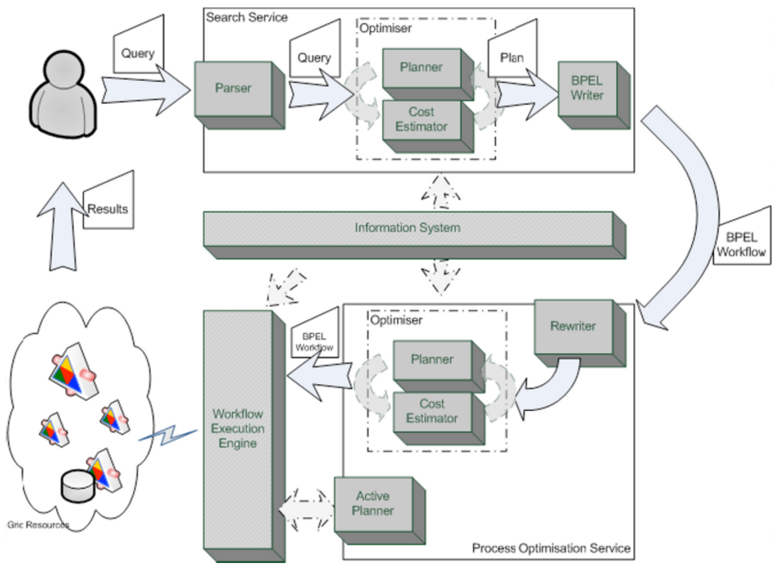


Figure 1. gCube Workflow Composition and Execution Architecture

narios require sufficient flexibility in their selection mechanisms, which can be achieved by describing the entities at a higher level of abstraction. Service Semantics capture not only the internal operation of an entity, but also its interfacing (parameters / results).

Communication mechanism: The various entities of a workflow communicate via technologies that are determined by their nature. For instance, in Web Services, communication is based upon W3C specifications, which, unfortunately do not supply sufficient support for managing large data sets or streams, which pushes towards proprietary implementations.

Data integration: Normalising the data for exchange among the various stake-holders is prerequisite of composition. Ontologies can be employed for resolving schema mappings, yet in several cases simpler solutions (direct mapping) can be applied. Data integration can be conceived as a sub case of service semantics, yet in applied systems it is often realised separately with simpler mechanisms (transformations).

3. Query Processing in gCube

In Figure 1 we render the main operational blocks of gCube workflow composition and execution mechanism, which us being s in the following paragraphs.

3.1 From query to workflow

gCube search engine exposes its capabilities via a functional, relational-algebra-like, well-formed query language. By the term "functional" we mean that every query operation acts as a function that applies to a given set of input arguments and produces a certain output. In this way, query operations can be put together, with one consuming the output of the other, forming an operation tree. Schematically, it resembles lisp function calls, in a pipelined manner. All traditional relational operations are supported (sort, project, join, etc.) along with several others stemming from the IR domain, such as full-text , geospatial, and content based search. The functional nature of the query language, offers true extensibility, allowing developers to add their own custom-made operations, while conditional execution allows true/false branching.

Behind this language, the framework follows a three step procedure for carrying out its operation, roughly assimilating the operation of modern RDBMSs, with several modifications in order to fit in the distributed nature of wide area networks and more specifically the Grid:

Query Parsing. User-defined queries are fed to the query parser, which processes the requests, validates the queries and forms the corresponding internal query representations as strongly-typed graphs. The validation process includes checks against data source definition, argument incompatibilities and obviously the necessary syntactic conformance.

Query Planning. The query planner is responsible for producing an execution plan that computes the original query expression. This plan is actually a web service workflow. Its nodes represent invocations of service instantiations and its edges communication channels between pairs of instantiations, in other words, producer-consumer relations. The input data stream of the producer service instance are being transported to the consumer instance for further processing via a transportation leveraging mechanism called gRS (gCube Result Set) [3].

The planner exploits registered service semantic descriptions [1], in order to decide which service instances can compute given query operations. Through the same mechanism the instance invocation parameters are generated, in accordance with the user query expression. Finally, data incompatibilities are resolved based on (data) source descriptions registered in the infrastructure.

As a result the planner creates an initial (non-optimal) execution plan. Although optimisation and service scheduling are left for subsequent stages, pre-

[1]metadata descriptions of service capabilities and instantiation procedures, expressed in XML Schema format [XSD]

liminary query optimisation is also applied here, based on heuristics and predefined cost estimations that take advantage of IR domain-specific knowledge.

The execution plan, which is still in an internal representation form, goes through the BPELBuilder component which produces the BPEL process, which along with some supplementary information, specific to each BPEL implementation, can be redirected to any BPEL engine. Due to the Workflow creation cost a caching mechanism is employed which, in the case of identical queries, requires the minor cost of validating against potential stale instances.

3.2 Process Optimisation

The gCube Process Optimisation Services (POS) implement core functionality in the form of libraries and web services for Process scheduling and execution planning. POS is comprised by a core optimisation library (POSLib) and two Web Services (RewriterService and PlannerService) that expose part of the library's functionality. POSLib implements three core components of process optimisation. POS is an integral part of query execution in gCube, since it is responsible for the optimised scheduling of workflows produced by the query planner and consequently is a key player in alleviating the grid overhead in query execution

Rewriter Provides structural optimisation of a process. It receives as input a BPEL process, analyses the structure, identifies independent invocations and formulates them in parallel constructs (BPEL *flow* elements) in order to accelerate the overall process execution. It is the first step of optimisation that takes place before the process arrives at the execution engine.

Planner Performs the pre-planning of the process execution. Receives an abstract BPEL process and generates various scheduling plans for execution. The generation of an executable plan implies that all references to abstract services are replaced by invocations to concrete, instantiated services in a gCube infrastructure. The Planner uses information provided by the gIS which keeps up-to-date metrics for resources employed in the grid (physical machines, services, etc). This information is input to various cost functions (applied by the paired Cost Estimator) that calculate the individual execution cost of a candidate plan.

The selection of best plans is performed by a custom implementation of the Simulated Annealing algorithm. The outcome of the planning is a set of executable BPEL processes that are passed to the workflow execution engine. Cost calculation can be guided by various weighted optimisation policies passed by the author (human or application) of the BPEL process inside the BPEL description.

ActivePlanner Provides run-time optimised scheduling of a gCube process. It is invoked during the process execution before any invocation activity to ensure that the plan generated by the Planner (during pre-planning) is still valid (e.g. the selected service end-point is still reachable) and optimal (according to the user-defined optimisation policies). If any of the former criteria has been violated the ActivePlanner re-evaluates a optimal service instance for the current process invocation. It can also work without pre-planning being available.

3.2.1 Optimisation Policies. The Planner and ActivePlanner components perform optimised scheduling of abstract BPEL processes based on user defined policies. Optimisation policies are declared within the BPEL document and can apply to individual *partnerLinkTypes* or to the whole process.

The selection of a specific Web Service instance to be used in a particular process invocation is driven by the optimisation policy applied either on the process level or on a partnerLink level. Currently POS supports six different optimisation policies:

Host load: Hosts with the lowest system load take precedence.

Fastest CPU: Hosts are ranked based on their CPU capabilities and the best is selected.

Memory Utilisation: Hosts are ranked according to the percentage of available memory as reported by the Java VM. The one with the highest percentage is selected.

Storage Utilisation: Hosts are ranked according to their total available disk space. The one with the larger available space is preferred.

Reliability: Hosts are ranked based on their total uptime. Precedence is given to hosts which have been running without interruption for longer time.

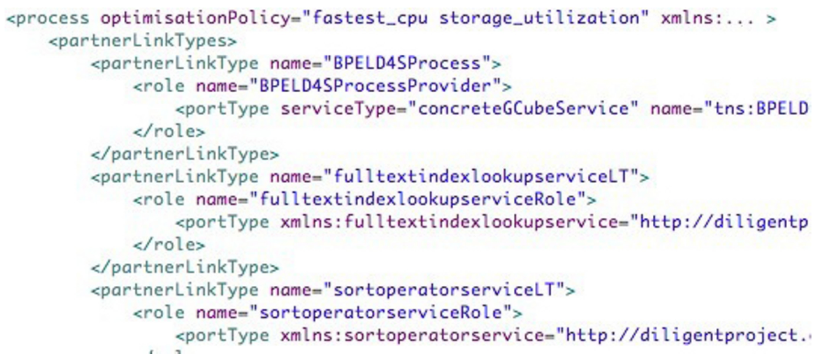
Network Utilisation: When the Planner evaluates multiple possible scheduling plans it will show preference to those plans where the web services are located close to each other (based on the reported host locality information). The Planner will avoid co-scheduling invocations to the same host in order not to overload it.

3.2.2 BPEL Optimisation Extensions. gCube POS functionality heavily depends on the BPEL standard (notation based on BPEL4WS v1.1). BPEL XML schema has been extended to include optimisation information such as process policy information per partnerLinks, the definition of abstract or concrete services, allocation relationship between invocations etc.

To define process wide optimisation policies we introduce the *optimisationPolicy* attribute at the BPEL process element. For example the process defined by the BPEL excerpt in Figure 2 will be scheduled according to the *fastest_cpu* policy (with higher weight) and the *storage_utilization* policy (lower weight).

If no policy is defined the default used is the *host_load* policy.

The policies defined on process-wide level pertain the planning of all partnerLinks included in the process unless a specific policy is defined on the part-



```
<process optimisationPolicy="fastest_cpu storage_utilization" xmlns:... >
    <partnerLinkTypes>
        <partnerLinkType name="BPELD4SProcess">
            <role name="BPELD4SProcessProvider">
                <portType serviceType="concreteGCubeService" name="tns:BPELD
            </role>
        </partnerLinkType>
        <partnerLinkType name="fulltextindexlookupserviceLT">
            <role name="fulltextindexlookupserviceRole">
                <portType xmlns:fulltextindexlookupservice="http://diligentp
            </role>
        </partnerLinkType>
        <partnerLinkType name="sortoperatorserviceLT">
            <role name="sortoperatorserviceRole">
                <portType xmlns:sortoperatorservice="http://diligentproject.
```

Figure 2. Process wide optimisation policy definition in BPEL

nerLink element. To define such policy we use the *partnerLinkPolicyType* attribute of the BPEL partnerLink element. This attribute is used similarly to the above example, except that the network_utilization policy, if defined, is ignored, since this policy makes sense only for the whole process and not for a particular web service.

3.3 Execution Stage

Apart from the execution engine, which does not fall within the scope of this paper and can be mostly outsourced, two important aspects of the Execution Stage are the services themselves and the data transport mechanism. Although potentially any Web Service can be employed in such a workflow as long as it gets sufficiently described for the framework, gCube comes with a rich set of components that implement core logic of structured and semi-structured data processing and information retrieval. Furthermore, as already mentioned, gRS is the special mechanism employed for data transports, that overcomes conceptual limitations and performance issues of Web Services and actually is the means via which data are streamed back to their requester.

4. Evaluation - Future Work

One inherent problem of Web Services based interactions is that they are not designed for low-latency, high-speed data transfers, while the SOAP processing stack, in practice proves to be quite a bottleneck for High Performance Computing.

The gCube framework has been exposing its facilities to selected user communities, through a Web Based user interface, which has given valuable feedback to the implementation team. Although the performance for interactive use cannot yet compete with the well known search facilities, the results are quite encouraging. Several optimisations, at various levels, allow for an acceptable response of the system to the interactive user even in non-optimally allocated resource schemes. The benefits of the system are materialised when the infrastructure is automatically reorganised and when data / computing intensive operations take place, such as the on-the-fly production of thumbnails of 100MB-sized images, or the extraction of the features of 1000s of images, without the employment of a pre-allocated infrastructure.

Practical issues rise with caching mechanisms, due to the size of the information they target and partially the insufficient support by underlying systems, which upon departs and arrivals of resources might become stale for short periods.

Today, gCube framework has reached a mature stage and is currently under way to production environments [19]. In this operational context, beyond the primary objective of maximum robustness, the aspects of optimisation and openness will be further elaborated. Optimisation techniques currently under development will exploit intermediate size estimations methods (via statistics, curve fitting etc). Steps considered for the future include rate-based optimisation methods for streamed flows and ontological matching of services. A low-level step towards performance will be the ability to dynamically combine executables (jars) under the encapsulation of a hosting web service.

Finally, driven from user requirements and system evolution, the Query Language will be revised in order to allow seamless integration of fundamentally different data sources.

Acknowledgments

This work is currently partially funded by the European Commission in the context of the D4Science project [19], under the 1st call of FP7 IST priority. Original work was partially funded in the context of DILIGENT project [20].

References

[1] gCube system, *http://www.gcube-system.org/*

[2] L. Candela, D. Castelli, P. Pagano, *gCube: A Service-Oriented Application Framework on the Grid*, ERCIM News, 48-49, Jan 2008

[3] Simeoni, Candela, Kakaletris, Sibeko, Pagano, Papanikos, Polydoras, Ioannidis, Aarvaag, Crestani, et al. *A Grid-based Infrastructure for Distributed Retrieval*, ECDL 2007, Proc. 11th European Conference on Research and Advanced Technology for Digital Libraries, Sept. 2007.

[4] Ian Foster, Carl Kesselman, Steven Tuecke. *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*, Lecture Notes in Computer Science (v2150), 2001

[5] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, *Modern information retrieval*, ACM Press, ISBN 0-201-39829-X, 1999

[6] E. Newcomer, G. Lomow, *Understanding SOA with Web Services*, Addison Wesley, ISBN 0-321-18086-0, 2005

[7] Foster et Al, *Modeling Stateful Resources with Web Services* whitepaper, IBM, 2004

[8] World Wide Web Consortium (W3C), *http://www.w3.org/*

[9] Organization for the Advancement of Structured Information Standards (OASIS), *http://www.oasis-open.org/*

[10] I. Foster, C. Kesselman, J. Nick, S. Tuecke, *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*, Globus Project, 2002

[11] Globus Toolkit 4, *http://www.globus.org/toolkit/*

[12] *Business Process Execution Language for Web Services version 1.1*, IBM, BEA Systems, Microsoft, SAP AG, Siebel Systems, 2002-2007

[13] Condor Project HomePage, *http://www.cs.wisc.edu/condor/*

[14] I. J. Taylor, E. Deelman, D. Gannon and M. S. Shields (eds.), *Workflows for e-Science: Scientific Workflows for Grids*, ISBN 978-1-84628-519-6, Springer London, 2007

[15] KWf Grid Project Web Site, *http://www.kwfgrid.eu/*

[16] OGSA Data Access and Integration (OGSA-DAI), *http://www.ogsadai.org.uk/*

[17] The EGEE Project, *http://www.eu-egee.org/*.

[18] gLite, *http://glite.web.cern.ch/*.

[19] The D4Science Project, *http://d4science.research-infrastructures.eu/*.

[20] The DILIGENT Project, *http://www.diligent-project.org/*.