Optimizing Resource Allocation for Scientific Workflows using Advance Reservations[⋆]

Christoph Langguth and Heiko Schuldt

Databases and Information Systems Group
Department of Computer Science, University of Basel, Switzerland
`{firstname.lastname}@unibas.ch`

Abstract. Scientific applications are more and more faced with very large volumes of data and complex, resource-intensive workflows that process or analyze these data. The recent interest in web services and service-oriented architectures has strongly facilitated the development of individual workflow activities as well as their composition and the distributed execution of complete workflows. However, in many applications concurrent scientific workflows may be served by multiple competing providers, with each of them offering only limited resources. At the same time, these workflows need to be executed in a predictable manner, with dedicated Quality of Service guarantees. In this paper, we introduce an approach to Advance Resource Reservation for service-oriented complex scientific workflows that optimize resource consumption based on user-defined criteria (e.g., cost or time). It exploits optimization techniques using genetic algorithms for finding optimal or near-optimal allocations in a distributed system. The approach takes into account the locality of services and in particular enforces constraints imposed by control or data flow dependencies within workflows. Finally, we provide a comprehensive evaluation of the effectiveness of the proposed approach.

Key words: Scientific Workflows, Advance Resource Reservation, Quality of Service

1 Introduction

Service-Oriented Architectures (SOA) have become widely adopted both in industry and research environments: standardized messages and message exchange formats such as WSDL and SOAP facilitate loose coupling, thus enabling service consumers and providers to interact in a much more flexible fashion than previously. One particularly interesting aspect of these SOAs is the possibility to combine several services into workflows (also known as "programming in the large").

Beyond the pure provisioning (or using) of functionality, however, both service providers and consumers usually have other interests: providers will strive

[⋆] This work has been partly supported by the Hasler Foundation within the project COSA (Compiling Optimized Service Architectures).

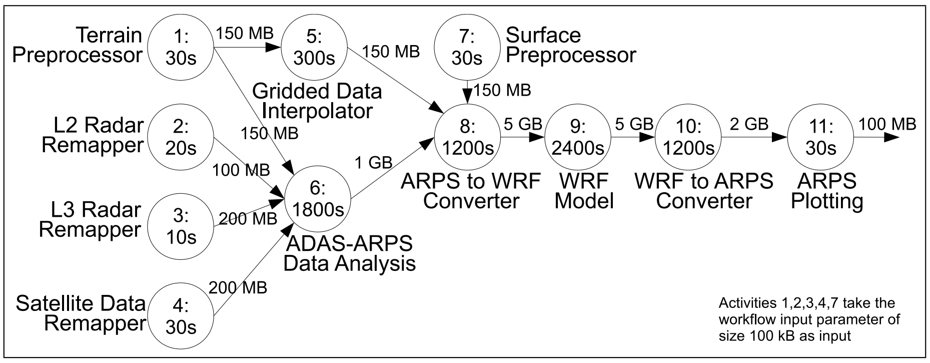

Fig. 1. Sample Weather Forecast Workflow

for the best possible usage of their provided resources in order to maximize profit; conversely, consumers may want to execute an entire workflow as fast as possible, or as cheap as possible (or combinations thereof).

Consider the sample workflow given in Figure 1, which is a simplified version of an actual scientific workflow presented in [5] and is used for producing weather forecasts. Reasonable non-functional criteria that an end user might specify for the execution (of the entire workflow) could be "as fast as possible", or "as cheap as possible, but with a deadline so that the results are available for the evening news". All of the operations are available as Web Services (WS) and may be provided by one or more service providers. Assume that details on timing and data quantities of individual operations are as indicated in Figure 1. This implies that the overall execution of a single instance of this workflow is in the range of several hours – the exact duration strongly depends on the available resources. Thus, in this constellation, we consider that workflow as a good example of a Scientific Workflow, as it is characterized by large volumes of data and contains long-running, CPU-intensive operations [17, 21].

To meet the aforementioned non-functional requirements, we propose a workflow engine (called DWARFS: Distributed Workflow engine with Advance Reservation Functionality Support) that is capable of providing Quality of Service (QoS) guarantees to the end users. When multiple independent clients run workflows concurrently, these clients are generally competing for the limited resources that providers have available. Our approach is based on the notion of Advance Resource Reservations (AR for short), which, in simple terms, reserves all required resources for running a workflow before the actual workflow execution.

The goal of the AR component of the DWARFS workflow engine is to find a combination of resource reservations at the providers of the individual operations that can satisfy the client's needs in terms of QoS, and to set up the required reservations, using negotiations based on WS-Agreement, so that the following execution of the workflow is guaranteed to meet these requirements. While we have presented an overview of the DWARFS system in [11], this paper focuses in detail on the planning phase. In particular, the contribution of this paper is an approach to make the execution of Scientific Workflows predictable by exploiting genetic algorithms for finding the optimal or near-optimal allocation of all resources needed at runtime (e.g., CPU, storage, specialized instruments, network bandwidth). The approach takes into account the locality of services and also enforces the different constraints imposed by control or data flow dependencies within a workflow. Finally, we provide a comprehensive evaluation of the effectiveness of the proposed approach.

The rest of this paper is structured as follows: Section 2 introduces the DWARFS system model. The optimization task is specified in Section 3. Section 4 presents in detail the application of genetic algorithms for advance reservation of resources. The results of a comprehensive evaluation of the DWARFS approach to AR is given in Section 5. Section 6 discusses related work and Section 7 concludes.

2 System Model and Assumptions

In this section, we shortly describe our overall approach, including basic assumptions. First and foremost, we assume that the operations that the workflow uses are provided in a SOA, i.e., that there are (possibly many) competing service providers that offer the required operations. We suppose that for the planning, we have the required knowledge about the infrastructure (such as which operation providers and workflow engines exist, network connectivity characteristics, etc.).[1]

2.1 Resources

Any operation invocation is assumed to require some resources for its execution. The most obvious case is that operation invocations require processing power (i.e., CPU cycles). However, "almost anything" involved in an operation can be considered a resource that needs to be taken into account when invoking an operation.

As an example, Figure 2 shows a more complete case of the resources that may need to be (co-)allocated for a particular operation execution at a specific service provider P: The actual operation requires reservations for CPU (2) and some hardware instrument (5). As the invocation takes place over the network, the system also considers reservations for bandwidth, required for the data upload (pre-invocation, (1)), and the download of the computed result (post-invocation, (3)), between the workflow engine E and the provider P. Finally, the provider specified that the client needs to reserve a certain amount of storage (4) for the entire duration of the operation call. Thus, even in this simple example, there are five resources involved, including dependencies on the timing of their allocation requirement. Note that the need for co-allocating multiple resources using such

[1] That information could for example be acquired using registries such as UDDI, or by any other means – an in-depth discussion is out of scope in this context.

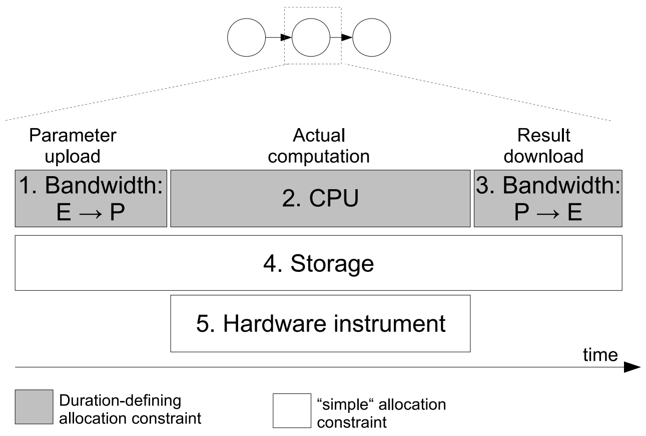

Fig. 2. Required Resource Allocations for Operation Call

dependencies is the rule, not the exception. Requirements to only allocate a single resource are rare, and are simply treated as the most elementary case of a co-allocation.

The information about which resources need to be allocated for a particular operation invocation is made available by the service provider. Staying with the example shown in Figure 2, a simplified form of these co-allocation constraints could look as follows:

- **CPU:** This invocation requires 25 billion CPU cycles and may use a maximum of 50% of the available computing power (for instance because this is a single-threaded operation deployed on a dual-core machine).
- **Hardware:** For the duration of the computation, you must reserve one mass spectrometer.
- **Storage:** For the duration of the invocation, you must reserve an amount of storage equal to twice the sum of the expected input and output sizes.

Finally, in addition to this information, every provider also makes available information about the current usage of each resource it provides (i.e., the amount of committed allocations already reserved by other clients; an example is shown in Figure 3), and the cost function for the respective resource. In the spirit of a SOA, all of this information is made available using standardized formats and usual mechanisms, e.g., embedded in, or referenced from, the service's WSDL or other public metadata.

These items also illustrate the relation between resource requirements and the actual duration of the resulting allocation: The duration is actually determined dynamically by the allocation request itself, and the available resources. While the provider specifies a maximum of 50% for the CPU allocation, clients may want to acquire less (say, a maximum of 20%, but a guaranteed minimum of 10% at all times), thus making the call take longer, but become cheaper. For more

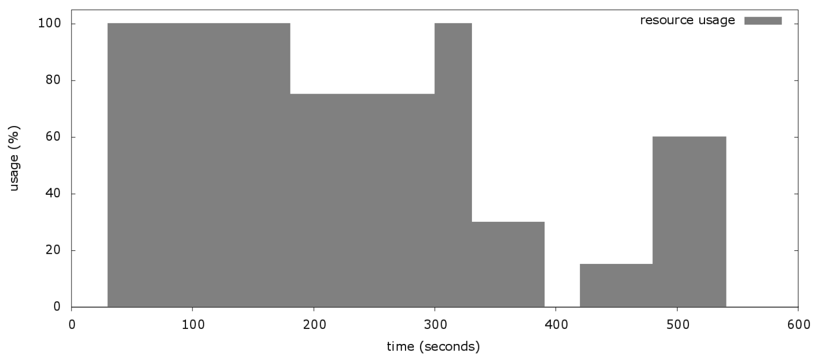

Fig. 3. Example of Resource Usage Representation

in-depth information about the calculation, provision, and usage of these data, please refer to [11].

2.2 Distributed Workflow Engine

While the discussion so far has mostly presented provider-side aspects, it has also touched one system-wide issue: the flow of data between the operation provider and the invoking workflow engine. Essentially, bandwidth capacity within the system is modeled as "just another resource" – so clearly, the network locations of the provider and the engine also influence the execution, especially when very large amounts of data are to be transmitted.

As opposed to traditional centralized workflow engines, DWARFS is meant to be a distributed workflow orchestration engine, which means that there are actually several co-operating workflow engines that can orchestrate a single workflow, and "move along" the network as the process execution evolves. Besides avoiding possible single hot-spots, this helps in leveraging operation locality – e.g., in the simplest case, choosing an execution engine that is close (in terms of network) to two consecutive operations in the workflow for invoking them will usually yield faster throughput. Finally, another advantage of executing long-running, resource-intensive Scientific Workflows by a set of distributed workflow engines is the possibility to seamlessly parallelize the orchestration of concurrent workflow instances across several engines.

3 The DWARFS Approach to Advance Reservation

As we have discussed, any workflow execution within DWARFS is subject to some user-defined QoS constraints. In order to meet these constraints, the resources that will be required at runtime need to be reserved in advance at the respective providers. In simple words, the question is thus: which combination of reservations fulfills the user's requirements best?

There are a multitude of variables that influence how suitable a particular combination is:

- **Chosen providers for operations:** There are possibly many providers offering the operations required for every single activity within the workflow, differing in the resulting cost, and execution time.
- **Operation providers' resource availability:** Every operation invocation needs to be able to reserve the required resources, as specified by the provider. However, because every provider keeps track of already allocated resources, not all theoretically possible reservation requests are actually feasible (or are only feasible at a later time, thus delaying the execution).
- **Resource allocation requests:** While providers state which resources are required and give bounds on their usage, this information may allow for flexibility. As shown in the previous section, some allocation requests may be modified to weigh execution time against cost.
- **Workflow Engine instance for activity:** Since we are dealing with a distributed execution engine, every workflow activity may be handled by any available workflow engine in the system. This choice again affects the overall planning, as for example network throughput (and availability) between different engines and operation providers differ.
- **Timing:** Because of the abovementioned aspects – especially those on the availability of resources – even small changes in the planned timing may substantially influence the overall outcome. As an example, starting the workflow execution a few seconds later may be able to use resources at a provider that was previously fully loaded, and thus result in a faster overall execution time.

Given the above discussion, it becomes clear that there is an extremely vast space of potential solutions to explore. For instance, the comparatively small experimental setup we used (see Section 5) has over 180 billion combinations of merely choosing a workflow engine and a provider for all activities – not yet counting the additional, essentially infinite, factor of possible resource allocation variations and timing offsets.

On the other hand, the inherent unpredictability of the system (caused by pre-existing provider-side resource allocations) makes it impossible to use traditional (precise) optimization techniques such as linear or non-linear optimization. Rather, this class of problem suggests using a metaheuristic approach.

We have investigated a number of possible metaheuristic approaches; in the end, we have decided to use Genetic Algorithms (GA), mainly because of three factors [4, 1]: i) in general, they avoid getting trapped in local optima as good as, or better than, other approaches; ii) a GA usually converges relatively fast; iii) GAs are a relatively simple and easy-to-understand, yet powerful concept.

4 The DWARFS Planner Implementation: Optimizing Allocations using a Genetic Algorithm

Genetic algorithms use an optimization approach that mimics natural selection as it happens in the real world: Each possible solution to the problem to be solved is represented as a genotype (or individual), represented by a single chromosome. The chromosome consists of a number of genes which can have different values (allele). Finally, a population consists of a number of individuals with different gene expressions. A fitness function is used to assign each individual within the population a value which determines its suitability in reaching the optimization goal.

The actual optimization then takes place by evolving the population into a new generation of individuals. For finding new problem solutions, two approaches are generally employed: mutation (i.e., randomly changing the value of a random gene), and crossover (mating of two individuals by recombining their genes). As these operations increase the value of genotypes, the fitness function is used to select the best individuals – considering both the originally existing individuals, and the newly created ones – and carry them over to the next generation (survival of the fittest).

4.1 Chromosome Layout

In trying to use a GA for our goal, the first question to be answered is: how can one represent a specific (possible) workflow instance, along with the required resource reservations, as a combination of genes that can be mutated and that form a chromosome? Because of the structure of a workflow definition, using a simple array of genes seemed too limiting. Rather, we use a feature of the JGAP framework [14] called SuperGene, which allows to build tree-like structures of genes. The actual allele values are then only stored in the leaf nodes (real genes in the GA sense) of the tree, but all internal tree nodes can be considered genes as well and as such mutated (delegating the actual mutation to child nodes).

Figure 4 shows an example of such a chromosome, representing a trivial workflow that consists of a single activity. Note that the representation of a workflow as a chromosome is directly derived from the workflow definition itself – there is no "one size fits all" chromosome representation suitable for all workflows. For instance, the sample process introduced in Section 1 results in a chromosome containing a total of 126 genes.

In the following, we will shortly walk through the example and explain the chromosome layout. For the genes that actually mutate (the leaf nodes of the tree), we provide a description of the values the allele can take. Note that many of the issues that have been mentioned in Section 3 have a direct correspondence with one of the presented genes.

– **WorkflowActivityGene**: This is the root gene of an entire workflow and essentially acts as a container for the other genes. It is also planned to be usable as a nested gene in order to represent sub-workflows (similar to the InvokeActivityGene), however this is left for future work.

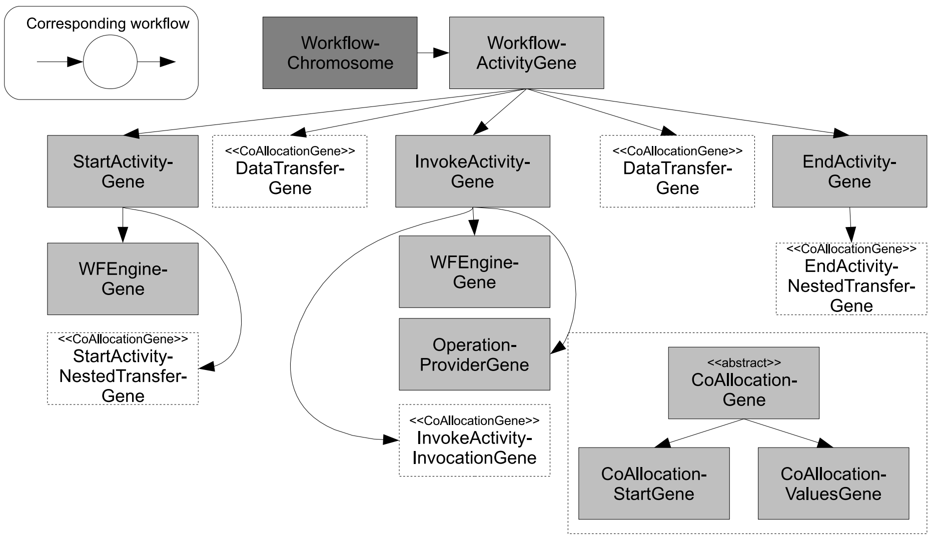
Corresponding workflow
Workflow-Chromosome
Workflow-ActivityGene
StartActivity-Gene
<<CoAllocationGene>> DataTransfer-Gene
InvokeActivity-Gene
<<CoAllocationGene>> DataTransfer-Gene
EndActivity-Gene
WFEngine-Gene
<<CoAllocationGene>> StartActivity-NestedTransfer-Gene
WFEngine-Gene
Operation-ProviderGene
<<CoAllocationGene>> InvokeActivity-InvocationGene
<<CoAllocationGene>> EndActivity-NestedTransfer-Gene
<<abstract>> CoAllocation-Gene
CoAllocation-StartGene
CoAllocation-ValuesGene

Fig. 4. Chromosome Layout for a trivial Process

- **StartActivityGene** and **EndActivityGene**: These genes represent explicit entry and exit points of the process. For example, all workflow activities that do not have at least one predecessor are represented by genes that will depend on the StartActivityGene, and the EndActivityGene depends on all activities not having successors. In addition, these genes make sure that the workflow start and end are actually scheduled at the same workflow engine, so that the SOA semantics of "a workflow invocation is perceived as just another web service" can be maintained.
- **WFEngineGene**: This gene represents the workflow engine executing the activity denoted by its parent gene. The possible allele values are in principle all workflow engines known to exist in the infrastructure.
- **StartActivityNestedTransferGene** and **EndActivityNestedTransferGene**: these genes represent the transfer of the input/output data from/to the end user, and the resources that are required for these transfers.
- **DataTransferGene**: These genes are inserted whenever data produced by one activity is required as input by another activity. In other words, they represent the data edges in the workflow graph[2]. Note that the actual resources that are affected depend on the source and target of the data transfer – i.e., if the source and target are actually the same workflow engine, this activity results in zero overhead, whereas physical network transmissions will require real allocations and influence the timing.
- **InvokeActivityGene**: This is a gene that models a remote operation invocation.

[2] For mere control flow edges, a (simpler) **ControlTransferGene** implementation exists (not used in the example).

– **OperationProviderGene**: This gene represents the chosen provider for the operation represented by its parent gene. Possible allele values are all providers known to offer the operation.
– **InvokeActivityInvocationGene**: This gene represents the actual invocation (and required allocations). It has dependencies on its parent and sibling genes for determining the allocations.
– **CoAllocationGene**: All genes that are shown in white dotted boxes are actually subclasses of this gene. The CoAllocationGene itself is responsible for finding the required allocations for the resources indicated by its subclasses.
– **CoAllocationStartGene**: This gene simply consists of an allele containing a number which influences the start timestamp at or after which possible allocations are to be found.
– **CoAllocationValuesGene**: This gene contains an allele that represents the variable part of the coallocation to be found. It is actually an array of values (so strictly speaking, contains more than one mutable part). For example, it may contain the minimum and maximum values to request for resources that have to be co-allocated.

4.2 Exploitation of the Information represented in the Chromosome

Every gene within the chromosome can be mutated independently, and every possible allele combination results in an – in principle – valid schedule (even if it may still be considered unsuitable with regard to the envisaged QoS constraints, e.g., because it takes longer than a user-specified deadline).

However, the alleles only denote the most basic information that is required to deduce the characteristics of the schedule that the individual represents. In fact, the interpretation of the genotype (answering questions such as: "when does it terminate, how much does it cost, which resources do I need to reserve?") is rather complex.

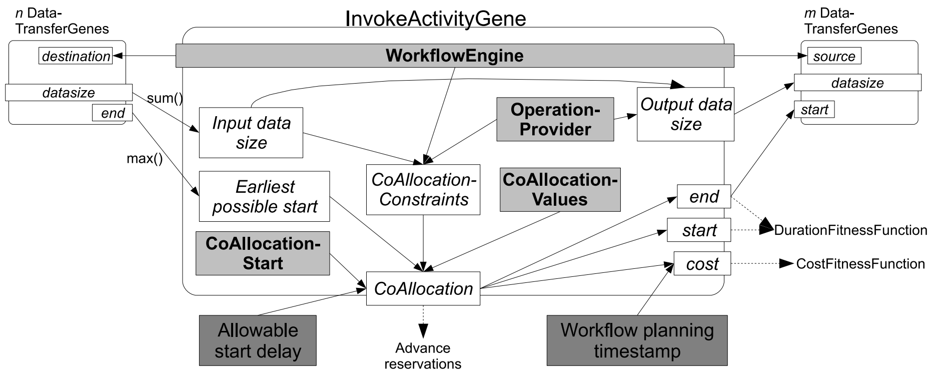

Fig. 5. Chromosome Interpretation and Interdependencies

Figure 5 presents a close-up look into the inner structure of the InvokeActivityGene, and its relation with its predecessor and successor genes. The gray boxes with bold text represent the actual alleles as introduced previously, while white boxes stand for computed values. The two dark gray boxes at the bottom can be considered static: they are given as input parameters for the optimization run. The arrows depict dependencies.

It is relatively straightforward to see that those dependencies are reflecting the structure, as well as integrity constraints, of the workflow. For example, the earliest possible start for an activity is the maximum of the timestamps when all of its predecessor activities have finished. All other genes have a similar structure, which in fact yields a directed acyclic graph of dependencies, when looking at the chromosome as a whole. This graph is spanning all of the alleles within the genotype: in order to determine e.g., the end timestamp of the EndActivityGene, all of the alleles of the entire workflow have to be inspected, and essentially all computations within the graph need to be performed. The computationally most expensive operation is to determine feasible co-allocations.

In order to reduce the computational overhead, we are caching already calculated results. When an allele is mutated, only the depending calculations are notified and re-performed, stopping whenever no further changes occur. This may still mean a considerable amount of re-calculations if an allele early in the workflow is mutated.

4.3 Fitness Functions

A fitness function assigns each chromosome a fitness value representing its suitability for solving the envisaged optimization criteria, and allows the algorithm to evolve towards better solutions.

We have implemented two straightforward fitness functions that represent the most sensible QoS requirements: A *CostFitnessFunction* which considers the total cost of all allocations generated by an individual, and a *DurationFitnessFunction* which considers the total runtime. Both functions have "smaller is better" semantics. While the raw values of cost or duration are easily understandable, they are not ideal as fitness measures. We rather perform the same multi-step transformation for both cost and duration, which results in normalized values. The fitnesses are calculated for the entire population at once, thus resulting in *relative* fitness values (i.e., how does the individual perform within this generation). The resulting values are such that each individual i of a population P is assigned a fitness value f_i, where $f_i \in [0, 1]$, and $\sum\limits_{i \in P} f_i = 1$. Larger values indicate a better fitness.

Even though this transformation results in the fitness values losing the clearly defined semantics the absolute values provide, they still remain correctly ordered and maintain the fitness proportions. However, this distribution offers two important advantages: i) the values are directly suitable for realistic crossover candidate selection, and ii) the values are combinable. While the first item is left for future work, the second can readily be used to formulate *combining fitness functions* such as "the cost is 10 times as important as the duration", thus allowing for essentially arbitrary combinations (and weighing) of otherwise incomparable QoS domains. We are also using the possibility of combining fitness functions for defining constraints like "Optimize for cost, but be ready by a certain deadline". In the latter example, as long as the deadline is not met by any individual (i.e., all would have a fitness of 0), the original weight factors of 1 and 0 are reversed, thus resulting in an (initial) optimization for deadline only until at least one individual meets the deadline constraints.

5 Evaluation

For evaluating the planner, we have used the workflow presented in Figure 1. For simplicity, the parameters (and thus the characteristics of the data and the operation runtimes) remain fixed for every workflow invocation, unlike the dynamic behaviour discussed in [11]. The workflow is scheduled 50 times on an initially empty infrastructure (i.e., no ARs exist before the first planning). The intervals between the planning runs are randomly chosen from a normal distribution between 0 (directly after the last planning run) and 3 hours later; additionally, the planned earliest start of the execution is uniformly distributed between 0 (directly after planning) and 5 hours after planning. The client that starts the invocation is randomly selected for each run. The QoS criteria that were used as optimization goal were also randomly chosen according to the following scheme (actual absolute numbers for the test run are in parentheses):

1. 30 % duration only (10)
2. 10 % cost only (8)
3. 20 % duration, but respecting a given budget (9)
4. 20 % cost, but respecting a given deadline (14)
5. 10 % weighing cost 80% over duration 20% (5)
6. 10 % weighing duration 20% over cost 80% (4)

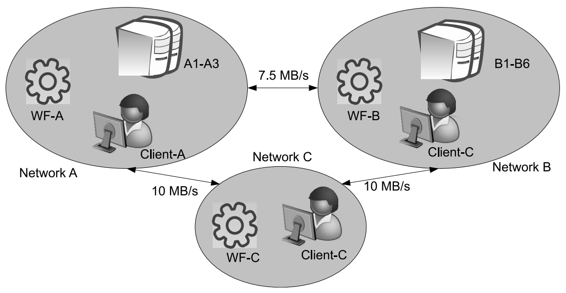

Fig. 6. Experimental Setup: Networks and Hosts

Host	*A1*	*A2*	*A3*	*B1*	*B2*	*B3*	*B4*	*B5*	*B6*
CPU class	medium	fast	fast	medium	medium	slow	medium	slow	fast
Storage	200GB	100GB	300GB	100GB	400GB	400GB	400GB	400GB	200GB
Op. 1	100%	100%	100%	100%					
Op. 2		25%	25%		50%				100%
Op. 3	25%			25%				25%	
Op. 4	25%	100%	50%	25%					
Op. 5			25%		50%				
Op. 6		100%					50%		100%
Op. 7	50%		100%		50%	50%			50%
Op. 8	50%		50%					25%	
Op. 9	25%	50%	50%			100%	100%		
Op. 10		25%				25%	50%		
Op. 11	25%	25%	25%					50%	

Table 1. Operation Provider and Host Specifications

5.1 Experimental Setup

The infrastructure that we simulated for all evaluations consists of 9 hosts that act as service providers, 3 available workflow engines, and 3 client hosts (from which the workflow instances are to be started). This setup is small enough to eventually saturate some hosts' resources, but large enough to induce a search space which is too large for naïve optimizations such as exhaustive search. All hosts in the system have a network capacity of 100 MB/s both for incoming and outgoing bandwidth, however the hosts are located in three separate networks which differ in available bandwidth between them. The networks and machines are shown in Figure 6.

The hosts acting as operation providers have been set up as shown in Table 1. This setup and deployment has been randomly auto-generated, so as to avoid any possible form of "human bias". We have knowingly refrained from using any kind of exact numbering (GHz, FLOPs or the like) for the CPU classes, but rather classified them as slow, medium, and fast; however, there is a linear relationship between these classes: medium is twice as fast as slow, and fast is three times faster. On the other hand, if the optimization goal is cost only (and only CPU is considered), it could be beneficial to use a slower host, as the "unit cost" for CPU are 0.25, 0.5, and 1.0 for slow, medium, and fast, respectively[3].

For a concrete example of how to interpret the table, consider operation 8 (ARPS to WRF Converter) of the sample workflow, which states 1200 seconds as its runtime. This number is to be interpreted as "If running on a fast host and at 100% of the CPU, it would take 1200 seconds". From the table, we can deduce that the fastest actually possible invocations for operation 8 are i) 3600

[3] The cost advantage may be outweighed by having to reserve other resources for the correspondingly longer time, though.

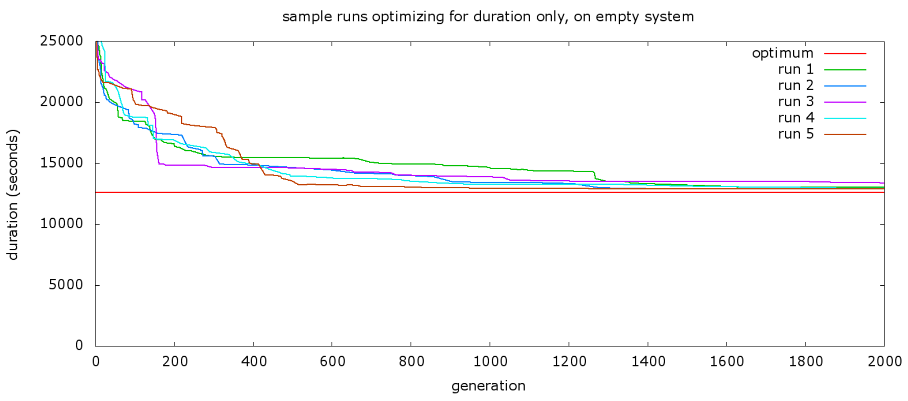

Fig. 7. Quality of the Evolution of the Planning using known Optimum

seconds if run on host A1 at 50%[4], ii) 2400 seconds if run on host A3 at 50%, or iii) 14400 seconds if run on host B5 at 25%.

All operations require reservations for storage exactly equal to the sum of the input and output sizes; all providers have the same cost function for storage, and all hosts have the same cost function for bandwidth. Please note that we currently apply a virtual unit for the costs. In future work, we plan to consider the exact costs incurring at the providers' sites by exploiting an economic model for ARs.

5.2 Evaluation: Result Quality of ARs for a single Run

Because of the reasons outlined in Section 3, it is generally not possible to analytically calculate how close the found solution is to the actual absolute optimum. However, we have manually calculated the absolute optimum for a feasible case, namely an optimization for duration only on an empty system, with the client being fixed to Client-C. The absolute achievable optimum in this case is 12597 seconds (approximately 3.5 hours). Figure 7 graphically shows how the algorithm evolved in 5 test runs, while Table 2 presents a more detailed analysis of 100 runs.

Figure 8 shows the CPU reservations that one of those runs has made at different providers. Note that at the beginning of the process, there are indeed overlapping boxes, representing parallel execution of activities on multiple hosts. The gaps between the shown reservations correspond to data transfers (we have not depicted bandwidth reservations, as those would clutter the figure). Finally, it is worth mentioning that the workflow is scheduled to be run by two different workflow engines: Activities 1 through 8 are executed by WF-A, then control (and data) is handed over to WF-B for executing activities 9 through 11. This is

[4] 3600 = 1200 * 1.5 (downgrade from fast to medium) * 2 (downgrade from 100% to 50%).

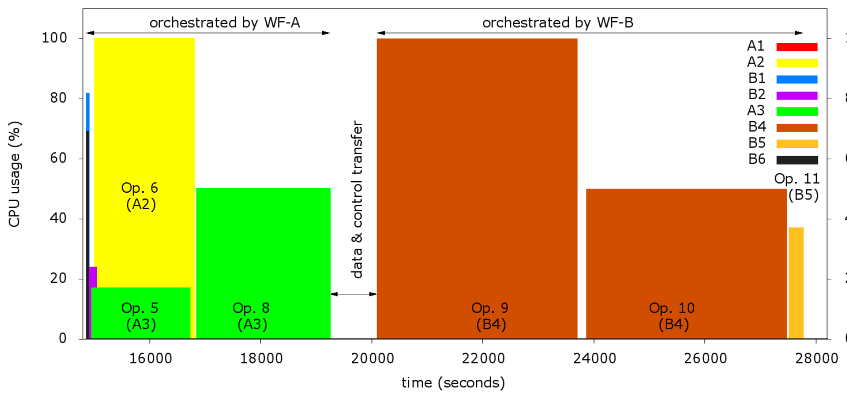

Fig. 8. Combined Usages of one single Process

one example of how multiple engines may leverage network locality to the target providers.

5.3 Evaluation of Resource Allocations after 50 Runs

Of the 50 planning runs, two did not achieve the required QoS criteria; both were optimizations for duration, with a minimal possible budget that was set too low. All other runs were successfully finished and made the resulting reservations in the system.

Figure 9 shows the evolution of cost and duration for one sample workflow scheduling. In this case, the goal was to optimize for cost, while keeping the duration of the process under a given deadline (120,000 seconds, which is roughly 10 times higher than the minimum that can be achieved when scheduling a single run on an empty system). The convergence rate of the criteria to be optimized is similar to the ones shown in Figure 7. Since cost is the primary optimization goal (which is achieved by using fewer resources and thus “traded” for runtime), and

criteria	*average*	*standard dev.*	*min.*	*max.*	*90th perc.*	*95th perc.*
duration	13116.88	228.26	12758	13865	13415	13606
% of optimum	104.1	1.8	101.3	110.1	106.5	108.0
effectiveness (%)	96.1	1.6	90.9	98.7	93.9	92.3
Minimum generation to reach effectiveness of:						
75%	159.77	87.71	28	498	263	321
90%	538.62	302.20	87	1892	942	1080
95%[a]	1073.38	392.74	400	1985	1643	1867

[a] Not all runs reached 95% effectiveness; the statistics are for the 79 runs that did.

Table 2. Qualitative Analysis of 100 Scheduling Runs with known Optimum

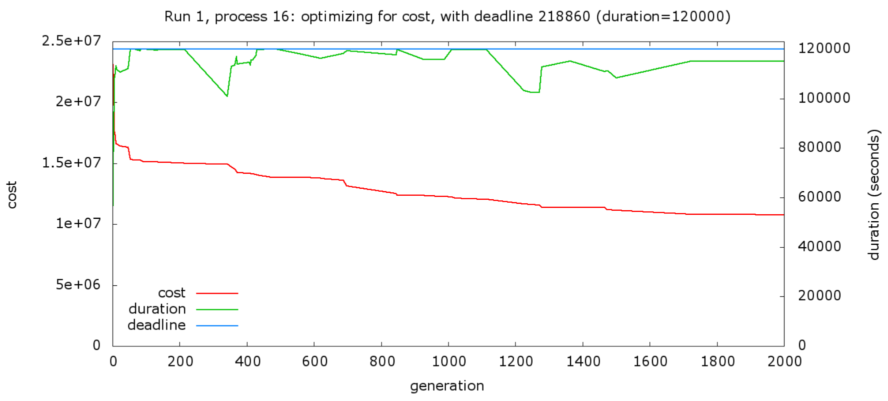

Fig. 9. Sample Process Optimization: Evolution of Cost and Duration

because resources are already partly allocated by previous runs, the duration is significantly higher than in the case depicted in Figure 7. In fact, the duration generally stays close to the deadline, as this allows to achieve the best cost.

As each successfully scheduled workflow reserves resources, the initially unallocated infrastructure is gradually getting filled. Figures 10 shows how the reservations of CPU at hosts B4 and B3 are evolving as new workflows are scheduled. The graphs are simple three-dimensional extrusions of the resource representation shown in Figure 3, where the added dimension (towards the "back" of the graph) is the number of already scheduled workflows.

We have chosen to present these two hosts because they exhibit an interesting case: B4 is much more heavily loaded than B3. The explanation lies in the setup of the infrastructure: B4 provides the fastest available alternative for the (long-running) operations 9 and 10, whereas B3 is slow. B3 is thus being used mostly by processes which prefer a cheap execution – which in turn means that even the

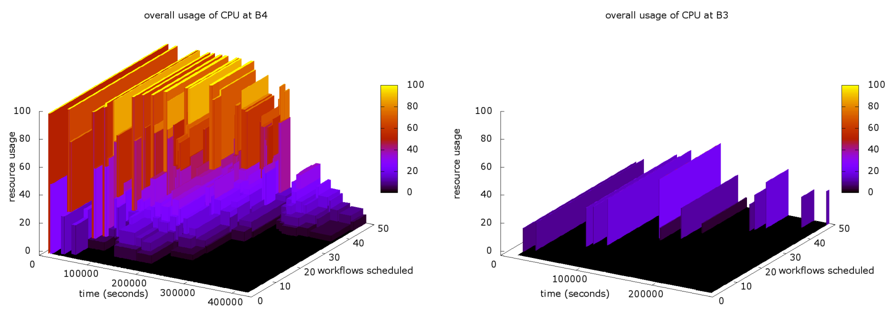

Fig. 10. CPU Reservations of Hosts B4 and B3 during Evaluation Run

slow CPU is not used up to its capacity –, or as an emergency alternative when all faster alternatives are booked out for so long that the slow host becomes the fastest available one.

5.4 Performance Evaluation

The experiment was run on a normal modern laptop. Each optimization run evolved a population of 32 individuals over 2000 generations. All runs took about 35 ms to evolve one generation, resulting in a total runtime of around 70 seconds per optimization.

Since the biggest share of the complexity of the calculations is in finding possible co-allocations, we also decided to evaluate a worst-case scenario, in which all clients (all 50 runs) would want to start the workflow at the same time, optimizing for duration only. This naturally results in the system usage rapidly evolving towards a state where almost all resources are fully loaded almost all of the time. This in turn means that finding "free spots" for allocations should take longer and longer with every scheduled workflow. Our measurements have confirmed this expectation: the worst-case behaviour shows a relatively linear increase in the time required for planning that starts at the 6th schedule attempt – i.e., once the infrastructure is "booked out" by the previously scheduled processes. The 50th run took around 11 minutes, i.e., 10 times longer than the average.

The AR approach presented in this paper explicitly addresses complex, long-running Scientific Workflows. For clients of these workflows, planning and advance reservation is a significant qualitative benefit that leads to predictable executions. In particular, due to the long execution time of a single workflow instance, the planning phase itself is not considered time-critical, and an overhead of several seconds up to a few minutes seem absolutely acceptable – especially since in the worst-case scenario with longest planning time due to already booked resources, possible allocations – and thus possible execution times – will be delayed anyhow (e.g., since a workflow cannot be started immediately due to lack of resources).

6 Related Work

Scientific Workflows have received considerable attention in the last years; this has among others resulted in complete production-ready systems like Kepler [13], Taverna [15], JOpera [16], Trident [19], VIEW [12] or VisTrails [7]. While they provide sophisticated support for the execution of Scientific Workflows or for analyzing the provenance of workflow results, none of them address the predictability of the execution.

To provide such predictability, several proposals have been made. For instance, [3] shows how QoS criteria for a workflow can be computed based on attributes of the contained services; in contrast to DWARFS, it does not attempt to actually enforce those criteria. In fact, most of the research to give

QoS guarantees comes from the related field of Grid computing: [6] and [20] propose using ARs for pre-allocating resources to Grid jobs. The GridCC project presents an approach that combines workflow execution with Advance Reservations [9]; our work differs in one important aspect: whereas in Grid environments, the user is supposed to know the resources that are required, we argue that in a truely service-oriented architecture, this is dynamic information that differs between – and must be obtained from – the service providers.

The actual optimization problem we presented is closely related to classical scheduling problems. Many publications have demonstrated that GAs are indeed an effective approach to solving scheduling problems at various levels of complexity, e.g., for scheduling jobs for multiprocessor systems [10] or in the Grid [8]. The work presented in [2], similarly to our proposal, uses GAs to find optimal workflow schedules according to QoS criteria; however, they do not explicitly consider the involvement of (limited) resources. Conversely, [18] presents how GAs can be used to schedule scientific workflows in an ASKALON Grid environment. Resource availability is taken into account by migrating tasks at runtime if resource shortage occurs, whereas DWARFS explicitly addresses the limitation of resources at planning time by using Advance Reservations, in an attempt to limit such rescheduling operations.

7 Conclusion and Future Work

In this paper, we have presented the DWARFS approach that employs Advance Resource Reservations to enable the predictable execution of Scientific Workflows with user-definable QoS criteria. Based on the providers' specifications of the resources needed for executing the operations they offer, DWARFS supports the definition and usage of reservations. To that end, we have devised a Genetic Algorithm that can be used to find near-optimal combinations of the required resource allocations to meet the requested QoS, and provided an evaluation of its qualitative and quantitative performance.

Our current and future work includes some further improvement of the GA implementation, with the two main topics being the addition of a crossover operator (which may increase the convergence rate), and an exploration of possible performance improvements in situations where the infrastructure is heavily loaded. While this optimization phase results in a plan of resource reservations to be made, the actual integration of these reservations with the SOA (i.e., the providers), using the WS-Agreement protocol, lacks an implementation and is part of our future work, as is the integration with the actual execution and enforcement of the resource reservations. We also plan to include more sophisticated strategies for shipping data within the workflow, and partial re-planning and re-negotiation of reservations during the execution of a workflow. The latter is of high practical relevance when, for instance, a provider fails to meet the QoS level that it committed to.

References

1. M. Affenzeller, S. Winkler, S. Wagner, and A. Beham. *Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications (Numerical Insights).* Chapman & Hall, 2009.
2. G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani. An approach for QoS-aware service composition based on genetic algorithms. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, 2005.
3. 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.
4. Y. Collette and P. Siarry. *Multiobjective Optimization: Principles and Case Studies (Decision Engineering).* Springer, 2004.
5. K. Droegemeier et al. Service-Oriented Environments for Dynamically Interacting with Mesoscale Weather. *Computing in Science and Eng.*, 7(6):12–29, 2005.
6. 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.
7. J. Freire, C. Silva, S. Callahan, et al. Managing Rapidly-Evolving Scientific Workflows. In *Int'l Provenance and Annotation Workshop*, pages 10–18, 2006.
8. Y. Gao, H. Rong, and J. Z. Huang. Adaptive grid job scheduling with genetic algorithms. *Future Gener. Comput. Syst.*, 21(1):151–161, 2005.
9. L. Guo, A. McGough, A. Akram, et al. QoS for Service Based Workflow on Grid. In *UK 2007 e-Science All Hands Meeting*, August 2007.
10. E. S. H. Hou, N. Ansari, and H. Ren. A genetic algorithm for multiprocessor scheduling. *IEEE Trans. Parallel Distrib. Syst.*, 5(2):113–120, 1994.
11. C. Langguth, P. Ranaldi, and H. Schuldt. Towards Quality of Service in Scientific Workflows by using Advance Resource Reservations. In *IEEE 2009 Third International Workshop on Scientific Workflows (SWF 2009).*
12. 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.
13. B. Ludäscher, I. Altintas, C. Berkley, et al. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience*, 2006.
14. K. Meffert et al. Java Genetic Algorithms Package. `jgap.sourceforge.net`.
15. 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.
16. C. Pautasso, T. Heinis, and G. Alonso. JOpera: Autonomic Service Orchestration. *IEEE Data Eng. Bull.*, 29(3):32–39, 2006.
17. B. Plale. Workload Characterization and Analysis of Storage and Bandwidth Needs of LEAD Workspace. In *Linked Environments for Atmospheric Discovery*, 2007.
18. R. Prodan and T. Fahringer. Dynamic scheduling of scientific workflow applications on the grid: a case study. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 687–694, New York, NY, USA, 2005. ACM.
19. 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.
20. 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.
21. Y. L. Simmhan, B. Plale, and D. Gannon. A Framework for Collecting Provenance in Data-Centric Scientific Workflows. In *ICWS '06: Proceedings of the IEEE International Conference on Web Services*, 2006.