Proceeding of the Third IEEE International Workshop on Scientific Workflows - SWF2009
Los Angeles, US, 2009/07

Towards Quality of Service in Scientific Workflows by using Advance Resource Reservations*

Christoph Langguth, Paola Ranaldi and Heiko Schuldt
Department of Computer Science, Database and Information Systems Group
University of Basel, Switzerland
firstname.lastname@unibas.ch

Abstract

The increasing interest in web services has led to a recent proliferation of service-oriented platforms for supporting Scientific Workflows in a variety of e-Science domains. These platforms mainly address the application-specific composition and enactment of workflows. However, since scientific workflows are characterized by complex computations and large volumes of data, non-functional criteria like Quality of Service (QoS) guarantees can significantly raise the predictability of the computations, and thus constitute an important benefit to users. This paper introduces the DWARFS vision of a distributed execution engine for Scientific Workflows which features support for QoS guarantees by making use of Advance Resource Reservations. In particular, we present the DWARFS methodology to enact workflows under QoS constraints and report on the ongoing implementation activities.

1. Introduction

Scientific workflows are rapidly emerging as a paradigm for representing and managing data intensive and complex computation, helping scientists to conceptualize and manage data analysis, visualization and data integration. The groundworks for enabling such workflows are laid by the adoption of Service Oriented Architectures, in which functions are separated into distinct units (services), and standardized interfaces and protocols such as WSDL and SOAP provide the basis for loosely coupled, distributed systems. These services may then be accessed by any standards-compliant client, and can be combined and reused as building blocks for more complex workflows. Consider a workflow from [3], of which Figure 1 presents a simplified version. This process is used to acquire and analyze atmospheric data, and results in a weather forecast. Activities in the workflow encompass data acquisition and preprocessing, data transformation and data analysis. This weather forecast workflow is a typical example of a process involving large amounts of data and complex computations, while at the same time it is a representative case of a computation whose results are needed in a timely manner. The processing is hence bound to time constraints.

Currently, most services are provided in a best-effort manner, where all clients are effectively competing for resources, and service providers aim to serve all clients in a "fair" way. Thus, in a possible overload situation, all clients experience equally bad performance. Since a workflow is essentially an aggregation of multiple service calls, the performance of a workflow directly depends on the performance of (all of) its contained activities. If more predictable processing is required, as is the case with time-constrained workflows like the one above, a common approach is to "set aside" dedicated machines for particular computations or clients. This effectively constitutes an exclusive reservation of those resources for a particular client; while it is the easiest solution to giving some kind of guarantees for those

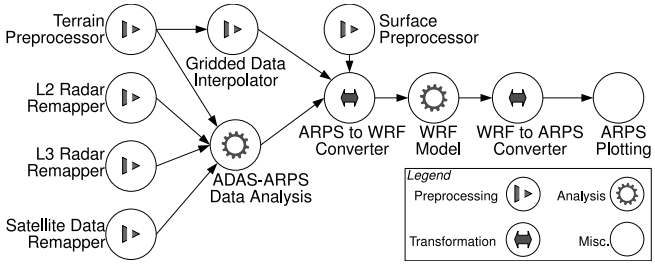

Figure 1. Weather Forecast Workflow

*This work has been partly supported by the Hasler Foundation in the context of the COSA project (COmpiling Service-oriented Architectures).

clients, it may also result in a waste of resources when these machines are idle. As one of the main opportunities of adopting SOAs lies exactly in the ability to provide services to any compatible client, such a strategy also seems like a step backwards: other clients are completely prevented from using services not because of interoperability issues, but merely because of their (potential) performance impact.

A more flexible approach to tackle the problem involves Service Level Agreements (SLAs), in combination with Advance Resource Reservation (AR): clients and service providers establish an agreement by which a client reserves certain resources of the provider in advance; at runtime, providers make sure that they are obeying the agreements they committed to, by providing clients with the respective resources. However, they are also free to serve other clients – within the bounds of the remaining resource capacities, of course. SLAs and ARs thus constitute an appropriate basis for enabling Quality of Service guarantees.

In this paper, we present our vision of a workflow execution system that provides Quality of Service guarantees at the level of entire workflows: if individual service providers (the activities of the workflow) can provide QoS guarantees, a workflow encompassing these activities should also be able to provide such guarantees. As the system specifically targets Scientific workflows, it also includes specialized support for the management of large data volumes. Finally, the system supports distribution, in the sense of multiple distributed engines cooperating to orchestrate workflow execution, for improved scalability and optimal performance.

Some of the fundamental concepts required for the functioning of the envisaged system have been implemented, yet many of the topics we present are subject to ongoing research. The purpose of the paper is thus to give an overview on the general concepts that are required to build such a workflow engine, and to present our approach and first results of its realization.

The rest of this paper is structured as follows: Section 2 gives an overview of the main features of the system and fundamental prerequisites of the environment. Section 3 describes the steps needed for enacting a QoS-aware workflow and presents underlying concepts. Section 4 presents related work, and section 5 concludes.

2. The DWARFS System

The main goal of the DWARFS system (Distributed Workflow engine with Advance Reservation Functionality Support) is to provide a workflow execution engine, with additional features that make it particularly suitable for scientific workflows.

2.1. Overview

Using one of the current commonly used workflow engines, the enactment of a process would involve the following steps: i) the user deploys a process definition on an engine (possibly along with required auxiliary data); ii) on user request, the engine instantiates and executes the process.

There are several observations to be made with regard to the execution. Firstly, web services are currently executed in a "best-effort" manner: it is reasonable to assume that service providers will do their best to answer client requests, however usually no guarantees are given. Simultaneous invocations by multiple clients may result in these operations competing for resources (such as CPU, storage, or even specialized hardware equipment), especially in the case of scientific data processing, where computationally expensive, long-running operations are to be performed. In the best case this will slow down the delivery of results for all parties, while in the worst case it will result in erroneous termination because of resource shortage. Since individual activities of the process are provided in such a best-effort manner, the entire process itself is as well. A possible way to alleviate this problem is to reserve resources (or shares thereof) for individual clients, thus being able to provide better guarantees about the Quality of Service – in the broad sense of the word – to that client. Such reservations have to be negotiated before needing them, and be established using SLAs with ARs.

A second observation is that a scientific workflow context may involve massive amounts of data, which have to be shipped from one target service to another. Because the engine is the central orchestration instance for the workflow, all of that data needs to be piped through that engine, thus being shipped "back and forth" with each service call. This data transfer can usually not be avoided; however, by placing the execution engine close (in terms of network distance, or speed) to the target services, the negative impact on speed and possibly cost can be reduced. If the target services are physically too far apart to find an execution engine which is close to both, one may want to fragment the process into multiple interlinked parts, distributing the orchestration over multiple engines and pre-shipping data between engines as required.

This short discussion motivates the main features that our envisaged solution will leverage, which are summarized hereafter.

Decentralized Orchestration. A process to be executed within DWARFS is split into multiple fragments, each executed by one of several cooperating execution engines. To name just a few advantages, this reduces the probability of introducing a single execution engine as a hot spot or single point of failure. Note that this fragmentation approach subsumes the complete spectrum of possibilities: using a single fragment for the entire process corresponds to a traditional centralized execution, while the other extreme – one fragment for each invocation, where engines are colocated with the target services – corresponds to a fully distributed system.

Quality of Service, Service Level Agreements and Advance Reservations. A fundamental aspect that we target in DWARFS is the notion of QoS. From an end-user perspective, this means that the execution of the entire process is subject to some QoS guarantees. One of the most prominent use cases would be the possibility to give guarantees about the execution time of the process: before a workflow is started, users should be able to specify their needs and expectations towards the execution, e.g., "I need this process to finish as fast as possible, no matter how much it will cost", or "I want to execute it at the lowest possible cost, even if this takes a longer time", or even "I want to execute it as energy-efficient as possible". Note that these are just examples – however, they show a crucial aspect of the usefulness of our approach: it raises the *predictability* of computations far beyond the current state.

Technically, the QoS aspects of an execution are formalized using SLAs. Each process instance is subject to execution in the context of an instance-specific SLA, which must be negotiated between the client and DWARFS before the execution can start. However, since processes at their core involve the invocation of other ("target") Web Services, any negotiation of such SLAs for the process internally depends on the same kind of negotiations with the target services. When a user wants to submit the execution of a process within a particular SLA to the DWARFS system, this has to locate the resources required for that process and reserve these resources for the time and under the requirements needed for satisfying that SLA. As processes imply a partial order of the target service invocations, the reservations with the target service providers will refer to time intervals in the future. Thus, ARs are a first-class citizen in the DWARFS system: any service which is to be called within the context of a process must support SLA negotiation and AR; likewise, any process execution in DWARFS itself is subject to the same kind of Advance Reservations, where clients negotiate with the DWARFS system for future execution of processes. The objectives of the process-level SLA between the client and DWARFS system can be determined in a bottom-up manner based on the agreements with the target services.

Efficient handling of large data volumes. One of the crucial differences between scientific and business workflows is the amount of data that are computed and that need to be shipped between the activities constituting the workflow. Fragmenting the workflow and distributing the resulting processes over different engines within the system leads to the new challenge of handling the data shipping among the engines – which must be done in a way that conforms to the QoS constraints imposed by the workflow client. The engines must be able to efficiently store the output data of workflow activities, as well as to efficiently retrieve the input data in order to send it to subsequent activities. Once a fragmentation for the process is identified, if data shipping among engines is still needed, a storage management system must be able either to locate a storage service "close" to both the engines or to locate different storage services, each one close to a workflow engine, where data must be replicated.

2.2. Basic Assumptions

There are some basic assumptions about the environment which are needed for the DWARFS system to provide its advanced features.

A first assumption is that the data flow within the process is actually taking place within the messages transmitted during operation invocation. In other words, the engine actually gets to "see" the data. Unfortunately, the SOAP protocol was initially not designed, nor working well, for large data transfers. In fact many real-life implementations have resorted to what could be called workarounds, such as using intermediate storage (GridFTP etc.), and only shipping references or "pointers" to the data [2, 9]. Whereas this approach works, it does not "scale" in the sense of interoperability, because it requires all involved parties to support the same notion of storage references, and to be able to handle the additional protocols. With the adoption of SOAP attachments and similar specifications such as MTOM, one can return to the more natural approach of sending the actual data inside the messages – and let the system optimize the data transfer strategy.

We furthermore assume that workflows are entirely automated, i.e., not involving any human interaction or feedback. The reason for this is the automation of the negotiation, and enforcement, of the SLAs.

Because the goal of DWARFS is to determine and

enact an optimal execution strategy in a dynamic environment according to instance-specific constraints, the workflow specification itself must allow for this flexibility: operation invocations therein should be specified at the porttype level of abstraction (instead of concrete bindings), so that DWARFS can make use of appropriate mechanisms such as UDDI registries to determine the concrete instances to use for execution.

We assume that all providers of target services are able to support the negotiation and enforcement of SLAs. This is a task that is rather independent of the specifics of individual services, and can be addressed by the container. One result of our work will be a modified WS container with this kind of support built-in.

For planning the execution, the DWARFS system requires providers to be able to supply additional metadata, concerning predictions about i) the resources an operation will require, ii) the timing of the operation, and iii) data characteristics (e.g., "given input data of some class X and of a certain size, the output data is expected to be of this and that size and class Y"). These predictions are vital to the correct functioning (and usefulness) of our entire approach, but there is no magic one-size-fits-all prediction service that accomodates every possible use case. In fact, the actual contents of the metadata is meant to be almost entirely opaque to the execution engine – which means that it can (and usually needs to) be tailored to the specific application domain. Thus, the metadata should contain the pieces of domain-specific information that allow to derive the best possible predictions both for data and timing. Whereas the exact approach for determining those predictions remains an open issue, section 3.2.1 presents first results on the aspect of (simple) runtime prediction; concerning the data aspect, ideas include the possibility of the service designer expressing various rules for determining the information, having the service container itself determine them by machine learning mechanisms and heuristics, or a combination of these.

Despite these requirements, we are nevertheless trying to limit the adaptations needed – in particular, we strive to keep the impact on the implementation of the target services themselves as low as possible, avoiding the need for changing and recompiling the scientific services which should be used within DWARFS.

2.3. Model and architecture

The following gives a brief overview of the most important aspects of a model which will be used to formally represent the relevant parts of the system.

Services consist of *operations*, and are deployed on different *hosts* (*service instances*). Each host has a set of *resources* of different *resource types* (e.g. CPU, storage, etc.). In other words, each host has an individual capacity of a resource type. Operations, in order to be executed, make use of some of the resources of the host, to a certain degree – which we call a *resource share*. Each operation must be invoked in the context of one or more *reservations* encompassing all of the required resources. Reservations have a lifetime represented by their start and end times. An intuitive representation of a sample reservation could be "30% of the processing power, and 50 GB of storage, from 8:00 to 8:30".

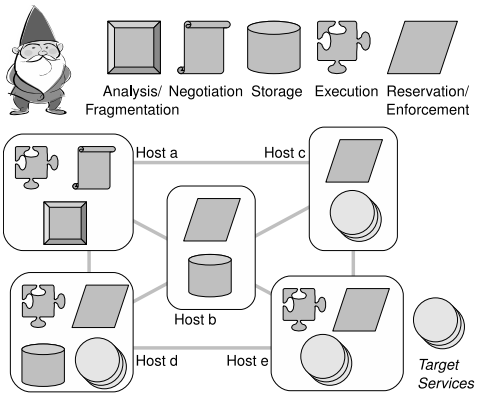

Figure 2. DWARFS modules and deployment

Figure 2 shows the different modules that DWARFS consists of, and a possible deployment scenario of these modules on different hosts.

3. QoS-aware Workflow Execution

Considering the goals of DWARFS and the resulting identified assumptions, the lifecycle of the execution of a process instance in DWARFS involves several steps, and looks as follows:

1. Process Definition: the user provides the process definition, along with the required metadata, to the DWARFS system – this starts the negotiation phase between the user and the system.

2. Analysis: the definition is analyzed to gather information about the data flow inside the workflow. At the same time, information about potential providers of the individual activities (operations) is collected.

3. Fragmentation: DWARFS finds a fragmentation strategy for distributing the workflow orchestration.

4. Negotiation: DWARFS negotiates with the providers, in order to come up with reservations for the remote operation invocations.

Note that this is an iterative strategy, i.e., the goal is to optimize both the fragmentation and the reserva-

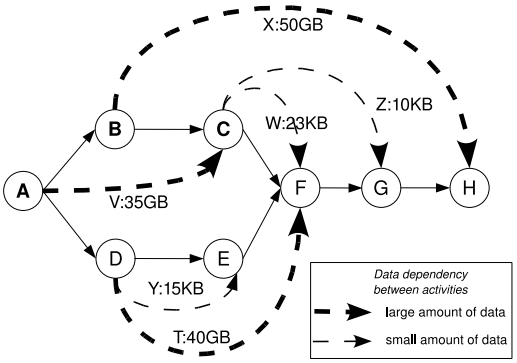

Figure 3. Sample Process: Data flow

tions. Once the fragmentation and SLAs have been set up, the system is ready to execute the process.

5. Execution and Enforcement: the process is executed under the terms of the SLA negotiated with the user. While the process is being executed, DWARFS – and the involved service providers – must take care to monitor and enforce the abidance to the SLAs.

6. Re-negotiation: in case of exceptional situations during the execution, re-negotiations with providers might be required.

7. Accounting & Billing: charge the resources actually exploited to the user. Although this is a necessary complement to ARs, it is not in the focus of our work.

In what follows, we address the most relevant of the above steps in more detail.

3.1. Analysis and Fragmentation

As a first step, the process to execute, and additional metadata as described later, is analyzed and preprocessed by the system. The goal of this step is to fragment the input process, in order to obtain several smaller processes to be executed on different execution engines within the system. This fragmentation is meant to optimize the execution of the process by minimizing the latency introduced by data transfers. To this end, additional information about the data flow inside the process is required. One possibility to get this information is to rely on it being provided as additional metadata to the process execution. As a second alternative, one can rely on the information being present in the process itself, by leveraging a process description language which makes data flow explicit, such as Scufl [10] or BPEL-D [15]. Finally, as shown in [6], it is also possible to derive the needed information with good accuracy from a "plain" BPEL process description.

In addition to the mere information about the data flow sources and sinks (i.e., "what goes where"), further information about the expected volumes of data is desirable. The rationale is that not all data inside a process instance can be assumed to be equal, in terms of their size and their purpose. While our assumption is that one property of a scientific workflow is that it deals with large amounts of data ("processing data"), there are other pieces of information ("auxiliary data", e.g., metadata about data formats, correlation and controlling information etc.) which are very likely to appear in the handled processes and which tend to be very small in size. Figure 3 shows the process definition, enriched with the data flow information.

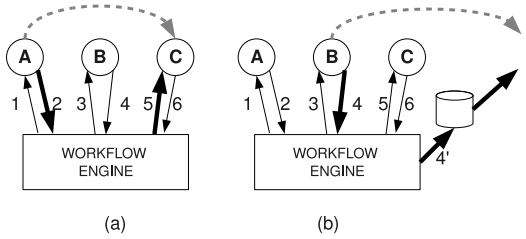

Figure 4. Low-level data flow example

While conceptually one can distinguish between data flow and control flow, technically (at the lowest level) both is done by passing messages between hosts. Figure 4 shows only a fragment of the process, where the actual message exchange between an orchestration engine and the invoked services is shown. Any output data of a service is transferred to the workflow engine. If it is required as input to another service, it will again be shipped there using a different message. Thus, the dataflow between operations A and C of our sample workflow would correspond to the messages 2 and 5, as depicted in Figure 4(a). Exploiting the proximity of service providers and execution engine (in terms of bandwidth) will therefore help to minimize latencies due to data transfer.

One of the goals of the preprocessing is to keep most large data transfers within the same fragment. Where those transfers must happen between fragments, we make use of the storage subsystem, as depicted in Figure 4(b). To speed up the data transfer between different workflow engines, the storage management system must be able to locate a storage provider that has high-bandwidth connections with these engines. The storage management system must provide either a single storage provider – where the data will be both stored by the "producing" engine and retrieved by the "consuming" engine –, or multiple storage providers, among which the data are shipped when needed, and with each storage provider having a high-bandwidth connection with an engine.

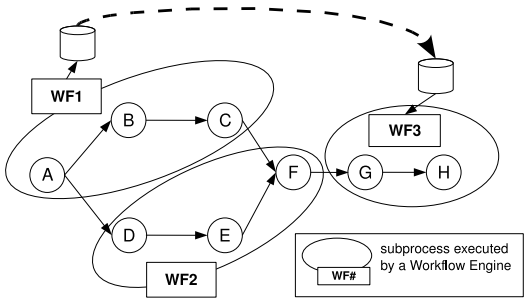

Figure 5. Sample Process: Fragmentation

The final fragmented process, rewritten to include explicit references to the storage providers, is depicted in Figure 5. It shows that now the process has been fragmented into three subprocesses, each handled by a different engine. The remaining large inter-fragment dataflows – in this case only the one between operations B and H – have been replaced by references to storage service instances.

Once a good possible fragmentation has been found, the next step is to determine a set of reservations (expressed as SLAs) that is "optimal". What optimal means in this sense is influenced mostly by constraints given by the user (e.g., "as fast as possible"), but also by system criteria. Examples for the latter are achieving a combination of reservations that does not waste a lot of time, or having reservations that last as short as possible, but still long enough to be reasonably sure the activities will finish within.

3.1.1 Determining additional Information about Data

We have previously referred to the volume of data, which is information that is taken into account for the analysis, in order to determine where and how the storage subsystem should be integrated into the process. The idea is the following: for each data item that serves as input or output, there is additional metadata available. This metadata is mostly opaque to the orchestration engine, i.e., it contains domain-specific information about the data. Initially, the engine only knows the metadata about the process inputs (which is supplied by the client wishing to enact the process). By following the dataflow edges, it can iteratively acquire the same kind of metadata about the outputs – which may again serve as inputs later – by querying the service provider. Note that while most of the metadata is opaque, in the sense that it does not need to be understood by the orchestration engine, it must at least be able to extract the information about the data volume from within that "black box". The exact representation of this information (i.e., whether to use some common vocabulary, ontologies etc.) is still subject to future research. We expect that all instances of a given service, when given the same metadata as input, return the same output information (regardless of the host they are deployed on).

3.1.2 Determining required Resources and Execution Times

As reservations are a first-class citizen in DWARFS, another fundamental information that is needed for the planning is the resources required for performing, and the estimated runtime of, an operation. This information as well can only be obtained from the service providers. In contrast to information about data, these estimations will differ between hosts, because they depend on the actual capacities of resources at the individual hosts. For the sake of completeness, it should be mentioned that the concept of resources is not limited to just CPU and storage, but can encompass many more aspects. For example, hosts can have physically attached hardware which they are able to interact with – like sensors, microscopes, telescopes etc. From the point of view of the model, it is possible for all these resources to be treated equally, even without knowing their purpose. Given that each resource on a host has a finite capacity, one can argue that any kind of resource may be used to a certain extent, between 0 (no usage) and 1 (usage of full capacity). For the sake of readability, we will mostly refer to the corresponding percentage values. While the applicability of the approach to CPU and storage should be clear (e.g., running a single thread on a dual-core CPU will utilize a maximum of 50% of it; 5 GB of a 50 GB harddisk corresponds to 10% of its capacity), this can even be applied to other devices: if there are two thermometers attached to a machine, one can use either 0%, 50%, or 100% of the resource. Whereas in this case other shares (e.g., 44%) do not make sense, the same notion of share can nevertheless be reused.

As an example of the above, consider the operation `A` of the workflow, of which the providing service is deployed on two hosts. These hosts differ in their hardware configuration – for instance, host 1 has a powerful dual-core processor, 250 GB of harddisk space, and 10 pieces of Data Acquisition Instruments (DAI) attached. Host 2 has an older single-core CPU, 1.5 TB of storage, and one DAI. After determining that these two hosts are candidates for the given operation, and knowing the metadata about the input, the hosts can be queried about their predictions for the

	Host 1	Host 2
INPUT		
operation	A	
data info	in1: size=50 G, class=X	
OUTPUT		
timing	2:30h	3:50h
resources	CPU: 50% max inv.prop.	CPU: 100% max inv.prop.
	stor.: 30% min —	stor.: 5% min —
	DAI: 10% exact —	DAI: 100% exact —
data info	out1: size=25 GB, class Y	

Table 1. Metadata example

operation execution. As shown in Table 1, both hosts predict different execution times, CPU shares, storage shares, and hardware shares, according to their locally available resources. However, there is additional information shown, namely the allowed limits of resource shares, and the relationship of resource shares with the execution time. For example, Host 1 states that this operation request should request a maximum of 50% of the CPU, and that execution time of the operation is inversely proportional to the requested share; the storage share to reserve has a lower bound of 30%, and no implication on the runtime; finally, the share of hardware is fixed and does not influence the timing.

3.1.3 Combination of Reservations as an Optimization Problem

In addition to the previously presented ones, there are two additional items of information which need to be available from the hosts: information about resource allocations and the corresponding time intervals (to determine when resources are available to which degree), and information about the cost (monetary or otherwise) of envisaged reservations. Note that we assume to be able to obtain full knowledge about these factors a priori to planning, to be able to do all planning locally without additional communication with the hosts. From an implementation point of view, this could mean that hosts provide their full cost function definition to the client, instead of "hiding" it and only providing clients with results. After all this information is finally available, one is able to formulate an optimization problem and explore the solution space. By making all variable parts of the system explicit, and also providing the rules by which changes to the variables affect the calculations, all calculations can indeed be performed locally without requiring further interactions.

3.2. Execution and Enforcement of SLAs

Advance Reservations and SLAs provide the foundation for a better predictability of the behaviour of all parties involved in the workflow execution. However, it is of utmost importance to have a system in which the predictions that lead to an agreement are actually dependable enough to be useful, and in which participants can enforce the guarantees that they give.

There are several kinds of commitments that the participants enter:

Reservations for "static" resources concern resources such as storage, and arguably other hardware. These kinds of reservations, and their enforcement by providers, are generally well understood [13]. However, the timing (from when to when should the reservation be made), and the exact amount (e.g., how much storage capacity should it encompass) are provided by the client and subject to the discussion below.

Reservations for "dynamic" resources, such as CPU shares. Dynamic in this context refers to the difficulty of constantly providing an exact amount of the resource.

Timing predictions. This is arguably one of the most crucial aspects, as they are used for determining the duration of reservations. Additionally, in a workflow, they also affect the start times of successive operations.

Data characteristics predictions. Similarly to timing predictions, these are used to determine essential information for the reservations.

3.2.1 Results

As to the timing predictions and reservations for CPU shares discussed above, our observation is that they are tightly related. To summarize our previous work [7], one can confine the usage of CPU to the requested level with relatively good accuracy (the average error is within 5% of the requested share). This, in turn, enables us to derive good predictions for future runtimes of the same operation, with an average variation of 2% to 5 %, and the quality of predictions increasing with the number of past runs and the duration of the operation. While that work considered only the class of strictly CPU-bound computations that do not depend on the input, the investigation of other factors (i.e., IO-bound operations; influence of varying input data) is planned. Note that the hen-and-egg problem of predictions before statistics are available is not solved yet, but could be tackled by other means, such as making pessimistic predictions based on other hosts' statistics, or "dry-running" operations.

Finally, it is worth noting that re-negotiation of agreements during the runtime of a process may be

required. One cannot preclude the possibility of service providers being unable to fulfill their guarantees (because of inaccurate predictions, or simply because of other failures). The same argument holds for exceptions triggered at the process level. In order not to have to abandon the entire long-running and valuable process, renegotiations should be supported and considered an integral part of failure recovery.

4. Related Work

The field of Scientific Workflows and their application has attracted a lot of research in the past years and has among others resulted in complete production-ready systems like Kepler [9], Taverna [10], JOpera [11], VisTrails [5], VIEW [8] or Trident [12]. Most of these systems feature support for SOAs by leveraging the Web Service stack of protocols, however they do not include support for concepts such as QoS, SLAs or ARs. In the closely related domain of Grid computing, several approaches such as [4] or [14] propose the use of ARs for pre-allocating resources to Grid jobs, with the latter following an approach similar to ours, i.e., using SLAs. Our work is slightly different in scope, as it considers the combination of several SLAs at the level of a workflow. In the general field of workflows and QoS, [1] presents a model for determining QoS characteristics for entire processes from QoS information of the underlying activities. While it addresses the computation and monitoring of QoS characteristics, it does not consider the usage of reservations or SLAs, nor the enforcement of QoS guarantees.

5. Conclusion

In this paper, we have presented our vision of a novel process execution system called DWARFS. Its characteristics, such as distributed orchestration and built-in support for large data volumes, make it particularly suitable for Scientific Workflows. Leveraging SLAs and providing tools to enforce them at runtime, in our opinion, will also deliver a significant added value to the end-user, namely a general increase in terms of Quality of Service and predictability of the processing.

While many of the presented aspects are still subject to research, the most important building blocks for the system to come into life have been identified, and to some extent implemented and proven to work. Our current and future work addresses topics at different levels of abstraction, such as the determination of predictions for data and timing issues; refinements of the planning model, and actual optimization strategies for it; protocols and specifications for representing the concepts and establishing agreements; strategies for the enforcement, and the detection of violations of, SLAs; renegotiation strategies and protocols.

References

[1] J. Cardoso, A. Sheth, J. Miller, et al. Quality of Service for Workflows and Web Service Processes. *J. Web Sem.*, 1(3):281–308, 2004.

[2] E. Deelman and A. Chervenak. Data Management Challenges of Data-Intensive Scientific Workflows. In *CCGRID*, pages 687–692, 2008.

[3] K. Droegemeier, D. Gannon, D. Reed, et al. Service-Oriented Environments for Dynamically Interacting with Mesoscale Weather. *Computing in Science and Eng.*, 7(6):12–29, 2005.

[4] I. Foster, C. Kesselman, C. Lee, et al. A Distributed Resource Management Architecture that supports Advance Reservations and Co-Allocation. In *Proceedings of the International Workshop on Quality of Service*, pages 27–36, 1999.

[5] J. Freire, C. Silva, S. Callahan, et al. Managing Rapidly-Evolving Scientific Workflows. In *Int'l Provenance and Annotation Workshop*, pages 10–18, 2006.

[6] O. Kopp, R. Khalaf, and F. Leymann. Deriving Explicit Data Links in WS-BPEL Processes. In *Services Computing*, volume 2, pages 367–376, 2008.

[7] C. Langguth and H. Schuldt. Enforcing Advance Reservations for E-Science Workflows in Service Oriented Architectures. In *Proceedings of the 3rd Workshop on Emerging Web Services Technology*, 2008.

[8] C. Lin, S. Lu, Z. Lai, et al. Service-Oriented Architecture for VIEW: A Visual Scientific Workflow Management System. In *IEEE SCC*, pages 335–342, 2008.

[9] B. Ludäscher, I. Altintas, C. Berkley, et al. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, 2006.

[10] T. Oinn, R. Greenwood, M. Addis, et al. Taverna: Lessons in Creating a Workflow Environment for the Life Sciences. *Concurrency and Computation: Practice and Experience*, 18(10):1067–1100, 2006.

[11] C. Pautasso, T. Heinis, and G. Alonso. JOpera: Autonomic Service Orchestration. *IEEE Data Eng. Bull.*, 29(3):32–39, 2006.

[12] RogerBarga, D. Fay, D. Guo, et al. Efficient Scheduling of Scientific Workflows in a High Performance Computing Cluster. In *Proc. CLADE*, pages 63–68, 2008.

[13] A. Shoshani, A. Sim, and J. Gu. Storage Resource Managers: Essential Components for the Grid. In *Grid Resource Management: State of the Art and Future Trends*, pages 321–340. Kluwer, 2004.

[14] M. Siddiqui, A. Villazón, and T. Fahringer. Grid Allocation and Reservation - Grid Capacity Planning with Negotiation-based Advance Reservation for Optimized QoS. In *Supercomputing*, page 103, 2006.

[15] J. Vazquez Fernandez. BPEL with Explicit Data Flow: Model, Editor, and Partitioning Tool. Master's thesis, University of Stuttgart, 2007.