Robust Execution of Service Workflows Using Redundancy and Advance Reservations

Sebastian Stein, Terry R. Payne, and Nicholas R. Jennings, *Fellow*, *IEEE*

Abstract—In this paper, we develop a novel algorithm that allows service consumers to execute business processes (or workflows) of interdependent services in a dependable manner within tight time-constraints. In particular, we consider large interorganizational service-oriented systems, where services are offered by external organizations that demand financial remuneration and where their use has to be negotiated in advance using explicit service-level agreements (as is common in Grids and cloud computing). Here, different providers often offer the same type of service at varying levels of quality and price. Furthermore, some providers may be less trustworthy than others, possibly failing to meet their agreements. To control this unreliability and ensure end-to-end dependability while maximizing the profit obtained from completing a business process, our algorithm automatically selects the most suitable providers. Moreover, unlike existing work, it reasons about the dependability properties of a workflow, and it controls these by using service redundancy for critical tasks and by planning for contingencies. Finally, our algorithm reserves services for only parts of its workflow at any time, in order to retain flexibility when failures occur. We show empirically that our algorithm consistently outperforms existing approaches, achieving up to a 35-fold increase in profit and successfully completing most workflows, even when the majority of providers fail.

Index Terms—Business process dependability, managing and adaptively controlling end-to-end dependability properties, managing, establishing, and assessing interorganizational trust relationships.

1 Introduction

Large-scale and open, distributed systems, including the web, peer-to-peer systems, and computational Grids, enable participants to share resources and services with each other across organizational boundaries (e.g., complex data-processing services running on expensive hardware or traditional business services that are accessed through software interfaces). To facilitate such interorganizational interactions, service-oriented computing is emerging as a popular approach for allowing consumers to dynamically procure services for complex business processes (or workflows). These workflows usually consist of multiple interdependent tasks, each of which can be completed by a number of competing providers that offer the same type of service at different levels of quality and price. In this context, it is increasingly common for service consumers and providers to agree on explicit service-level agreements (SLAs), which state the price and performance characteristics of a service (e.g., using WS-Agreement [1]). Employing such agreements, consumers are able to reserve services in advance to ensure they are available when needed, while providers can better schedule their resources [2].

Despite the existence of these advance reservation mechanisms, executing workflows in a robust, dependable manner in distributed systems is still an open challenge. In particular, participants in these systems are typically autonomous, self-interested agents that act according to their own decision-making mechanisms [3]. In more detail, in the interorganizational settings we consider, services are provided by distinct companies that will have very different, sometimes conflicting aims to the consumer. This means that their behavior is inherently uncertain—they may decommit from SLAs if they receive a better offer or wish to serve a more valued customer (possibly paying a penalty), and some may even default completely, perhaps to damage a competitor. This unreliability makes it vital to explicitly consider trust relationships with potential providers.

Unintentional failures further exacerbate this unreliability, as machines may crash due to hardware problems or malicious attacks. Whether intentional or not, a single failure can have a highly detrimental knock-on effect on other services that have already been reserved for future time slots, but which require the output of the failed service to proceed. Furthermore, service availability can change over time, resulting in the disappearance of some providers, but also offering opportunities as better providers enter the system. Hence, we believe that such uncertainty and dynamism must be considered, especially when a workflow has an intrinsic value and when it needs to be completed in a timely manner.

In this paper, we extend the state of the art in service selection by proposing a novel strategy that uses decision theory as a principled approach for reasoning about service uncertainty and taking actions to maximize the consumer's expected utility (i.e., profit). In particular, we build on our previous work on selecting multiple providers in parallel to mitigate their unreliability [4], but significantly extend it to cover dynamic environments where service availability

- *S. Stein and N.R. Jennings are with the School of Electronics and Computer Science, University of Southampton, Southampton, SO17 1BJ, United Kingdom. E-mail: {ss2, nrj}@ecs.soton.ac.uk.*
- *T.R. Payne is with the Agent ART Group, School of Computer and Mathematical Sciences, University of Liverpool, Ashton Building, Ashton Street, Liverpool L69 3BX, United Kingdom. E-mail: T.R.Payne@liverpool.ac.uk.*

Manuscript received 5 May 2009; revised 5 Oct. 2009; accepted 8 Feb. 2010; published online 15 Oct. 2010.
For information on obtaining reprints of this article, please send e-mail to: tsc@computer.org, and reference IEEECS Log Number TSCSI-2009-05-0113. Digital Object Identifier no. 10.1109/TSC.2010.47.

changes over time and where services are reserved in advance using explicit service-level agreements. To address these environments, our strategy reserves only part of a workflow at a given time (e.g., it may reserve services only for immediately executable tasks to retain high flexibility, or it may identify and reserve services for a number of highly critical tasks, if advance reservations are generally more reliable). It also constructs contingency plans where outcomes are uncertain (to predict the impact of failures and the cost of recovering from them) and refines initial high-level decisions at runtime to incorporate new information about the performance and availability of services (e.g., when a highly reliable service fails, it will reserve a new service immediately and adjust its strategies for later tasks in the workflow to get back on schedule).

By conducting a thorough empirical evaluation, we show that existing approaches are unsuitable for highly dynamic and uncertain environments, and we demonstrate that our proposed strategy outperforms the state of the art in most environments when services are unreliable (and also achieves good results when services never fail). In our experiments, we show that these trends hold over a range of environments, including where providers fail maliciously (i.e., do not pay compensation). In highly uncertain environments, we show that our strategy completes 86.1 percent of all workflows within its deadline, compared to only 1.6 percent achieved by current approaches (resulting in a 35-fold improvement in average utility).

The remainder of this paper is organized as follows: In Section 2, we begin by giving an overview of related work. Then, in Section 3, we formalize our system model, and, in Section 4, we outline our workflow execution strategy. In Section 5, we evaluate this and finally conclude in Section 6. In a separate online Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSC.2010.47, we show how our algorithm is applied to a real workflow.

2 Related Work

There is already some existing research that focusses on building more robust and dependable service-oriented systems. Here, a number of approaches use redundancy or replication on the provider's side to create fault-tolerant services [5], [6], but such work is typically ad hoc in nature and requires a considerable investment on the part of the provider. Other efforts rely on a certain level of cooperation between the consumers and providers, targeting mostly unintentional failures that arise due to infrastructure problems. This includes the WS-Reliability specification, which allows service consumers and providers to communicate over potentially unreliable transport networks [7]. Furthermore, much work has focused on developing transactional workflows, which are based on earlier work in database technologies and seek to ensure that the execution of a workflow results in a consistent outcome, even if some of the constituent tasks fail [8], [9]. This is achieved using specifications such as WS-Transaction, which allows service consumers to roll back service operations if later parts of the workflow fail. While such techniques may reduce the risk for the consumer, they rely on the cooperation of the service providers, and they are limited to settings where the cost of rolling back a service is minimal. This is unrealistic in scenarios such as cloud or Grid computing, where a service consumes valuable resources that cannot be recovered retrospectively.

As cooperative approaches cannot address intentional failures, other work uses game theory to analyze when providers may have incentives to fail deliberately or to lie about their capabilities [10], [11]. These approaches model providers as rational self-interested agents that act to maximize their utility. With such a model, it is then possible to design mechanisms that incentivize the providers to act in a certain manner—for example, to give accurate estimates about their completion probability [10]. However, these techniques have some drawbacks. First, it is difficult to accurately model the preferences of providers in such open and heterogeneous systems as we consider here. More specifically, some providers may purely aim to maximize profits, others are concerned about their long-term reputation with consumers, and some may act irrationally, e.g., due to bugs in their scheduling routines or because they use heuristic algorithms. Second, the interaction mechanisms between consumers and providers are often well-established and the consumer is not at liberty to change them. Instead, it must take the best course of action, given these mechanisms. For these reasons, it is more appropriate to use a decision-theoretic approach here, which models the behavior of providers probabilistically and can help the consumer choose the best course of action [12].

Now, some work already considers potential strategies that a service consumer can employ to deal with unreliable and highly heterogeneous providers. This usually focusses on service selection strategies that assign providers to workflow tasks based on nonfunctional properties, such as reliability, cost, and speed. These are closest to our work and therefore our empirical comparison in Section 5 will use them as benchmarks. In more detail, some work uses local constraints or simple preferential orderings to select a service for a given task [13], [14], e.g., to select the most reliable or fastest service, but consequently makes myopic decisions without considering their impact on the overall workflow. This can lead to overspending and missed deadlines, especially when services fail frequently. To some extent, this myopia can be addressed by reserving services in advance to maximize an overall aggregated quality-of-service measure, subject to constraints (e.g., a deadline or budget constraint). Typically, this measure is a weighted sum of various qualities, such as the overall reliability, duration, and cost of the workflow [13], [15]. A similar approach is taken in [16], which proposes a decision-theoretic measure to guide the reservation decisions. While these approaches consider service uncertainty by including qualities such as the reliability and availability of services, they tend to be vulnerable to failures. As they reserve only a single service for each task, one failure may result in the loss of the complete workflow or require time-consuming replanning. Finally, reserving all services in advance is risky, as the consumer may lose its reservation costs if some tasks fail.

In the following, we will build closely on existing models to describe our problem more formally. We will also return to some of the key service selection approaches mentioned here for our empirical evaluation.

3 Service-Oriented System Model

To allow us to devise a generic and principled approach, we base our work on an abstract system model, rather than any particular service standard or implementation. To this end, our model builds closely on previous work in the area [13], [16], [4]. In the following, we first describe the types of workflows we consider (Section 3.1), then detail the reservation process (Section 3.2) and finally discuss how services are invoked (Section 3.3).

3.1 Workflows

In this paper, we are interested in the execution of a single *workflow*, which is selected by the service consuming agent at time $t = 0$ (we assume that time passes in discrete integer time steps). This workflow is an abstract description of the types of services that are needed to fulfil some high-level goal, as well as appropriate ordering constraints. In practice, it may be constructed by a human user, selected from a workflow library or even synthesized by an automatic planner.

Formally, we define a workflow as a tuple, $W = (T, E, \tau, u)$, where $T = \{t_1, t_2, t_3, \ldots, t_n\}$ is a set of n *tasks* and $E : T \leftrightarrow T$ is a set of *precedence constraints*—a strict partial order over T, where $(t_1 \mapsto t_2) \in E$ means that task t_1 must be completed before starting t_2. The function $\tau : T \to \mathcal{T}$ maps each task in T to an abstract *service type*, where $\mathcal{T} = \{T_1, T_2, T_3, \ldots\}$ is the set of all service types. Finally, $u : \mathbb{Z}_0^+ \to \mathbb{R}$ describes the *reward* of completing the workflow at a given point in time t. This represents the value that the agent (or its owner) attaches to the workflow and may, in practice, be the expected financial gain of completing the workflow, or simply a private utility value [12]. It is defined by a *maximum reward*, $u_{\max}$, a *deadline*, $t_{\max}$, and a cumulative *penalty* for late completion, δ:

$$u(t) = \begin{cases} u_{\max}, & \text{if } t \le t_{\max}, \\ u_{\max} - \delta(t - t_{\max}), & \text{if } t > t_{\max} \wedge t < t_{\max} + \frac{u_{\max}}{\delta}, \\ 0, & \text{if } t \ge t_{\max} + \frac{u_{\max}}{\delta}. \end{cases}$$

We treat a workflow as *completed* when all tasks in T have been executed successfully in the order prescribed by E. The overall *profit* of a workflow execution is the difference between the reward (or 0 if W is still not completed at time $t_{\text{zero}} = \lceil t_{\max} + u_{\max}/\delta \rceil$) and the total cost it has incurred by reserving and invoking services.

This formalization of a workflow is sufficiently generic to apply in a wide range of settings. We note, however, that we have omitted some characteristics that may occur in concrete applications. In particular, tasks may need to be completed with a certain level of *quality*, the nature of which depends on the problem domain. For example, a video rendering task may require a minimum output resolution or frame-rate, or an optimization task might have to be solved within a given bound of the optimal. We note that this could be included in our model by attaching appropriate quality constraints to each task and then selecting only those services that meet these (as discussed in the following section). Other more complex models can be derived by making u conditional on the actual quality that is achieved. However, we leave this to future work, instead focussing here on a more general, domain-independent problem. For the same reason, we also omit the possibility that services expose transactional operations, as described in Section 2.

TABLE 1
Service-Level Agreement Terms

Term	Description
$s(o) : \mathcal{T}$	The *service type* offered (equal to requested type).
$t(o) : \mathbb{Z}_0^+$	The *starting time* at which the service can be invoked (equal to the requested time).
$c_{\text{r}}(o) : \mathbb{R}$	The *reservation cost*, which must be paid immediately by the consumer when entering the contract.
$c_{\text{e}}(o) : \mathbb{R}$	The *execution cost*, which is the remaining cost (after the reservation cost) that the consumer must pay when invoking the service.
$d(o) : \mathbb{Z}^+$	The *duration*, i.e., the number of time steps it will take for the service to complete.
$\delta_{\text{f}}(o) : \mathbb{R}$	The *failure penalty*, which is paid to the consumer when the service fails to complete successfully within the agreed duration.
$q(o)$	A domain-specific *quality-of-service* vector that describes other non-functional properties of the offer.

3.2 Service Reservation

To complete its workflow, the consumer needs to discover and reserve suitable services for each task in the workflow. We have adopted the contract-net protocol to model this process, as it is simple and has been widely used in distributed systems [17]. However, our approach does not depend on this particular protocol—the consumer could similarly contact a central UDDI registry, where providers deposit potential SLAs, or it could use the WS-Discovery protocol to locate services and then engage in a bilateral negotiation to elicit SLAs for different time slots. It may even combine different approaches, e.g., relying on the contract-net protocol to secure advance reservations, but invoking other services on demand using standard web service protocols.

To formalize this process in our model, the consumer may send a *call for proposals*, $\varphi : \mathcal{T} \times \mathbb{Z}_0^+$, to the service market to request a particular service type at a given time step. For example, $\varphi = (T_1, 2)$ indicates that the consumer requires service type T_1 to start at $t = 2$. In response to each call, the market returns a set of *offers*, O_φ. These are potential service-level agreements that the service providers participating in the system are willing to offer to the consumer. Each offer $o \in O_\varphi$ contains a number of terms, as given in Table 1. Most of these terms apply universally to a wide range of service types that are found in practice, including the service cost, invocation time, and duration. For more specific nonfunctional parameters, we use a generic quality-of-service vector, $q(o)$, which may include other parameters of interest to the consumer, including, for example, the output picture quality of an image rendering service or the optimality guarantees of a scheduling service. As discussed above, however, in order to retain a domain-independent model, these are not used further in this paper other than to restrict the choice of suitable offers.

Now, the process of requesting services and receiving responses may be repeated arbitrarily often during a given time step, but we assume that the offers returned for two requests with the same service types and times are always identical. Furthermore, we assume that the consumer has

TABLE 2
Trust Information (Outcome Probabilities)

Prob.	Description
$P_{\mathrm{f}}(o)$	The *failure probability* is the probability that the service will not complete successfully within the agreed duration and pay the failure penalty $\delta_{\mathrm{f}}(o)$.
$P_{\mathrm{d}}(o)$	The *defection probability* is the probability that the provider will fail to provide the service as described and also fail to pay the agreed penalty $\delta_{\mathrm{f}}(o)$ (e.g., if the provider crashes, leaves the market or maliciously defects).
$P_{\mathrm{s}}(o)$	The *success probability* is the probability that the provider will successfully complete the requested service within the agreed duration.

some information about the probabilities of the possible outcomes of each offer, as shown in Table 2. Together, these probabilities describe all possible, mutually exclusive outcomes of an offer, such that $P_{\mathrm{f}}(o) + P_{\mathrm{d}}(o) + P_{\mathrm{s}}(o) = 1$. To retain a simple, generic model, and in line with existing work, we assume that the outcomes of any two offers are independent.

During the same time step, the consumer may reserve any number of these offers for the tasks of its workflow. To do this, it sends a single acknowledgment to the market, which maps offers to the corresponding tasks of the workflow. At this point, the consumer pays the appropriate reservation costs, and all other offers are assumed to be rejected. Importantly, the consumer may reserve several offers for a single task, in order to increase its overall success probability.

Now, this model makes a number of important assumptions that need to be justified. First, we assumed that outcome probabilities of different service offers are independent. We believe this typically holds, because services are executed on physically separate machines, will likely use different implementations, and do not directly interfere with each other. However, we recognize that correlated failures can occur, e.g., when several providers rely on a third-party service that fails, or when an infrastructure failure disrupts a large part of the network. At the simplest level, such correlation could be modeled through external events that happen with a certain probability. Alternatively, the consumer could derive complex joint probability distributions to express correlations between service outcomes. Both options represent realistic and feasible extensions to our model, but for reasons of space, we leave this to future work.

Our second assumption is that the outcome probabilities are known. While the consumer may have interacted with some providers in the past and thereby built up statistical distributions about their behavior, it is not realistic to assume this is available for all providers. This might be because interactions are too costly or because the population of providers changes frequently. Furthermore, providers might try to strategically exploit the consumer's perception of its success probability, possibly fulfilling a number of low-value offers before defaulting on a particularly critical and expensive service. However, we note that there is a growing body of work in the area of trust and reputation that deals with this problem [18]. Techniques from that field can be used to aggregate and exchange experience from many consumers using reputation mechanisms, newcomers can be modeled using Bayesian priors, which are updated as more information becomes available [19], and changes in behavior trends can be detected using heuristic exploration strategies [20]. As these issues are being addressed separately, we do not consider them directly in our work.

3.3 Service Invocation

During each time step, the consumer may also invoke the offers reserved for that step, provided that all relevant precedence constraints given by E have been satisfied. The outcome of the invocation is one of those listed in Table 2. More specifically, if the service is *successful*, we assume that the result is sent back to the consumer at the beginning of the time step at which the service was scheduled to end (i.e., at time $t = t(o) + d(o)$). To limit the scope of this paper, we assume the result is correct or that it can be easily verified.[1] If the provider does not return a correct result at this time and within the specification of the original offer o, it means that it has *failed* to honour the service-level agreement and must immediately pay the agreed penalty. If no payment is made, it is assumed that the service provider has *defected*. In summary, the consumer only discovers the outcome of a given invocation at the beginning of time step $t = t(o) + d(o)$, by which a successful result will have been received (*success*), a penalty will have been paid (*failure*), or nothing is received from the provider (*defection*).

4 Flexible Workflow Execution

In this section, we detail our workflow execution strategy. We begin in Section 4.1 with a brief overview, which is then elaborated upon in Sections 4.2 to 4.6.

4.1 Strategy Overview

As outlined in Section 1, we are interested in building a rational agent that acts to maximize its expected utility. For the purpose of this paper, we assume that the agent is risk-neutral and therefore that the utility it gains from executing a workflow is equal to the profit it makes.

Hence, we want our agent to adopt a *strategy* (an appropriate mapping from observed system states to actions) that maximizes the expected difference between the reward and cost of following it. Formally, given a workflow W and some probabilistic beliefs about the behavior of the market, we want to find strategy Ψ^*, defined as $\Psi^* = \mathrm{argmax}_{\Psi}\ \mathrm{E}(R(\Psi) - C(\Psi))$, where $R(\Psi)$ and $C(\Psi)$ are random variables describing the final reward and cost, respectively, when using strategy Ψ and $\mathrm{E}(\cdot)$ is the expectation operator.

However, finding Ψ^* is intractable for the same reasons as described in [4]. First, selecting service providers for the tasks of the workflows we consider is a combinatorial, NP-hard problem, even when all offers are known in advance and service behavior is deterministic. Second, calculating

1. We note that many computational problems that are difficult to solve can be verified easily, e.g., those in NP. When this is not the case, our approach, as described in Section 4, is inherently well-suited to deal with more general Byzantine faults, because it employs redundancy and can therefore use voting techniques to discard erroneous results. However, we leave this for future work.

TABLE 3
Task Strategy Parameters

Parameter	Description
$t_{\mathrm{a}}(\omega) : \mathbb{N}$	Number of time steps to reserve in advance.
$t_{\mathrm{w}}(\omega) : \mathbb{Z}^+$	Time interval to request services for.
$n(\omega) : \mathbb{Z}^+$	Maximum number of offers to reserve.
$\vartheta(\omega)$	Strategy for choosing offers to reserve when more than $n(\omega)$ offers are available.

the probability distribution for the duration of a workflow with uncertain task durations is known to be $\#P$-complete and this makes it difficult to calculate $\mathrm{E}(R(\Psi))$ [21]. Finally, encoding a potential strategy is far from trivial due to the potentially huge decision space. For these reasons, we adopt a heuristic approach in our work, which allows us to find good solutions in a reasonable amount of time. More specifically, we use local search techniques to find a strategy that maximizes the expected profit, we rely on fast approximations where analytical solutions are too costly, and we search a subset of potential strategies that consider only a limited number of contingencies.

In the remainder of this section, we first discuss the types of high-level reservation decisions and contingency plans we consider for each workflow task (Section 4.2). Then, we describe how these are used to estimate the overall utility (Section 4.3) and how task strategies are selected (Section 4.4). In Section 4.5, we discuss how the utility estimates are revised at runtime, and in Section 4.6, we summarize the consumer's behavior.

4.2 High-Level Task Strategies

In general, it is inefficient for a consumer agent to reserve services for all workflow tasks in advance. Doing so would restrict the agent unnecessarily, as it must commit to particular services and execution times, and is therefore inflexible when services fail. On the other hand, some providers may offer better service terms when reserved in advance, and the consumer should decide automatically whether it is appropriate to trade off a higher quality with decreased flexibility. To this end, our agent initially considers simple high-level *reservation strategies* that determine when and how it intends to submit a call for proposals for a given task, and how it will select from the returned offers. In this section, we formalize these strategies and discuss how they can be extended to express task strategies with contingencies.

4.2.1 Task Library

High-level reservation strategies are available to the consumer as a strategy library, $l : \mathcal{T} \rightarrow \mathcal{P}(\Omega)$, that maps each service type to a set of strategies (Ω is the set of all strategies). Each strategy $\omega \in \Omega$ is described by a number of parameters, as shown in Table 3. The first two of these prescribe how the consumer will formulate its call for proposals, e.g., if $t_{\mathrm{a}}(\omega) = 100$ and $t_{\mathrm{w}}(\omega) = 3$, it will request services 100 time steps in advance and for three consecutive time steps. The latter two describe how it will select from the returned offers. Here, we consider four selection strategies for parameter $\vartheta(\omega)$: $\{\texttt{cost}, \texttt{unreliability}, \texttt{end\_time}, \texttt{balanced}\}$. The first three indicate that the consumer will always choose the offers with, respectively, the lowest expected cost $(c_{\mathrm{r}}(o) + c_{\mathrm{e}}(o) - P_{\mathrm{f}}(o)\delta_{\mathrm{f}}(o))$, the lowest probability of not succeeding $(1 - P_{\mathrm{s}}(o))$ or the lowest end time $(t(o) + d(o))$. The selection strategy `balanced` will pick the offers that minimize a sum of these parameters, each normalized to the interval $[0, 1]$, so that 0 corresponds to the offer with the lowest parameter and 1 to the highest. We denote by ω_{null} the strategy to do nothing and stop the task.

TABLE 4
Average Performance Statistics When Following Strategy $\omega(\epsilon \in \{\texttt{success}, \texttt{unavailable}, \texttt{failed}\})$

Statistic	Description
$\check{c}_{\mathrm{r}}(\omega) : \mathbb{R}$	Average of the reservation cost.
$\check{c}_{\mathrm{e}}(\omega) : \mathbb{R}$	Average of the expected execution cost.
$\check{c}(\omega) : \mathbb{R}$	Overall expected cost $(\check{c}_{\mathrm{r}}(\omega) + \check{c}_{\mathrm{e}}(\omega))$.
$\check{p}(\omega, \epsilon) : [0, 1]$	Average of the probability of outcome ϵ.
$\check{d}(\omega, \epsilon) : \mathbb{R}$	Average of the expected time until outcome ϵ is known (measured from first time step that call for proposals was submitted for).
$\check{d}^2(\omega, \epsilon) : \mathbb{R}$	Average of the expected squared time until ϵ is known.
$\tilde{v}(\omega, \epsilon) : \mathbb{R}$	Variance of time $(\tilde{v}(\omega, \epsilon) = \check{d}^2(\omega, \epsilon) - \check{d}(\omega, \epsilon)^2)$.

Furthermore, we assume that the consumer has some performance information about each of the strategies, which it previously learned by observing the response of the market to various calls for proposals. Specifically, we assume that it has repeatedly submitted calls for proposals corresponding to its known strategies to the market. Then, using simple calculations,[2] it has recorded a number of statistical averages for the probabilities of various outcomes, for the expected costs and for the durations associated with the different reservation decisions (based on its trust information and without necessarily reserving and invoking any offers). This information is summarized in Table 4. Here, ϵ denotes the overall outcome of the strategy, with $\epsilon \in \{\texttt{success}, \texttt{unavailable}, \texttt{failed}\}$ (referring, respectively, to the events where at least one offer is successful, where no suitable offers were found and where some offers were invoked but failed). As with the outcome probabilities of a given offer, we here assume that this performance information is accurate. As discussed earlier, we note that obtaining this is not trivial and that changes in these statistics must be monitored carefully, e.g., when market conditions change.

4.2.2 Planning for Contingencies

The simple reservation strategies discussed so far allow the consumer agent to make some predictions about the likely outcomes, the cost and duration for completing a task, given that it adopts a certain strategy. However, assigning a single strategy to each task is unlikely to be sufficient in uncertain environments as the consumer needs some capabilities to plan for contingencies and predict their impact on the cost and feasibility of the workflow. Hence,

2. Example strategies and performance statistics can be found in the online Appendix, which can be found on the Computer Society Digital library at http://doi.ieeecomputersociety.org/10.1109/TSC.2010.47, to this paper. Full details are in [22].

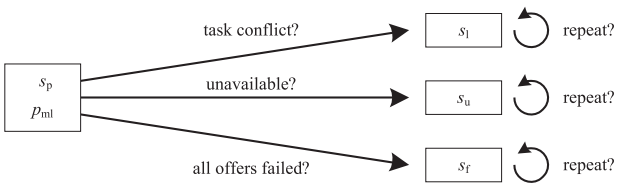

Fig. 1. Task contingencies.

we decided to include several contingent strategies that the consumer can fall back on.

These are shown in Fig. 1. Here, s_p is the main strategy for the task, but the consumer also has three other strategies to employ if s_p was unsuccessful:

- s_l is used to reserve further offers when the preceding tasks in the workflow have not been completed by the time the initial offers are available for invocation. In this case, the consumer will wait until the preceding tasks have completed and then follow s_l.
- s_u is used when s_p did not result in any reservations at all (e.g., if no offers were made). In this case, the agent waits until all preceding tasks have been completed and adopts s_u.
- s_f is adopted when the initial offers were invoked, but did not complete successfully. It is carried out as soon as the last offer completes unsuccessfully.

To further extend the number of strategies we consider, we note that the consumer might continue to repeat certain strategies until a task is completed (e.g., when the consumer does not have a tight deadline, it may decide to select the cheapest offer on the market, attempt it, and, in case of failure, simply try another cheap offer until the task is eventually completed). Hence, we extend the space of possible strategies for s_l, s_u, and s_f by adding a repeated strategy, ω_r for each $\omega \in \Omega$. The statistics for a repeated strategy ω_r can be directly derived from those for the nonrepeated ω (see [22]).

Now, when the service consumer plans to reserve a given task further in advance (indicated by a large $t_a(s_p)$), it is often desirable for it to do so earlier in the workflow, when some predecessors of the task may still be executing. This means that the consumer will waste less time waiting to invoke tasks after their predecessors have been completed, but it also increases the risk of conflicts with preceding tasks if these take longer than expected (e.g., due to failures or uncertainty about the offers that will be available). To express the risk the agent is willing to take, and to determine the time of making a reservation, we attach a *late probability*, p_{ml}, to each task. This is the largest acceptable probability when reserving services for task t_i that one of its predecessors will still not have been completed successfully by the reserved time step t_i. More formally, the consumer will reserve task t_i with primary strategy s_p at the earliest possible time step t where $p_p(t_i, t + t_a(s_p)) \leq p_{ml}$, and $p_p(t_i, x)$ is the probability that at least one of the predecessors of t_i has still not been completed successfully at time step x. Expressing the starting time of a task in such a way allows us to succinctly express when to start reserving offers relative to other tasks in the workflow.

Generally, as p_{ml} becomes smaller, the gap between the starting time of t_i and the end times of preceding tasks becomes larger. This means that the consumer may take longer to execute the workflow, but it also reduces the risk of losing reserved services. To estimate this delay (denoted $\hat{w}_i$), we examine the predecessors of t_i and determine the task during which reservation will take place so that the above condition for p_{ml} is satisfied. To this end, we consider the critical path[3] to task t_i. As described in [21], this is defined as the longest path to the task considering the complete duration of each preceding task (the sum of the expected duration $\bar{t}_i$ and the waiting time $\hat{w}_i$). We then proceed backwards along the critical path to identify the task during which to reserve services for t_i, as shown in Algorithm 1. Here, the input $\mathcal{C}$ is a set of tasks on the critical path to task t_i. The functions d, w, and v map each task to its respective duration, waiting time and variance (as w of a given task is established by the algorithm, we run it iteratively in topological order over the tasks).

Algorithm 1. Determining the reservation time.

procedure FINDRESERVATIONTIME $(\mathcal{C}, d, w, v, p_{ml}, s_p, t_i)$
  if $p_{ml} = 0 \ \vee \ |\mathcal{C}| = 0 \ \vee \ t_a(s_p) = 0$ **then**
    return $(t_i, 0, t_a(s_p), 0)$ ▷ No advance reservation
  $d_{pre} \leftarrow 0$ ▷ Total duration of tasks preceding t_i
  $v_{pre} \leftarrow 0$ ▷ Total variance of tasks preceding t_i
  $t \leftarrow -1$ ▷ Reservation time
  while $\mathcal{C} \neq \emptyset \wedge t < 0$ **do** ▷ Step backwards along $\mathcal{C}$
    $t_x \leftarrow$ element of $\mathcal{C}$ that is nearest to t_i
    $\mathcal{C} \leftarrow \mathcal{C} \setminus t_x$
    $d_{pre} \leftarrow d_{pre} + d(t_x)$
    $v_{pre} \leftarrow v_{pre} + v(t_x)$
    $t \leftarrow \lceil \Phi^{-1}_{m_{pre}, v_{pre}}(1 - p_{ml}) \rceil - t_a(s_p)$
    if $t < 0$ **then** ▷ Earlier reservation needed
      $d_{pre} \leftarrow d_{pre} + w(t_x)$ ▷ Add waiting time
  if $t < 0$ **then**
    $t_x \leftarrow$ none ▷ Reserve immediately
    $t \leftarrow 0$
  $\hat{p}_l \leftarrow 1 - \Phi_{m_{pre}, v_{pre}}(t + t_a(s_p))$ ▷ New late probability
  $w \leftarrow \int_t^{t + t_a(s_p)} \phi_{m_{pre}, v_{pre}}(x)(t + t_a(s_p) - x) \mathrm{d}x$
  if $t > 0$ **then**
    $w \leftarrow w + t_a(s_p)\Phi_{m_{pre}, v_{pre}}(t)$ ▷ Early completion
return $(t_x, t, w, \hat{p}_l)$

The algorithm returns a tuple $r = (t_x, t, w, \hat{p}_l) : ((T \cup \{\texttt{none}\}) \times \mathbb{N} \times \mathbb{R} \times [0, 1])$. Here, t_x is the task during which services for t_i should be reserved (or the special case none if services should be reserved immediately) and t is the time of making the reservation, relative to the starting time of task t_x (specifically, the first time step of any offers reserved for t_x). The returned value w is the expected amount of time between the last completion time of any of the predecessors of t_i and the first time step for which t_i was reserved—this is effectively the expected time that the agent will waste due to the advance reservation process. Finally, $\hat{p}_l$ is a revised late probability that is used by the consumer to update its calculations for the task ($\hat{p}_l \leq p_{ml}$).

3. This is a common approximation for workflow durations [21], [4].

Briefly, the algorithm begins in line 2 by considering the trivial case where $p_{\mathrm{ml}} = 0$, where the task has no predecessors, or where the reservation strategy contains no advance notice time. In these cases, the consumer will always start reserving services only when the task itself becomes available.

In all other cases, the algorithm will work backwards from task t_i along the critical path to find a suitable task t_x for commencing the reservation process. At each step, it estimates the time it will take from that task until t_i becomes executable by using a normal distribution with mean and variance equal to the sum of all duration means and variances along the path so far. Using the late probability p_{ml}, the algorithm then determines the earliest acceptable reservation time, relative to the start time of t_x (line 12). If this is negative, it continues to consider further predecessors of t_i. If no suitable task is found in the set of predecessors, the consumer will reserve services for the task immediately (i.e., $t_x = \texttt{none}$, line 16). Finally, the algorithm calculates the expected waiting time, considering both the case that the predecessors finish after the reservations were made but before t_i is started (line 19) and that they finish before any services for t_i are reserved (line 21).

It is important to note that the algorithm presented here is a heuristic approach for estimating the durations of tasks and for determining appropriate reservation times. It relies on several simplifying assumptions. Specifically, task durations are not independent of each other when the agent has reserved offers in advance (i.e., when one task takes longer, then this may have an impact on the duration of following tasks). Furthermore, our treatment of task waiting times simplifies the real problem, as they are not independent from task durations and may also lead to a reduction in variance along the workflow, which we ignore here. Finally, the algorithm uses a normal distribution even when considering a small number of tasks, and this can be inaccurate.

Despite this, we chose to adopt the algorithm to make fast predictions about waiting and reservation times. As we use an adaptive approach, these possibly inaccurate estimates are continuously revised during execution and eventually replaced by concrete offers (see Section 4.5). Furthermore, our empirical experiments in Section 5 show that our approach works well in practice.

To conclude, we note that, given a primary strategy, s_{p}, the contingency strategies, s_{l}, s_{f}, and s_{u}, as well as the late probability, $\hat{p}_{\mathrm{l}}$, it is easy to extend the statistics from Table 4 to the whole task by considering the possible outcomes of Fig. 1 (full details are in [22]). In the following, we use these to estimate the workflow utility.

4.3 Utility Estimation

As discussed in the previous sections, we can now calculate a number of performance parameters for every task of the workflow. More specifically, for each task t_i, these include an overall success probability, p_i, an expected execution cost, $c_{\mathrm{e}i}$, an expected reservation cost, $c_{\mathrm{r}i}$, an expected execution time, $\bar{t}_i$, the variance of the execution time, v_i, and an estimated waiting time, $\hat{w}_i$. These allow us to estimate the overall utility of the workflow in a similar manner as done in previous work [4].

First, the overall success probability of the workflow, p, is simply the product of all task success probabilities, $p = \prod_{i \in \mathcal{I}} p_i$, where $\mathcal{I}$ is the set of all task indices. Next, the overall expected workflow cost, denoted $\tilde{c}$, can be estimated by taking the sum of all task execution costs, each multiplied by the probability that they are reached, and all reservation costs, each multiplied by the probability that they are paid for, i.e., $\tilde{c} = \sum_{i \in \mathcal{I}} (c_{\mathrm{e}i} \prod_{j \in B_i} p_j \; + \; c_{\mathrm{r}i} \prod_{j \in B_{r(i)}} p_j)$, where B_i is the set of the indices of all tasks that precede t_i, and $r(i)$ is a function that returns the index of the task during which services for t_i will be reserved, as given by t_x in Algorithm 1 (if $t_x = \texttt{none}$, we assume $B_{r(x)} = \emptyset$).

We approximate the duration of the workflow again using the critical path and a normal distribution. To this end, we first attach a predicted completion time ($d_{i,\mathrm{end}}$) and variance ($v_{i,\mathrm{end}}$) to each task:

$$
\begin{aligned}
d_{i,\mathrm{end}} &= \hat{w}_i + \bar{t}_i + d_{i,\mathrm{pre}}, \\
v_{i,\mathrm{end}} &= v_i + v_{i,\mathrm{pre}}, \\
d_{i,\mathrm{pre}} &= \begin{cases} 0, & \text{if } B_i = \emptyset, \\ \max_{j \in B_i} d_{j,\mathrm{end}}, & \text{otherwise}, \end{cases} \\
v_{i,\mathrm{pre}} &= \begin{cases} 0, & \text{if } B_i = \emptyset, \\ v_{\operatorname{argmax}_{j \in B_i} d_{j,\mathrm{end}}}, & \text{otherwise}. \end{cases}
\end{aligned}
$$

Next, we estimate the overall workflow duration and variance using the task that is expected to finish last:

$$
\begin{aligned}
\lambda_{\mathrm{W}} &= d_{l,\mathrm{end}}, \\
v_{\mathrm{W}} &= v_{l,\mathrm{end}}, \\
\text{where } \; l &= \arg\max_{i \in \mathcal{I}} d_{i,\mathrm{end}}.
\end{aligned}
$$

Given these, we estimate the final expected reward conditional on overall success, denoted $\tilde{r}$, using the density function d_{W} of a normal distribution with mean λ_{W} and variance v_{W} to approximate the workflow duration, i.e., $\tilde{r} = \int_0^\infty d_{\mathrm{W}}(t) \cdot u(t) \mathrm{d}t$. This can be written in closed-form and quickly calculated as in [4]. Finally, we combine the parameters to derive an estimated workflow utility, $\tilde{u} = p \cdot \tilde{r} - \tilde{c}$. Next, we describe how we use this utility estimation technique to find a good strategy.

4.4 Optimization Algorithm

As a brute-force approach is clearly not practical, we use a local search algorithm to find a good overall strategy for the workflow. More specifically, we adopt simulated annealing, which is less prone to finding low-quality local optima than other local search algorithms [23].

This is detailed in Algorithm 2 and starts with an initial candidate solution, Ψ, which is an assignment of high-level strategies and maximum late probabilities for each workflow task, i.e., a tuple $(s_{\mathrm{p}i}, s_{\mathrm{u}i}, s_{\mathrm{f}i}, s_{\mathrm{l}i}, p_{\mathrm{ml}i})$ for each t_i (we will formalize this further in Section 4.6). Given this, the algorithm then repeatedly adds small random variations to Ψ (using the GENERATENEIGHBOR function in line 4, which we will elaborate on shortly), accepting it as the new candidate solution if it yields a higher utility than the original (line 7), or, with a certain probability, if its utility is less. As is common in simulated annealing, this probability

depends on the utility difference, an initial temperature Θ, a decay factor α, and the number of steps so far. The algorithm terminates after $n_{\max}$ steps or if a better solution has not been found after n_{fail} consecutive attempts (this applies only after the first $n_{\exp}$ steps, to encourage exploration initially).

Algorithm 2. Local Search Algorithm

```
1: procedure OPTIMIZE(Ψ, n_max, n_fail, n_exp,Θ,α)
2:     i, f ← 0
3:     repeat
4:         Ψ′ ← GENERATENEIGHBOR(Ψ)
5:         Δũ ← PREDICTUTILITY(Ψ′) −
           PREDICTUTILITY(Ψ)
6:         if Δũ > 0 then
7:             Ψ, f ← Ψ′, 0
8:         else
9:             x ← drawn uniformly at random from [0, 1]
10:            if x ≤ e^{Δũ/(Θα^i)}
11:                Ψ ← Ψ′
12:        i, f ← i + 1, f + 1
13:    until i = n_max ∨ (f > n_fail ∧ i > n_exp)
14:    return Ψ
```

For the neighbor generation in line 4, we first choose uniformly at random[4] whether to change the strategy associated with a particular task or the structure of the workflow. In the former case, we pick a random task t_i and randomly apply one of the following changes:

- All strategies ($s_{\text{p}i}$, $s_{\text{u}i}$, $s_{\text{f}i}$, $s_{\text{l}i}$) and the late probability, $p_{\text{ml}i}$, are reassigned randomly.
- A strategy, ω, is picked and changed to ω' so that exactly one of its parameters ($t_{\text{a}}(\omega')$, $t_{\text{w}}(\omega')$, $n(\omega')$, $\vartheta(\omega')$) is different from the original. This is done in one of four ways: by increasing or decreasing the parameter by a single step, or by randomly choosing one of the remaining higher or lower values.
- A strategy, ω, is picked and changed in one of three ways: to a random ω', to its repeated or nonrepeated equivalent, or to ω_{null}.
- The late probability, $p_{\text{ml}i}$, is changed to $p'_{\text{ml}i}$ in one of three ways: by randomly choosing a value from $(p_{\text{ml}i}, 1)$, from $(0, p_{\text{ml}i})$, or by setting $p'_{\text{ml}i} = 0$.

When altering the workflow structure, we change the precedence constraints E by either introducing or removing temporary edges. This allows us to represent the fact that the consumer may prefer to delay the reservation or invocation of certain tasks until the outcome of other tasks is known. For example, the consumer might decide to delay a particularly expensive task until it knows the outcome of another task that precedes it. Clearly, we pick only from new edges that do not introduce cycles and we update transitive dependencies.

In testing our optimization algorithm, we noticed that we could consistently improve its performance by making small adjustments, which are, for brevity, not shown in Algorithm 2. First, we apply an additional penalty to solutions that result in a negative expected utility, to generate a new expected utility value, $\tilde{u}'$, as follows:

$$\tilde{u}' = \begin{cases} \tilde{u}, & \text{if } \tilde{u} > 0, \\ \tilde{u} - \delta_{\text{fail}}, & \text{if } \tilde{u} \leq 0 \wedge \lambda_{\text{W}} \leq t_{\max}, \\ \tilde{u} - \delta_{\text{fail}} - \delta_{\text{late}}, & \text{otherwise,} \end{cases}$$

where $\delta_{\text{fail}} = (1 - p) \cdot u_{\max}$ and $\delta_{\text{late}} = (\lambda_{\text{W}}/t_{\max} - 1) \cdot u_{\max}$. This encourages the algorithm to avoid a common local maximum, where the consumer reserves only a few cheap services with a low probability of success, thus incurring a small overall loss. Second, we found that we could generally decrease the time to find a solution by immediately reconsidering the same neighbor generation strategy in line 4 if it resulted in a higher utility.

So far, we have discussed how the consumer can make high-level decisions about reserving offers for its workflow. In the next section, we describe how our mechanism is extended to deal with new information as it becomes available during execution.

4.5 Dynamic Adaptation

As we use a local search, our strategy is easily extended to incorporate information at runtime and act on it if necessary. For example, if reliable services suddenly fail, the agent may need to reserve further offers for the task and possibly even change its strategies for subsequent tasks, in order to meet its deadline. Similarly, the agent may come across new opportunities; for example, if it discovers a particularly attractive offer on the market and is able to immediately reserve it for a current task.

In this context, it is straightforward to incorporate new information into our calculations. First, when the consumer reserves services for a particular task (according to s_{p} and at the time determined by the procedure in Algorithm 1), we use the actual performance statistics of each reserved offer to immediately replace those for s_{p} (the detailed calculations can be found in [22]). This gives us a more accurate estimate of the probabilities of various outcomes, the late probability, the completion time, and the cost for the task. Similarly, as services fail, we remove them from their respective tasks, and when reservation or invocation payments are made, we remove them from the calculations as sunk costs.

Next, we also refine the overall completion time of the task. Specifically, we consider two cases: the preceding tasks finish in time for at least one of the reserved offers to be invoked or they finish too late for any offer to be invoked. In the former case, we derive a probability distribution for the completion time that assigns probabilities to the various end times of the reserved offers and a normal approximation if the reserved offers fail (using the performance statistics associated with the contingency strategy). In the latter case, when there is a conflict with the previous task, we use a normal approximation with mean $m_{i,\text{late}} = t_{\text{a}}(s_{\text{l}}) + \check{d}(s_{\text{l}}, \texttt{success}) + (1 - \mathcal{E}_i(\hat{t}_{\text{s}}))^{-1} \int_{\hat{t}_{\text{s}}}^{\infty} \mathcal{E}'_i(x)x \, \text{d}x$ and variance $v_{i,\text{late}} = \tilde{v}(s_{\text{l}}, \texttt{success}) + (1 - \mathcal{E}_i(\hat{t}_{\text{s}}))^{-1} \int_{\hat{t}_{\text{s}}}^{\infty} \mathcal{E}'_i(x)x^2 \, \text{d}x - m^2_{i,\text{late}}$, where $\hat{t}_{\text{s}}$ is the latest starting time of any offer reserved for task t_i and $\mathcal{E}_i$ is a cumulative distribution function for the completion time of the predecessors of t_i. This is either obtained using a simple normal approximation or, if concrete offers have been reserved for the predecessors, using the equations described here.

4. All random choices in this section are made uniformly at random.

Now, combining the two cases described above into a single distribution (each occurring with probability $1-\hat{p}_{\mathrm{l}}$ and $\hat{p}_{\mathrm{l}}$, respectively) gives us a more accurate estimate of the completion time for the task, as we now take into account the reserved offers. We then use this instead of the simpler normal approximation in all calculations.

Furthermore, we modify the neighbor generation procedure described in Section 4.4 to consider adding to or removing offers from a previously reserved task. These are chosen randomly from all available offers or from the set of offers that the agent plans to reserve during that time step (as we will discuss in the next section, offers are not reserved until the end of a time step). More specifically, in addition to changing the structure and high-level reservation strategies during the neighbor generation procedure, we include the possibility of changing a reserved task. When this occurs, we select a random task that has some associated reserved offers (denoted by the set $\gamma_i \subseteq \mathbb{C}$), and randomly carry out one of the following changes:

- **Add offer:** We first sample a value t_{r} from an exponential distribution with mean $\lambda^{-1} = \frac{1}{|\gamma_i|} \cdot \sum_{o \in \gamma_i} t(o) - t_{\min}$, where $t_{\min}$ is the lowest starting time in γ_i (when $\sum_{o \in \gamma_i} t(o) - t_{\min} = 0$, we use $\lambda^{-1} = 1$). Then, we submit a request for offers for time step $t = t_{\mathrm{r}} + t_{\min} - 20$, and add a random returned offer to γ_i. This process allows us to select a random offer, but with a bias toward offers at a similar time as those already in γ_i.
- **Remove offer:** If $|\gamma_i| > 1$, select a random offer that has been added to γ_i during the same time step and remove it again.

Finally, we also modify Algorithm 1 to terminate its main loop when it examines a task for which offers have already been reserved. This is because the agent has already decided when to start invoking that task and, as a result, the normal approximation will be far less accurate. If t is still negative at this stage, we use the starting time of the task as an anchor and infer the absolute reservation time from there (e.g., if $t = -10$ and the earliest reserved service is to start at time step $t(o) = 120$, the algorithm returns the time $t = 110$ and specifies the target task $t_x = \texttt{none}$ to signal that services should be reserved at an absolute time step).

We now summarize our strategy in the form of an overall decision-making algorithm.

4.6 Overall Algorithm

In this section, we can now detail the strategy that addresses the optimization problem outlined in Section 4.1. In particular, building on the work described in previous sections, we define the strategy Ψ more formally as a tuple $\Psi = (\alpha, \beta, \gamma, d_\beta, d_\gamma, E')$, where α, β, and γ are a set partition of T, describing the current state of each workflow task. Here, α contains the tasks that have been completed successfully, β contains the tasks for which some offers have been reserved, and γ contains the tasks for which no offers have been reserved. The functions d_β and d_γ provide further information about the agent's high-level decisions for the members of β and γ, respectively. Based on previous sections, $d_\beta(t_i)$ of a reserved task $t_i \in \beta$ is $d_\beta(t_i) = (\gamma_i, s_{\mathrm{l}i}, s_{\mathrm{u}i}, s_{\mathrm{f}i})$, where γ_i is the set of offers already reserved for t_i, while the other variables refer to the contingent strategies.[5] Similarly, $d_\gamma(t_j)$ of a task $t_j \in \gamma$ is $d_\gamma(t_j) = (s_{\mathrm{p}j}, s_{\mathrm{l}j}, s_{\mathrm{u}j}, s_{\mathrm{f}j}, p_{\mathrm{ml}j})$, where $s_{\mathrm{p}j}$ is the primary reservation strategy and $p_{\mathrm{ml}j}$ is the late probability. Finally, $E' : \mathcal{P}(T \times T)$ is the set of current precedence constraints.

Given this, Algorithm 3 contains a high-level overview of our strategy. At time $t = 0$, the consumer creates an initial execution strategy Ψ to form the basis of its local search[6] (line 2). Then, at each time step, the consumer first updates its current plan with any service outcomes (line 5), followed by an optimization process that refines the plan by changing its high-level task strategies and by altering already reserved offers (line 7), as described in the previous two sections. In line 8, the agent considers the reservation of due tasks, as determined by Algorithm 1. It does this by carrying out the associated primary strategy, but only temporarily associates the chosen offers with the workflow (they are not yet explicitly reserved). If any such reservations are added to the workflow, the consumer then repeats the optimization stage, so that the initially chosen offers can be improved (and possibly replaced by better ones), and this continues until no more new offers are reserved.

Algorithm 3. Summary of Flexible Strategy

```
 1: t ← 0
 2: Ψ ← create initial strategy
 3: abandoned ← false
 4: repeat
 5:     Ψ ← update strategy with recent service outcomes
 6:     repeat
 7:         Ψ ← local search for better strategy
 8:         Ψ ← use high-level strategies to reserve services
 9:     until Ψ was not altered in line 8
10:     if PREDICTUTILITY (Ψ) > 0 then
11:         reserve new services
12:         invoke services that are due
13:     else
14:         abandoned ← true
15:     t ← t + 1
16: until abandoned = true or workflow completed
```

Following that, if the consumer expects to receive a positive utility from continuing, it reserves any new offers that have been added to the workflow during that time step and invokes due services (lines 11 and 12). This procedure continues until the consumer either expects a nonpositive utility or the workflow is completed.

Clearly, it is time-consuming for the service consumer to carry out a long optimization stage during every time step—especially as the expected utility of the workflow may not change at all. Hence, we have found it sufficient to carry out further optimization only when its expected utility changes significantly from an earlier estimate, and also to

5. In a slight departure from our previous notation, we add the subscript i to refer to the task t_i.

6. In our work, we start with a simple allocation that uses ω_{r} with $t_{\mathrm{a}}(\omega_{\mathrm{r}}) = 0$, $t_{\mathrm{w}}(\omega_{\mathrm{r}}) = 10$, $n(\omega_{\mathrm{r}}) = 1$, and $\vartheta(\omega_{\mathrm{r}}) = \texttt{unreliability}$ as the primary and contingent strategies (all repeated) for every task and set $p_{\mathrm{ml}i} = 0.01$. This already constitutes a feasible strategy in most environments, thus, typically leads to a quicker convergence than a completely random initial strategy.

TABLE 5
Simulated Annealing Parameters

	$n_{\max}$	n_{fail}	n_{exp}	Θ	α	Threshold
initial	-1	5000	2000	100	0.999	-
short	-1	75	200	50	0.99	0.2
long	-1	1000	500	50	0.99	0.4

vary the amount of time spent during the optimization depending on the magnitude of the change.

More specifically, we have experimented with various optimization strategies and found the following approach to work quickly and effectively in a variety of environments. First, we always carry out an extensive initial simulated annealing run with the parameters given in the first row of Table 5. This is repeated up to three times if the resulting allocation does not yield a positive expected utility. Then, at each time step, we calculate the difference between the current expected utility and the total costs incurred so far. We carry out a "long" optimization run (see Table 5) if this value is at least 40 percent higher or lower than the same value when this was last run. Otherwise, if it is at least 20 percent higher or lower than after the last optimization run, we run a "quick" optimization procedure. It should be noted that these parameters can be easily adjusted for particular problems. For example, when time is critical, $n_{\max}$ can be set to a fixed cutoff time.

The online Appendix of this paper outlines how our approach reserves services for a real bioinformatics workflow. In the following section, we detail a number of experiments we carried out to test its performance.

5 Empirical Evaluation

As our approach is heuristic and due to the difficulty of finding an analytical solution (as described in Section 4.1), we have conducted a thorough empirical study of our algorithm in a simulated environment and compared it to a number of current approaches. The primary focus of this section is to investigate the feasibility of our approach in environments of varying uncertainty (i.e., where services are more or less likely to fail) and also in environments where the market favors certain reservation approaches (e.g., where early reservations are rewarded by more reliable services). In the following, we first describe how we simulate the market (Section 5.1), then we detail the strategies we test (Section 5.2) and the hypotheses that guide our investigation (Section 5.3). Finally, we describe our results (Sections 5.4-5.6).

5.1 Market Setup

In our experiments, we assume that there are five different service types ($\mathcal{T} = \{T_1, T_2, T_3, T_4, T_5\}$). To simulate the market, we keep a list of currently available offers associated with each time step, from the current step t to $t + 250$ (hence, the consumer may reserve services up to 250 time steps in advance). During the simulation, at the beginning of each time step, we first generate new offers that become available in the market by drawing the number of new offers and their parameters from random distributions. More specifically, for each time step in the list $(t, t+1, \ldots, t+250)$, we generate offers using the distributions in each row of Table 6. First, we generate the number of offers by drawing a sample from a Poisson distribution with a mean given by the birth rate in that row.[7] Then, for each such generated offer, we assign it the service type given in the table and draw a value for the reservation cost, execution cost, and service duration from the specified distributions.[8] All other offer parameters, such as the failure probability and penalties, are determined according to our experimental parameters detailed below. At the end of each time step, we remove offers in a similar way by drawing a random sample from a Poisson distribution with its mean given by the death rate. This models the demand for such services and we remove the generated number of offers from that time step (or all if the number exceeds the supply).

TABLE 6
Service Type Parameters

Row	Type	Reserv. Cost	Exec. Cost	Time	Birth Rate	Death Rate
1	T_1	$\mathcal{U}_h(25)$	$\mathcal{U}_h(25)$	$\mathcal{U}_h(5)$	$r_b/2$	$r_d/2$
2	T_1	$\mathcal{U}_h(5)$	$\mathcal{U}_h(5)$	$\mathcal{U}_h(40)$	$r_b/2$	$r_d/2$
3	T_2	$\mathcal{U}_h(1)$	$\mathcal{U}_h(5)$	$\mathcal{U}_h(50)$	r_b	r_d
4	T_3	$\mathcal{U}_h(10)$	$\mathcal{U}_h(10)$	$\mathcal{U}_h(35)$	r_b	r_d
5	T_4	$\mathcal{U}_h(50)$	$\mathcal{U}_h(1)$	$\mathcal{U}_h(25)$	r_b	r_d
6	T_5	$\mathcal{U}_h(1)$	$\mathcal{U}_h(50)$	$\mathcal{U}_h(25)$	r_b	r_d

In our simulations, a consumer is rewarded a maximum utility of $u_{\max} = 2{,}000$ for completing a workflow, with penalty $\delta = 40$ and deadline $t_{\max} = 200$. Each workflow consists of eight tasks (with random types) and we generate them by randomly filling an adjacency matrix until at least a quarter of the total number of possible edges have been added, thus ensuring that there are several parallel and sequential tasks in the workflow.

We chose these parameters to represent a realistic and challenging scenario with a relatively short deadline, but a sufficient maximum utility to allow the agent to afford a number of failed service invocations in uncertain environments. The workflows we test here are small, because the existing work that relies on integer programming techniques was unable to deal with larger cases. This is because we consider environments with potentially hundreds of offers for each task and solving this optimally can take hours or longer for larger workflows. In particular, even in the current scenario, finding a solution sometimes took longer than five minutes, which we believe is unacceptable in the time-critical settings we cover in this paper. Despite this, we compared our approach to the remaining strategy in other environments, including larger workflows with up to 50 tasks, and obtained the same broad trends as presented here (see [22] for details).

5.2 Strategies

We evaluate the performance of four strategies: the first three are based closely on the work presented in [13], but,

7. This is a common distribution for modeling random arrival events. We use $r_b = r_d = 0.005$, unless noted otherwise.

8. We use $\mathcal{U}_h(m)$ to refer to a uniform distribution with mean m that varies around m by a proportion of at most h, i.e., $\mathcal{U}_h(m)$ is a uniform distribution on the interval $[(1-h)\cdot m, (1+h)\cdot m]$. We use $h = 0.2$ in all our experiments, indicating a fairly high heterogeneity of offers.

more generally, represent common service selection approaches that are widely used in the literature, such as [15]. The fourth is the flexible reservation strategy proposed in this paper. We briefly describe each below.

5.2.1 Local Weighted Optimization

This strategy procures services completely on demand (i.e., only when the respective task becomes available). At this time, the consumer considers all offers in the next n time steps (we set $n = 20$ as this produces good results for the environments we consider) and then reserves the offer o^* that maximizes a weighted sum:

$$o^* = \underset{o}{\operatorname{argmax}} \sum_{i=1}^{3} w_i \cdot Q_i(o), \tag{1}$$

$$Q_i(o) = \begin{cases} 0, & \text{if } q_{\max,i} = q_{\min,i}, \\ \frac{q_{\max,i} - q_i(o)}{q_{\max,i} - q_{\min,o}}, & \text{otherwise}, \end{cases}$$

where $q_1(o) = c_{\mathrm{e}}(o) + c_{\mathrm{r}}(o)$ is the combined total cost of the offer, $q_2(o) = 1 - P_{\mathrm{s}}(o)$ is the probability that the task will not succeed and $q_3(o) = t(o) + d(o)$ is the end time of the offer. The values for $q_{\max,i}$ and $q_{\min,i}$ are the largest and smallest of these parameters among the offers that are considered, and each weight $w_i \in [0, 1]$ attaches a relative importance to the associated parameter (with $\sum_i w_i = 1$). We also assume that the strategy will immediately attempt any failed tasks again. For the purpose of our experiments, we set $w_1 = w_2 = w_3 = \frac{1}{3}$, which strikes a balance between the various qualities (in most environments, we did not observe a significant difference in performance when adopting other weights).

5.2.2 Global Weighted Optimization

This is perhaps the most widely adopted approach for reserving services in the literature [13], [15]. Here, the agent observes the market once, then selects and reserves one offer for each task, so that a weighted sum similar to (3) is maximized. This sum now aggregates the quality parameters over the entire workflow and may contain constraints, such as an overall budget or time limit. Compared to the work in [13], we have added suitable extensions to deal with explicit time slots for services and we use ILOG CPLEX to solve the associated integer programming problem. We again use $w_1 = w_2 = w_3 = \frac{1}{3}$ and set the overall cost constraint to $u_{\max}$ and the time limit to $t_{\mathrm{zero}} - 1$.

5.2.3 Adaptive Global Weighted Optimization

This strategy is similar to the previous, but it reserves new services in case of failures.

5.2.4 Flexible Strategy

This is our robust strategy as presented in the previous section. As learning and trust are not the focus of this paper, we assume that the agent has access to accurate trust information and a large strategy library (see [22] for details on how this is generated).

5.3 Hypotheses

In our experiments, we are interested in testing four hypotheses. The first two consider environments where providers fail maliciously without paying compensation, the next one considers cases where providers offer full refunds for failures, and the final hypothesis looks at environments where providers offer better services when reserved with varying advance notice periods (e.g., where there are discounts for early or late reservations).

- **Hypothesis 1**: In environments where the service performance does not depend on the time of reservation and where services fail maliciously without paying penalties, the *flexible* strategy results in a higher profit than any of the other examined strategies, averaged over all cases.
- **Hypothesis 2**: In the above environments, the *flexible* strategy successfully completes a higher proportion of workflows than any of the other examined strategies, averaged over all cases.
- **Hypothesis 3**: Hypotheses 1 and 2 also hold when services offer full refunds for failures.
- **Hypothesis 4**: Hypotheses 1 and 2 also hold in environments where the performance of services is dependent on the time of reservation.

In the following, we discuss the results of our experiments. Where appropriate, we test the above hypotheses by carrying out ANOVA, followed by pairwise t-tests. For reasons of space, we do not report individual p-values, which were all at the $p = 0.001$ level or better. We also provide 95 percent confidence intervals for all data.

5.4 Malicious Providers (Hypotheses 1 and 2)

During our first experiments, we evaluate the performance of the strategies in environments where service providers are increasingly unreliable. To this end, we vary an overall average defection probability $\bar{d}$ across several experiments and use this to generate the defection probability of offers.[9] We also assume that services always either succeed or defect. This case is challenging, because consumers do not get compensation for failures, but it is realistic in highly dynamic distributed systems, where some providers may act maliciously and never perform the service they were paid to do. Examples of such systems include peer-to-peer systems, where providers frequently leave the system and where the enforcement of contracts is difficult.

The results of our experiments are shown in Fig. 2, which plots the average failure probability against the average profit (as a proportion of $u_{\max}$) that each strategy gains.[10] To complement this, Fig. 3 shows the associated proportion of workflows that the strategy managed to complete with a positive reward. When providers never defect ($\bar{d} = 0$), all strategies perform well, achieving between 70-90 percent of the maximum reward, and there is no significant difference between either of the global optimization approaches and the flexible strategy. Intuitively, both global strategies are equivalent here, because there is no need to reattempt failed tasks, and they both perform well due to the certain information they have about the cost and duration of the

9. Again, we draw from $U_h(\bar{d})$, where $h = \min(0.2, h')$ and h' is the largest real number with $(1 - h') \cdot \bar{d} \geq 0 \wedge (1 + h') \cdot \bar{d} \leq 1$.

10. We average the profit over 750 runs for the flexible and the local approaches, while we average it over 250 runs for the global optimization approaches due to their more time-intensive nature.

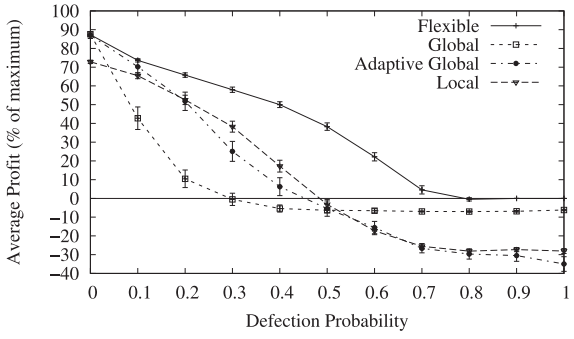

Fig. 2. Performance when providers increasingly defect.

complete workflow. The flexible strategy similarly performs well—although it does not reserve the complete workflow in advance, it makes accurate predictions at the start (with little uncertainty) and reserves services as it proceeds through the workflow. The local optimization approach performs worse, as it takes myopic decisions and therefore occasionally exceeds $t_{\max}$ or even t_{zero}.

As $\bar{d}$ increases, all strategies generally perform worse, because they have to pay for services that do not perform as promised. The nonadaptive global optimization strategy is most affected as $\bar{d}$ begins to rise, due to it only attempting one execution of the workflow before giving up. If it succeeds, it gains a relatively high reward, but if it fails, it loses its initial investment. At $\bar{d} = 0.3$ and beyond, the strategy no longer makes a profit, as it begins to fail most workflows and lose its investments.

In contrast to this, the adaptive optimization strategy performs considerably better than the nonadaptive one as the defection probability begins to rise, up to $\bar{d} = 0.4$. On this interval, failures occur occasionally and the adaptive consumer is generally able to reserve new offers to meet its deadline. However, at $\bar{d} = 0.5$, failures become too numerous (the consumer now fails to complete 69.0 percent of its workflows before t_{zero}) and the consumer begins to make an overall loss. As the defection probability rises further, this loss increases, eventually levelling off toward $\bar{d} = 1.0$. This considerable loss occurs because the consumer lacks the capability of predicting the overall cost it will incur by reattempting failed tasks and whether this investment is rational, given the defection probabilities of services. Rather, it will persist in retrying more services and making further investments, despite a high probability of failure (at $\bar{d} = 0.8$ and beyond, the consumer completes no workflows successfully).

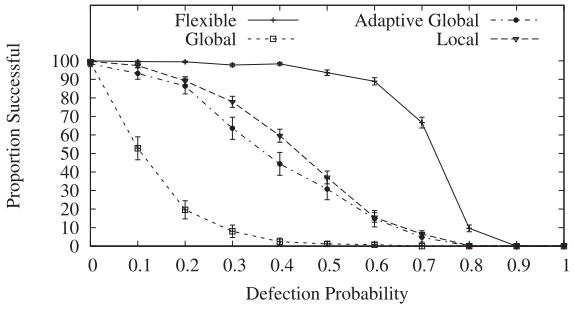

Fig. 3. Success probability when providers increasingly defect.

TABLE 7
Summary of Empirical Results

Strategy	Environment	Utility	Success %
flexible	malicious	725.83 ± 15.86	68.55 ± 0.51
global	malicious	173.80 ± 26.85	16.87 ± 1.39
adaptive global	malicious	183.60 ± 37.83	40.21 ± 1.85
local	malicious	212.30 ± 20.58	43.93 ± 1.07
flexible	refunds	1026.53 ± 14.23	81.96 ± 0.42
global	refunds	200.60 ± 26.07	16.74 ± 1.38
adaptive global	refunds	335.80 ± 34.30	41.57 ± 1.86
local	refunds	473.68 ± 17.13	43.30 ± 1.07
flexible	discounts	1143.61 ± 12.12	95.83 ± 0.22
global	discounts	109.69 ± 17.78	13.11 ± 0.90
adaptive global	discounts	428.40 ± 24.70	53.20 ± 1.36
local	discounts	516.37 ± 15.17	59.78 ± 0.77

Next, the average profit of the local strategy initially drops less quickly than the global strategies. This occurs because it is less affected by a small number of failures than the global approach, which may need to reserve new offers for its entire workflow upon a single failure. In some environments, when the defection probability is $\bar{d} = 0.2$ and $\bar{d} = 0.3$, it even outperforms the adaptive global approach for that reason. Beyond that, it drops more quickly and follows a broadly similar trend to the adaptive global strategy, as it also invests heavily in services without completing the workflow.

Finally, we consider the performance of the flexible strategy. At low defection probabilities, it achieves a similar performance to the global approaches. However, at $\bar{d} = 0.2$, it begins to clearly dominate all other strategies. Unlike the other strategies, it reasons explicitly about failures and their impact on the workflow cost and execution time, and so at these higher failure probabilities, the flexible strategy is able to deal proactively with failures, for example, by reserving them redundantly or by favoring more reliable providers. In more detail, this means that the flexible approach is able to achieve an almost 200 percent improvement over the best-performing nonflexible strategy at $\bar{d} = 0.4$ and it still makes a positive profit at $\bar{d} = 0.5$, $\bar{d} = 0.6$, and $\bar{d} = 0.7$, when all other strategies make a loss (in fact, the flexible strategy here successfully completes over 98, 93, 88, and 66 percent of its workflows before t_{zero}, respectively). At $\bar{d} = 0.8$, we notice that the flexible strategy makes a small net loss of -9.97 ± 18.25. However, this is not significant in this case.

Averaged over all values for $\bar{d}$ we tested, the flexible approach achieves a higher profit and completes more workflows than all other approaches, supporting Hypothesis 1 and 2 (see Table 7 for full results).

5.5 Failures with Refunds (Hypothesis 3)

Next, we are interested in environments where providers are not malicious, but offer full refunds to the consumer in case of failure. Hence, the setup is similar to the previous section, but we now assume that when providers fail, they immediately refund both the reservation and the execution cost of the service. This is a more realistic scenario when services are offered by reputable companies, when some central entity monitors the system or when contracts are easily enforceable. Examples may include web services or scientific Grids.

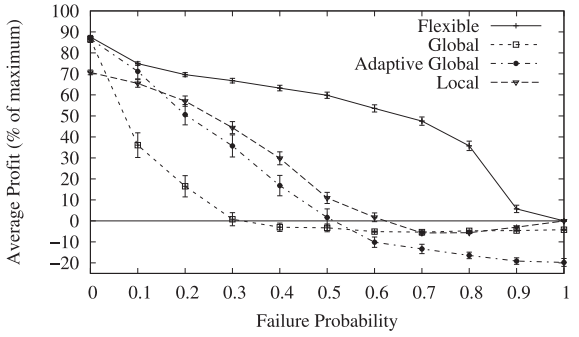

Fig. 4. Performance when providers give refunds.

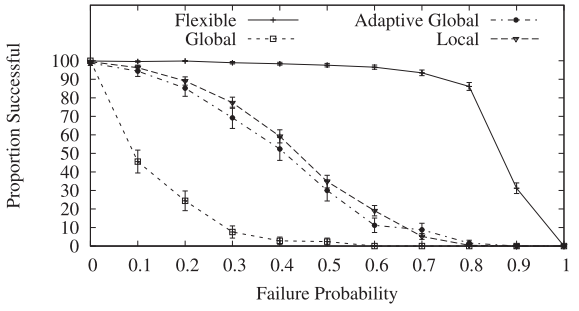

Fig. 5. Success probability when providers give refunds.

The results are shown in Figs. 4 and 5, and clearly highlight mostly the same trends as in the previous experiments for the nonflexible strategies (all achieve slightly higher profits and tolerate higher failure probabilities). The local strategy now performs better than before as it will pay at most once for each task in the workflow, and therefore even achieves a small positive average profit when the failure probability is $\bar{f} = 0.6$.

The flexible strategy performs significantly better in this environment, achieving a high positive profit even at failure probabilities of up to $\bar{f} = 0.9$. More specifically, at $\bar{f} = 0.6$, our strategy achieves an average profit of 1,071.22, with 96.5 percent of workflows completed before t_{zero}, compared to the best nonflexible profit of only 34.34 with 19.1 percent of workflows successful (an approximately 35-fold improvement in utility). At $\bar{f} = 0.8$, the flexible approach still completes 86.1 percent of workflows successfully, while the most successful alternative completes 1.6 percent. This good performance is due to the considerably lower cost of reserving services redundantly, as now the consumer pays for only those services that succeed. Even at $\bar{f} = 0.9$, the flexible approach still achieves a positive profit of 114.35 and completes 31.2 percent of workflows. These results support Hypothesis 3.

5.6 Different Market Conditions (Hypothesis 4)

Next, we test the performance of the strategies in environments where either advance or on-demand reservation is preferred and given a discount in execution cost and a higher reliability. Such conditions might occur, respectively, when providers prefer to be given early notice by consumers, so that they can plan their resource availability in advance, or when they find their resources underutilized and therefore offer discounted services at the last minute. To express this preference, we vary a discount factor, d, from -1 to 1. When negative, this indicates a preference for advance reservation and when positive, on demand reservation is preferred. In more detail, we use it during offer generation to adjust the distribution means for the execution cost and failure probability by a proportion given by $|d|$. We consider all offers generated for the current time step, t, as reserved on demand, and any offers generated for $t + 40$ and beyond as reserved in advance. Between these two, we vary the discount factor linearly. For example, when $d = -0.6$, $\bar{f} = 0.5$ and we generate an offer for t_{i+30}, then the corresponding mean failure probability is $(1 - 3/4 \cdot 0.6) \cdot 0.5 = 0.275$. We use all other experimental parameters as in the previous sections, but keep $\bar{f}$ at 0.5, and now set $b = 0.5$ and $d = 5$, to ensure that discounted offers are available only at their respective time steps.

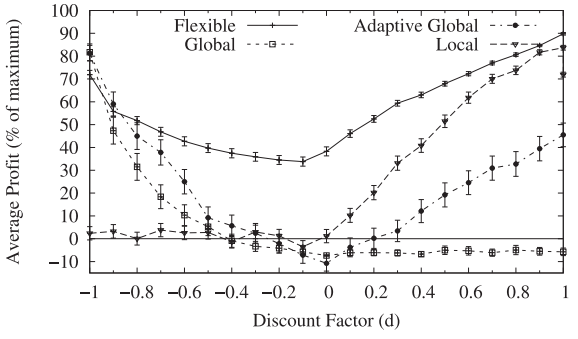

Fig. 6. Performance when advance reservations are preferred (negative d) and when on-demand is preferred (positive d).

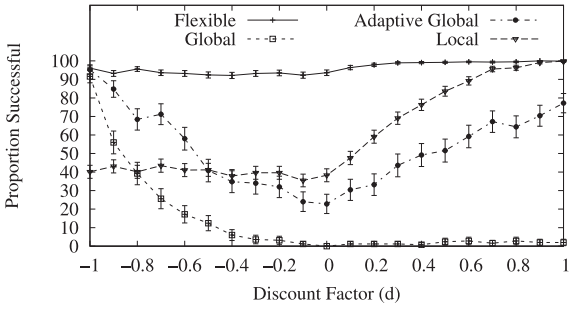

Fig. 7. Success probability when advance or reservations or on demand invocations are preferred.

The results for this setting are given in Figs. 6 and 7. Here, we note that the nonflexible strategies perform well only in extreme conditions—the global approaches excel when advance reservations are preferred, while the local strategy performs well as d tends to 1. When neither advance nor on-demand reservations are strongly preferred, none of the nonflexible strategies does well, because most services in the market are unreliable. In fact, at $d = -0.1$, these strategies all make a net loss. In contrast to this, the flexible strategy achieves a high profit over all environments, and, in most cases, significantly outperforms all others. This is because the flexible strategy adjusts its reservation strategies to the environment—at $d = -1$, it reserves services, on average, 43.05 ± 0.72 time steps in advance, at $d = 0$, this drops to 14.71 ± 0.36 and at $d = 1$, it reserves only 3.57 ± 0.12 time steps ahead. However, we also note that the flexible strategy is now outperformed at $d = -1$ (at $d = -0.9$, there is no significant difference). In this case, it suffers from not reserving all offers in advance

(and thereby producing a tight-fitting but reliable schedule). Instead, the strategy continues to reserve only parts of the workflow (although now reserving further ahead) and hence sometimes exceeds $t_{\max}$. Nevertheless, when averaging over all values for d, the flexible strategy outperforms the others, thus supporting Hypothesis 4.

To summarize our empirical evaluation, Table 7 shows the average utility each strategy gained in the various environments discussed in this chapter.

6 Conclusions and Future Work

In this paper, we have developed a novel workflow execution strategy that extends the state of the art in several ways. First, our approach reserves only part of a workflow at a time and adapts its decisions at runtime, making the strategy more robust and dependable in uncertain environments. Second, we consider highly dynamic systems, where service availability changes over time and where the consumer typically does not know what services will be available in the future. We address this by learning high-level strategies to predict the typical performance of certain workflow tasks. Finally, our approach proactively considers contingencies to deal with service failures and conflicts, but does so by exploring only a limited number of outcomes and by considering each workflow task in isolation, thus increasing the efficiency of our approach.

Our proposed strategy is highly relevant in a wide range of application areas, and we have adopted an abstract system model that can be easily applied to web services, Grid services, and peer-to-peer systems. We envisage that our techniques will fit naturally on top of existing workflow execution engines, and in particular, those that are already able to switch service providers dynamically at runtime, such as the self-healing WS-BPEL workflows described in [24]. Furthermore, although we use the contract-net protocol, our approach is readily applicable to other mechanisms and existing web service protocols, as described in Section 3.2. Finally, the high-level decisions discussed in Section 4.2 could refer to strategies for participating in auctions, to carry out negotiations or simply for selecting services published on a registry, and are therefore highly versatile.

There are several ways in which we plan to extend our work. First, we want to improve our utility estimation technique, which sometimes overestimates the expected utility of a workflow. Second, we will extend our contract model to cover more complex usage models, such as subscriptions for repeated service invocations. Finally, we will consider workflows with conditional branches.

Acknowledgments

This is an extended version of a previous conference paper [25]. The work was funded by BAE Systems and the EPSRC.

References

[1] H. Ludwig, T. Nakata, O. Wäldrich, P. Wieder, and W. Ziegler, "Reliable Orchestration of Resources Using WS-Agreement," *Proc. Second Int'l Conf. High Performance Computing and Comm. (HPCC '06)*, pp. 753-762, 2006.

[2] K. Czajkowski, I. Foster, and C. Kesselman, "Agreement-Based Resource Management," *Proc. IEEE*, vol. 93, no. 3, pp. 631-643, Mar. 2005.

[3] N.R. Jennings, "An Agent-Based Approach for Building Complex Software Systems," *Comm. ACM*, vol. 44, no. 4, pp. 35-41, 2001,

[4] S. Stein, N.R. Jennings, and T.R. Payne, "Provisioning Heterogeneous and Unreliable Providers for Service Workflows," *Proc. Sixth Int'l Joint Conf. Autonomous Agents and Multiagent Systems (AAMAS '07)*, pp. 1452-1458, 2007.

[5] M. Keidl, S. Seltzsam, and A. Kemper, "Reliable Web Service Execution and Deployment in Dynamic Environments," *Proc. Fourth Int'l Workshop Technologies for E-Services (TES '03)*, pp. 104-118, 2003.

[6] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Comm. ACM*, vol. 51, no. 1, pp. 107-113, 2008.

[7] H. Kreger, "Fulfilling the Web Services Promise," *Comm. ACM*, vol. 46, no. 6, pp. 29-34, 2003.

[8] F. Curbera, R. Khalaf, N. Mukhi, S. Tai, and S. Weerawarana, "The Next Step in Web Services," *Comm. ACM*, vol. 46, no. 10, pp. 29-34, 2003.

[9] F. Montagut, R. Molva, and S. Tecumseh Golega, "The Pervasive Workflow: A Decentralized Workflow System Supporting Long-Running Transactions," *IEEE Trans. Systems, Man and Cybernetics, Part C: Applications and Reviews*, vol. 38, no. 3, pp. 319-333, May 2008.

[10] R. Porter, A. Ronen, Y. Shoham, and M. Tennenholtz, "Fault Tolerant Mechanism Design," *Artificial Intelligence*, vol. 172, no. 15, pp. 1783-1799, 2008.

[11] E. Damiani, P. Ceravolo, S. Cimato, and G. Gianini, "Obfuscation for the Common Good," *Proc. Conf. Security in Network Architectures and Information Systems (SAR-SSI '08)*, pp. 15-35, 2008.

[12] H. Raiffa, *Decision Analysis: Introductory Lectures on Choices under Uncertainty*. McGraw-Hill, 1968.

[13] L. Zeng, B. Benatallah, A.H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-Aware Middleware for Web Services Composition," *IEEE Trans. Software Eng.*, vol. 30, no. 5, pp. 311-327, May 2004.

[14] E. Sirin, B. Parsia, and J. Hendler, "Template-Based Composition of Semantic Web Services," *Proc. AAAI Fall Symp. Agents and the Semantic Web*, pp. 85-92, 2005.

[15] R. Aggarwal, K. Verma, J. Miller, and W. Milnor, "Constraint Driven Web Service Composition in METEOR-S," *Proc. IEEE Int'l Conf. Services Computing (SCC '04)*, pp. 23-30, 2004.

[16] J. Collins, C. Bilot, M. Gini, and B. Mobasher, "Decision Processes in Agent-Based Automated Contracting," *IEEE Internet Computing*, vol. 5, no. 2, pp. 61-72, Mar./Apr. 2001.

[17] R.G. Smith, "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver," *IEEE Trans. Computers*, vol. 29, no. 12, pp. 1104-1113, Dec. 1980.

[18] S.D. Ramchurn, D. Huynh, and N.R. Jennings, "Trust in Multiagent Systems," *The Knowledge Eng. Review*, vol. 19, no. 1, pp. 1-25, 2004.

[19] W.T.L. Teacy, J. Patel, N.R. Jennings, and M. Luck, "TRAVOS: Trust and Reputation in The Context of Inaccurate Information Sources," *Autonomous Agents and Multi-Agent Systems*, vol. 12, no. 2, pp. 183-198, 2006.

[20] E.M. Maximilien and M.P. Singh, "Agent-Based Trust Model Involving Multiple Qualities," *Proc. Fourth Int'l Joint Conf. Autonomous Agents and Multiagent Systems (AAMAS '05)*, pp. 519-526, 2005.

[21] J.N. Hagstrom, "Computational Complexity of PERT Problems," *Networks*, vol. 18, pp. 139-147, 1988.

[22] S. Stein, "Flexible Service Provisioning in Multi-Agent Systems," PhD dissertation, Univ. of Southampton, 2008.

[23] S. Kirkpatrick, J.C.D. Gelatt, and M.P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, no. 4598, pp. 671-680, 1983.

[24] T. Friese, J.P. Müller, and B. Freisleben, "Self-Healing Execution of Business Processes Based on a Peer-to-Peer Service Architecture," *Proc. 18th Int'l Conf. Architecture of Computing Systems (ARCS '05)*, pp. 108-123, 2005.

[25] S. Stein, T.R. Payne, and N.R. Jennings, "Flexible Service Provisioning with Advance Agreements," *Proc. Seventh Int'l Conf. Autonomous Agents and Multi-Agent Systems (AAMAS '08)*, pp. 249-256, 2008.

Sebastian Stein received the PhD degree in computer science from the University of Southampton, United Kingdom, where he now works as a research fellow in the Intelligence, Agents, Multimedia Group. His research interests include the application of artificial intelligence, decision theory, and game theory to service-oriented and multiagent systems.

Terry R. Payne is a lecturer in the Agent ART Group at the University of Liverpool, United Kingdom, where he carries out research in agent-based computing and knowledge-based systems and services. His specific interest is in the use of ontological knowledge for the support of service discovery and provision.

Nicholas R. Jennings is a professor of computer science in the top-rated School of Electronics and Computer Science at Southampton University, a chief scientific advisor to the United Kingdom Government, an associate dean (Research & Enterprise) for the Faculty of Engineering, Science and Maths, Head of the Intelligence, Agents, Multimedia Group (which consists of some 120 research staff and postgraduate students), and the chief scientific officer for Aroxo. He is a fellow of the IEEE and the IEEE Computer Society.