In: Proceedings of the 8^{th} DELOS Workshop on Future Digital Library Management Systems: System Architecture & Information Access (FDLMS 2005), Schloss Dagstuhl, Germany, March/April 2005.

Dynamically Making Use of Distributed Data Sources in a Grid Environment (Ext. Abstract)

Manfred Wurz and Heiko Schuldt

University for Health Sciences, Medical Informatics and Technology
Eduard-Wallnöfer–Zentrum 1 A–6060 Hall in Tyrol, Austria
[manfred.wurz|heiko.schuldt@umit.at]

Abstract. To avoid the cost of multiple and costly examinations, health care institutions are in need to share information about scientific insights and patient data more intensively and transparently. The need for seamless but still robust and secure collaboration is rising. Based on that scenario, this paper proposes an architecture for dynamically parallelizing service requests in a grid environment without the need to change existing and conscientiously tested functionality. The task of preparing software for parallel execution is split into an application-specific part of partitioning requests and re-integrating results and a generic component responsible for the actual parallel calls, state management, failure handling, and robustness.

Introduction

One major goal of grid computing is to establish highly flexible and robust environments to utilize distributed resources in an efficient and transparent way. Due to the highly dynamic nature of such environments where computational nodes may leave or join in, it is essential to bind service invocations to concrete service instances at run-time. This allows to flexibly react to changes in the environment. In a service-oriented world, application logic is encapsulated by means of services. Standards like SOAP over HTTP can be used for the invocation of (Web) services, and WSDL for accessing information on the capabilities of services. When several instances of the same service exist in a grid environment, then it should be possible to dynamically make use of as many service instances as possible by parallelizing a (Web) service call and by submitting requests in parallel to them.

The goal of this parallelization is twofold and depends on the characteristics of the services which are subject to parallelization. First, we aim to make use of as many services as possible (and therefore of the data accessible by those), to increase the quality of the result. This is particularly true for the access to data sources, encapsulated by dedicated services. Second, having multiple service instances accessible opens the possibility to speed up the processing of computationally intensive tasks. Whereas examples for the latter have been presented in detail in [6, 5] and large scale experiments with more than 2500 worker nodes have been demonstrated in [1] using MW class library [3], this work focuses on

the goal of enhancing the quality of the request when accessing data sources by means of services.

The contribution of this work is to introduce an architecture of a service seeming to be an ordinary, callable service to the outside world, which is able to adopt its behavior controllable by optional quality of service criteria, and the resources available on a grid. In short, such *dynamic* services use meta information on the currently available service providers and their capabilities and partition the original request into a set of simpler requests of the same service types. These (sub-)requests are then submitted in parallel to as many service providers as reasonable, and their responses are finally integrated and returned as the result of the original request. In particular, this parallelization shall be carried out without the need to modify existing functionality or interfaces, and transparently to the user and developer. Due to its master/slave nature, it is especially suitable for grid environments [2].

To better illustrate the benefits of such an approach, the following scenario describes how these dynamic services can be used in healthcare applications.

A Sample Scenario: Genotype/Phenotype Correlation. *Donald, a patient, consults his family doctor telling him about pain in his chest, shortage of breath, and drowsiness. Besides these symptoms being an indicator for angina pectoris, Donald describes that starting a few hours ago, he additionally suffers from pain in his abdomen and changing paralysis of his right and left leg. Based on this description, the physician wants to ask for a second opinion and admits the patient to a hospital for further examination. The specialist who examines Donald in the hospital recognizes some very specific clinical artifacts which rise his interest (gigantism, a funnel chest, scoliosis, and acromacria). Due to these symptoms, Donald has to undergo clinical as well as molecular genetic examinations, since this clinical fingerprint might be caused through a fibrillin-1 (FBN1) gene mutation. To find out more about Donald's potential genetic mutation, his clinical fingerprint is used as an input for a genotype/phenotype correlation. The quality and statement of such a correlation is highly influenced by the amount of data that can be processed. Collaboration among various healthcare institutions, where data is made available by genotype/phenotype correlation services, is therefore of great importance. Luckily, the clinical fingerprint of Donald was indeed listed in the data set of a partnering hospital, connected to the grid and dynamically included in the search, and correlates with the Marfan–Syndrome. An expensive molecular biological examination in which this genetic defect is confirmed is conducted. Therefore, in addition to the acute vascular operation he has to undergo because of his clinical symptoms, his family can be invited for genetic screening to avoid similar costly, high risk operations in the future.*

Using the approach described in this work, the correlation service does not only consider the in-house service of a single hospital. Rather, it is a self-adaptive, *virtual* service that dynamically and in parallel calls all available genotype/phenotype correlation services in the system, thereby jointly accessing the data sources of several healthcare institutions. As a benefit of the architecture proposed, the client application as well as the genotype/phenotype correlation

functionality does not have to be updated or rewritten to participate in or benefit from such a self adaptive environment.

Although the above example is taken from eHealth digital libraries, the proposed architecture can of course be applied to other domains as well. Whenever it is appropriate to dynamically replace a single invocation of a service by multiple invocations, dynamic adaptation and parallelization can be highly beneficial.

The remainder of this extended abstract is organized as follows. We briefly describe the overall architecture and show the benefits of dynamic parallelization for the application scenario introduced above. Finally, a conclusion and an outlook on future work is given.

Overall Architecture

The goal of the architecture we propose is to improve the usability of a single (Web) service, as well as to facilitate faster and less error prone development for grid environments. This approach is based on the observation that, following the current proliferation of service-oriented architectures, the number of services and service providers in a grid will significantly increase. Especially services which are provider independent and are not bound to special resources can be distributed fast and widely in a grid environment or be deployed numerously on demand. Although the availability of many congruent services as well as the computational resources thereby offered seem to be within reach, adapting functionality for parallel execution is still necessary and tedious.

The task of partitioning request parameters and re-integrating results afterwards is highly application-specific and, from our perspective, cannot be solved in a generic way. Although we see the potential to identify classes of applications according to the mechanism they partition and re-integrate requests which allows to have pre-built splitter and merger services, an expert in the problem domain will be necessary to tailor them for the specific need or perform some additional, application domain specific work. We name a service, enriched with the capability to partition incoming requests and reintegrate partial results, a *dynamic* service.

Apart from that we introduce a *Dynamizer* component, which can be built generically and which is responsible for state management, failure handling, and service discovery. While the latter characteristic is independent concerning the twofold approach described in the previous section, failure handling and state management differs. In case of a dynamic service aiming to leverage the result quality, the unexpected absence of a 'worker' service already in charge of a partial task does not hinder in producing a result. If partitioning the request of a computationally intensive task to gain better performance, the overall results rely on each sub task and the *Dynamizer* is responsible for compensating any failing 'worker' service.

As shown in Figure 1, the following logical units can be identified for dynamic services. The box in the center of the left side, labeled 'Payload Service', represents the actual service. It is responsible for providing application seman-

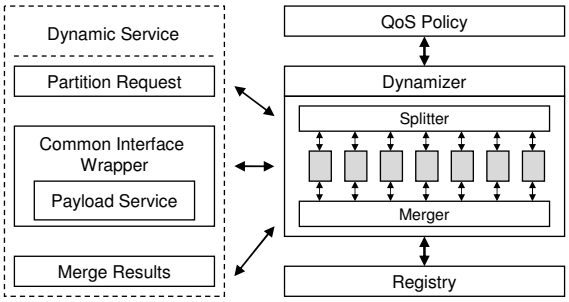

Fig. 1. Overall Architecture

tics, e.g., a genotype/phenotype correlation on a local database. This is usually a piece of business logic that has existed beforehand and is now supposed to be opened to the grid and enabled for parallel execution. To achieve this goal, it is surrounded by another box, labeled 'Common Interface Wrapper', which encapsulates the 'Payload Service' and enhances it with a common interface.

On top, 'Partition Request' encapsulates knowledge on how incoming parameters for the 'Payload Service' have to be partitioned, so that the original request can be decomposed into numerous new sub-requests. Each of these sub-requests can then be distributed on the grid and be processed by other instances of the originally targeted service. The box at the bottom ('Merge Results') integrates (partial) results returned from the grid to provide the original service requester with a consolidated result. It can therefore be seen as the counter operation to the 'Partition Request' service. The combination of these elements is referred to as 'Dynamic Service'.

To find instances of the originally targeted service (e.g., services where the description equals the one of the 'Payload Service'), a registry is used (depicted in the lower right corner of Figure 1) . This registry provides information on which services are available, how they can be accessed, and what their properties are (e.g., CPU load, connection bandwidth, access restrictions, etc).

The 'Dynamizer', depicted on the right hand side, makes use of the services mentioned above. It glues together the previously described services by making the parallel calls and coordinating incoming results. The 'Dynamizer' can interact with all services that adhere to a common interface, as ensured by the 'Common Interface Wrapper'. It can be integrated in environments able to call and orchestrate services, or it can be packaged and deployed together with specific services.

To make the best possible use of the 'Dynamizer', the user can send an optional description of the desired service quality along with the mandatory parameters needed to process the request. In this Quality of Service (QoS) policy, the user can, for example, describe whether the request should be optimized in terms of speed (select high performance nodes, and partition the input parameters accordingly), in terms of bandwidth (try to keep network usage low) or if it should aim for best accuracy (important for iterative approaches or database queries,

where there is an option to use different data sources). Since these specifications can be contradictory, adding preferences to rank the user's requirements is of importance. To better illustrate the mechanisms within the 'Dynamizer' regarding the user specified QoS policy, we consider the following example: the specialist from the previously introduced healthcare scenario specifies that he wants to use as many genotype/phenotype correlation information as possible and as affordable within a 300 Euro budget. The 'Dynamizer' finds 7 services with a total of 2 gigabytes of searchable data, each charging 60 Euros per query. Alternatively, there are 40 services available provided by smaller institutions, having just searchable amounts of data starting from 4 megabytes up to 30 megabytes and charging .50 per query. The algorithms on how to reconcile the user specifications, the details of the QoS description language and how to integrate this best with our existing implementation is currently investigated.

Conclusion and Outlook

In this paper, we have stated the importance and the usefulness of an easy to use, straightforward to develop and robust architecture to dynamically parallelize (Web) service calls without the need to change existing functionality. In future work, we plan to implement the assignment of QoS policies to service requests as well as adding the ability to use semantically equivalent instead of congruent services. First experimental results will be refined and further empirical studies will be conducted to verify the validity of the approach described. When dealing with semantically equivalent services, (partial) results are likely to be heterogeneous, and mechanisms for integrating them have to be developed. This, additionally to defining appropriate metrics for semantic equivalence in the context described, is currently under investigation. Along with these changes, the fault tolerance, robustness, and scalability of the introduced 'Dynamizer' component is improved by integrating it with OSIRIS [4], a distributed workflow environment.

References

1. Kurt Anstreicher, Nathan Brixius, Jean-Pierre Goux, and Jeff Linderoth. Solving Large Quadratic Assignment Problems on Computational Grids. In *Mathematical Programming 91(3)*, pages 563–588, 2002.
2. Ian Foster and Carl Kesselman, editors. *The Grid 2, Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, 2004.
3. Jean-Pierre Goux, Sanjeev Kulkarni, Jeff Linderoth, and Michael Yoder. An Enabling Framework for Master-Worker Applications on the Computational Grid. In *9th IEEE Int'l Symp. on High Performance Dist. Comp.*, pages 43–50, Los Alamitos, CA, 2000. IEE Computer Society Press.
4. C. Schuler, R. Weber, H. Schuldt, and H.-J. Schek. Scalable Peer-to-Peer Process Management - The OSIRIS Approach. In *Proceedings of the 2nd International Conference on Web Services (ICWS'2004)*, pages 26–34, San Diego, CA, USA, July 2004. IEEE Computer Society.

5. M. Wurz, G. Brettlecker, and H. Schuldt. Data Stream Management and Digital Library Processes on Top of a Hyperdatabase and Grid Infrastructure. In *Pre-Proceedings of the 6^{th} Thematic Workshop of the EU Network of Excellence DELOS: Digital Library Architectures - Peer-to-Peer, Grid, and Service-Orientation (DLA 2004)*, pages 37–48, Cagliari, Italy, June 2004. Edizioni Progetto Padova.
6. M. Wurz and H. Schuldt. Dynamic parallelization of grid–enabled web services. In *Proceedings of European Grid Conference 2005 (EGC 2005) (to appear)*, 2005.