

performs the state of the art in most environments when services are unreliable (and achieves good results even when services are deterministic). During these experiments, we show that these trends hold over a range of environments, including where providers fail maliciously (i.e., do not pay compensation). In certain environments, we show that our strategy achieves a 27-fold improvement in average utility compared to current approaches and successfully completes up to 50 times as many workflows within its deadline.

The remainder of this paper is organised as follows. In Section 2, we begin by formalising our model of a service-oriented system. Then, in Section 3, we outline our flexible provisioning strategy, and in Section 4, we evaluate it. Finally, in Section 5, we conclude.

2. SERVICE-ORIENTED SYSTEM MODEL

To allow us to devise a generic and principled approach to service provisioning, 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 [2, 9, 8]. In the following, we first describe the types of workflows we consider (Section 2.1), then detail the provisioning process (Section 2.2) and finally discuss how services are invoked (Section 2.3).

2.1 Workflows

In this paper, we are interested in the execution of a single workflow, which is selected by the agent at time $t = 0$ (we assume that time passes in discrete integer time steps). Formally, we define a *workflow* as a tuple, $W = (T, E, \tau, u)$, where $T = \{t_1, t_2, t_3, \dots, t_n\}$ 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 $\tau : T \rightarrow S$ maps each task in T to an abstract *service type*, where $S = \{s_1, s_2, s_3, \dots\}$ is the set of all service types. Finally, $u : \mathbb{Z}_0^+ \rightarrow \mathbb{R}$ describes the *reward* of completing the workflow at a given point in time t . It is defined by a *maximum reward*, u_{\max} , a *deadline*, t_d , and a cumulative *penalty* for late completion, δ :

$$u(t) = \begin{cases} u_{\max} & \text{if } t \leq t_d \\ u_{\max} - \delta(t - t_d) & \text{if } t > t_d \wedge t < t_d + u_{\max}/\delta \\ 0 & \text{if } t \geq t_d + u_{\max}/\delta, \end{cases} \quad (1)$$

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 agent’s reward (or 0 if none has been obtained before time step $t_{\text{zero}} = \lceil t_d + u_{\max}/\delta \rceil$) and the total cost it has incurred by provisioning and invoking services.

2.2 Service Provisioning

To complete its workflow, the service consumer needs to discover and enter contracts with (i.e., *provision*) suitable service providers 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 multi-agent systems [6]. In more detail, the consumer may send a *call for proposals*, $\varphi : S \times \mathbb{Z}_0^+$, to the service market to request a particular service type at a given time step. For example, $\varphi = (s_1, 2)$ indicates that the consumer requires a service of type s_1 to start at time step 2.

In response to each call, the market returns a set of *offers*, C_φ . These are potential contracts that the service providers participating in the system are willing to offer to the consumer. Each offer $c \in C_\varphi$ contains a number of terms, as given in Table 1.

This 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

Term	Description
$s(c) : S$	The <i>service type</i> offered (equal to the requested type).
$t(c) : \mathbb{Z}_0^+$	The <i>starting time</i> at which the service can be invoked (equal to the requested time).
$c_r(c) : \mathbb{R}$	The <i>reservation cost</i> , which must be paid immediately by the consumer when entering the contract.
$c_e(c) : \mathbb{R}$	The <i>execution cost</i> , which is the remaining cost (after the reservation cost) that the consumer must pay when invoking the service.
$d(c) : \mathbb{Z}^+$	The <i>duration</i> , i.e., the number of time steps it will take for the service to complete.
$\delta_f(c) : \mathbb{R}$	The <i>failure penalty</i> , which is paid to the consumer when the service fails to complete successfully within the agreed duration.

Table 1: Service contract terms.

Prob.	Description
$P_f(c)$	The <i>failure probability</i> is the probability that the service will not complete successfully within the agreed duration and pay the failure penalty $\delta_f(c)$.
$P_d(c)$	The <i>defection probability</i> is the probability that the provider will fail to provide the service and also fail to pay the agreed penalty $\delta_f(c)$ (e.g., if the provider crashes, leaves the market or maliciously disregards the market rules).
$P_s(c)$	The <i>success probability</i> is the probability that the provider will successfully complete the requested service within the agreed duration.

Table 2: Performance information (outcome probabilities).

types and times are always identical. Furthermore, we assume that the consumer has some information about the probabilities of the possible outcomes of each offer, as shown in Table 2. In practice, this may be obtained through a trust and reputation mechanism and may be based on the past behaviour of individual providers (but we are agnostic about its origin in this paper). Together, these probabilities describe all possible, mutually exclusive outcomes of an offer, such that $P_f(c) + P_d(c) + P_s(c) = 1$. To retain a simple, generic model, and in line with related work mentioned earlier, we assume that the outcomes of any two distinct offers are independent.

During the same time step as receiving offers from the market, the consumer may provision any number (or none) of these offers for the tasks of its workflow. To do this, it sends a single acknowledgement to the market, $a : C \rightarrow T$, that maps offers to the corresponding tasks of the workflow, where C is the set of all offers received during the time step. At this point, the consumer must pay the reservation costs of all provisioned offers, and any offers not in the domain of a are implicitly assumed to be rejected. We also assume that the consumer may provision several offers for a single task (e.g., to increase its overall success probability).

2.3 Service Invocation

At the end of any time step, the consumer may invoke its provisioned offers, provided that all relevant precedence constraints given by E have been satisfied and that the agreed starting time matches the current time. The outcome of the invocation is one of the outcomes listed in Table 2, but we assume that it is not known until the beginning of the time step at which the service is scheduled to end (e.g., if invoking offer c with $t(c) = 15$ and $d(c) = 10$, the consumer will only be notified of the outcome at the beginning of time step $t = 25$).

3. FLEXIBLE PROVISIONING

In this section, we detail our novel provisioning strategy. We begin in Section 3.1 with a brief overview of our strategy, which is then elaborated upon in Sections 3.2–3.4.

3.1 Strategy Overview

As outlined in Section 1, we are interested in building a rational agent that acts to maximise 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 maximises the expected difference between the reward and cost of following it. Formally, given a workflow W and some probabilistic beliefs about the behaviour of the market, we are interested in finding strategy S^* :

$$S^* = \operatorname{argmax}_S E(R(S) - C(S)) \quad (2)$$

where $R(S)$ and $C(S)$ are random variables describing the final reward and cost, respectively, of using strategy S to execute the workflow, and $E(X)$ denotes the expected value of X .

However, finding S^* is intractable for the same reasons as described in [8]. 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 behaviour is deterministic. Second, calculating 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 $E(R(S))$. 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 maximises 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 provisioning decisions and contingent strategies we consider for each task of a workflow (Section 3.2). Then, we describe how these are used to define an overall strategy S for a complete workflow (Section 3.3). Finally, we outline our overall provisioning algorithm that finds a good strategy and continuously adapts it during workflow execution (Section 3.4).

3.2 High-Level Task Strategies

We believe that it is generally inefficient for a consumer agent to provision offers 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 provisioned 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 *provisioning 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 formalise these strategies and discuss how they can be extended to express task strategies with contingencies.

3.2.1 Task Library

High-level provisioning strategies are available to the consumer as a library, $S \rightarrow \mathcal{P}(\Omega)$, that maps service types to strategies. Each strategy $\omega \in \Omega$ is described by a number of parameters, as shown in Table 3. The first two prescribe how the consumer will formulate its call for proposals, e.g., if $t_a(\omega) = 100$ and $t_i(\omega) = 3$, it will request services 100 time steps in advance and for 3 consecutive time steps. The latter two describe how it will select from the returned offers. Here, we consider four simple se-

Parameter	Description
$t_a(\omega) : \mathbb{Z}_0^+$	Number of time steps to provision offers in advance.
$t_i(\omega) : \mathbb{Z}^+$	Time interval to request services for.
$n(\omega) : \mathbb{Z}^+$	Maximum number of offers to provision.
$s(\omega)$	Strategy for choosing offers to provision when more than $n(\omega)$ offers are available.

Table 3: Task strategy parameters.

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

Table 4: Strategy performance statistics.

lection strategies for parameter $s(\omega)$: $\{\text{cost, unreliability, end_time, balanced}\}$. The first three indicate that the consumer will always choose the offers with, respectively, the lowest expected cost ($c_r(c) + c_e(c) - P_f(c)\delta_f(c)$), the lowest probability of not succeeding ($1 - P_s(c)$), or the lowest end time ($t(c) + d(c)$). The selection strategy *balanced* will pick the offers that minimise a sum of these parameters, each normalised to the interval $[0, 1]$, so that 0 corresponds to the offer with the lowest parameter and 1 to the highest. We also assume that there is a strategy not to do anything, ω_{null} (i.e., the agent will stop executing the task).

Furthermore, we assume that the consumer has some performance information about each of the strategies, which it previously learnt by observing the response of the market to various calls for proposals. Specifically, we assume that it has repeatedly submitted calls of proposals corresponding to its known strategies to the market. Then, using simple calculations¹, 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 provisioning decisions (based on its trust information and without necessarily provisioning and invoking any offers). This information is summarised in Table 4. Here, ϵ denotes the overall outcome of the strategy, with $\epsilon \in \{\text{success, unavailable, failed}\}$ (referring, respectively, to the events where at least one provisioned offer is successful, where no suitable offers were found and where some offers were provisioned but failed).

3.2.2 Planning for Contingencies

The simple provisioning 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, we decided to include several contingent strategies that the consumer will use if its primary strategy was not successful.

These are shown in Figure 1. Here, s_p is the main strategy the consumer will use to provision the task, but it also has several strategies to fall back on if s_p was not successful:

- s_l is used to re-provision 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 provision new offers using s_l .

¹Full details are omitted for space reasons, but can be found in [7].

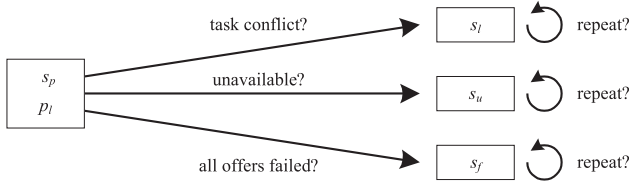


Figure 1: Task contingencies.

- s_u is used when the initial strategy did not result in any provisioned offers at all (e.g., if there were none available on the market). 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 non-repeated ω (see [7]).

Now, when the service consumer plans to provision 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 provisioning, we attach a *late probability*, p_l , to each task. This is the largest acceptable probability when provisioning task t_i that one of the predecessors of t_i will still not have been completed successfully by the time step t_i was provisioned for. More formally, the consumer will provision task t_i with primary strategy s_p at the earliest possible time step t where $p_p(t_i, t + t_a(s_p)) \leq p_l$, 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 provisioning relative to other tasks in the workflow.

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, p_l , it is straight-forward to generalise the statistics from Table 4 to the whole task by considering the possible outcomes of Figure 1. Furthermore, depending on p_l , we predict when provisioning will start for each task, estimate the expected waiting time between the start of the task and the completion time of its predecessors and produce a more accurate estimate of the late probability that depends on the expected duration and variance of the preceding tasks². Combining these statistics, we can estimate the expected completion time and its variance for each task. Finally, once a task has been provisioned

²For details, see [7]. Briefly, we use a normal approximation along the critical path [4] that leads to the task in question. Working backwards from the task, we then identify a predecessor during which provisioning will take place and use the probability density function of a normal distribution to estimate the time of provisioning (relative to that predecessor), the actual late probability and an expected waiting time.

(i.e., s_p has been carried out), it is easy to refine the expected outcomes of a task, considering the actual offers.

3.3 Workflow Provisioning Strategy

Given the high-level task strategies described in the previous section, we now outline how they are used to express the overall workflow strategy, \mathcal{S} , that our agent seeks to maximise in Equation 2. In particular, we describe the strategy as a tuple:

$$\mathcal{S} = (\alpha, \beta, \gamma, d_\beta, d_\gamma, E') \quad (3)$$

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 negotiated, and γ contains the tasks for which no offers are currently provisioned. 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 the discussion of the previous section, $d_\beta(t_i)$ of a provisioned task $t_i \in \beta$ is:

$$d_\beta(t_i) = (c_{pi}, s_{li}, s_{ui}, s_{fi}) \quad (4)$$

where c_{pi} is the set of offers already provisioned for t_i , while the other objects refer to the contingent strategies. Similarly, $d_\gamma(t_j)$ of a task $t_j \in \gamma$ is:

$$d_\gamma(t_j) = (s_{pj}, s_{lj}, s_{uj}, s_{fj}, p_{lj}) \quad (5)$$

where s_{pj} is the primary provisioning decision and p_{lj} is the late probability. Finally, $E' : T \leftrightarrow T$, is a set of temporary edges, so that $E \cup E'$ is still a strict partial order over T . This allows the agent to express additional precedence constraints between tasks where this increases the overall expected utility — for example, to indicate that a cheap and unreliable task should be completed before attempting a particularly expensive one.

Using the performance statistics outlined in the previous section, we can estimate the overall expected cost and reward of a strategy \mathcal{S} (see [7]), which we use to calculate the objective function that is maximised in Equation 2. In the next section, we describe how this information is used in the context of our overall provisioning strategy.

3.4 Adaptive Optimisation Algorithm

The overall behaviour of our flexible service consumer is shown in Algorithm 1. At time $t = 0$, the consumer first selects an appropriate workflow for its current objective (line 2). In practice, this may come from a plan library, it may be provided by a user or even synthesised at run-time using a planner. The agent then adopts an initial strategy for executing the workflow, $\mathcal{S} = (\emptyset, \emptyset, T, \emptyset, d_\gamma, \emptyset)$, where d_γ is some initial allocation of task provisioning strategies that the agent will improve later³ (line 3).

This is followed by the main loop of the algorithm (lines 5–19), which is repeated every time step. First, the consumer receives all service outcomes (line 6) and updates its current strategy accordingly (line 7). During this update, the consumer removes any failed or missed offers and moves successful tasks from β to α . If necessary, it also updates its high-level task strategies, for example by adopting the strategy s_{fi} as the primary strategy of task t_i if the last remaining provisioned offer for that task has just failed (in this case, the task is also moved to γ).

³In our work, we start with a simple allocation $d_\gamma(t_j) = (\omega, \omega, \omega, \omega, 0.01)$ for all tasks, with $t_a(\omega) = 0$, $t_i(\omega) = 10$, $n(\omega) = 1$ and $s(\omega) = \text{unreliability}$. This already constitutes a feasible strategy in most environments and leads to a quicker convergence during the initial optimisation stage than a completely random initial strategy.

Algorithm 1 Summary of flexible provisioning strategy

```
1:  $t \leftarrow 0$ 
2:  $W \leftarrow \text{SELECTWORKFLOW}$ 
3:  $S \leftarrow \text{CREATEINITIAL}(W)$ 
4:  $\text{abandoned} \leftarrow \text{false}$ 
5: repeat
6:    $\mathcal{O} \leftarrow \text{RECEIVEINVOCATIONOUTCOMES}$ 
7:    $S \leftarrow \text{UPDATEWITHOUTCOMES}(S, \mathcal{O})$ 
8:   repeat
9:      $S \leftarrow \text{OPTIMISE}(S)$ 
10:     $S \leftarrow \text{REALISESTRATEGIES}(S)$ 
11:   until  $S$  was not altered in line 10
12:   if  $\text{PREDICTUTILITY}(S) > 0$  then
13:      $\text{PROVISIONSERVICES}(S)$ 
14:      $\text{INVOKEDUESERVICES}(S)$ 
15:   else
16:      $\text{abandoned} \leftarrow \text{true}$ 
17:   end if
18:    $t \leftarrow t + 1$ 
19: until  $\text{abandoned} = \text{true}$  or workflow completed
```

Next, the algorithm attempts to improve its current strategy (line 9). We use a local search for this, as such an approach is well-suited for the high-dimensional decision space and the non-linear objective function we consider. It also deals naturally with updated information at run-time and has any-time properties that mean it can be adapted for environments where limited time is available at each time step. Specifically, we adopt simulated annealing, which is less prone to suffer from local maxima than deterministic local search techniques [3]. Briefly, our search generates new neighbours of a strategy S by: changing a single provisioning strategy of any task; changing late probabilities; and altering the temporary edges in E' . It also attempts to add or remove offers from an already provisioned task, without performing the associated provisioning.

When the optimisation procedure concludes, the consumer next identifies each task $t_i \in \gamma$ that is ready to be provisioned (as indicated by its late probability p_{t_i} , or if all its predecessors have been completed). For these tasks, the consumer simply follows the associated strategy s_{p_i} and updates its strategy with new offers that are currently on the market (line 10). As before, the consumer does not perform the actual provisioning yet, as it may still improve its initial selection. To this end, if any offers were added, the consumer again optimises the workflow — on one hand, this allows it to refine the selected offers, and on the other hand, it can adapt the remainder of the workflow to the newly selected offers. This continues until no more new tasks are provisioned.

After this, the consumer checks if it still expects to receive a positive utility from following the workflow (line 12). If so, it finalises its provisioning decisions by agreeing to the offers selected during the time step, and it invokes any executable tasks that are due at the current time step. During this, the consumer also removes from its current strategy all reservation and invocation costs it has just incurred (this is important because the consumer has now paid for parts of the workflow, effectively raising its overall value).

This overall behaviour continues until the consumer either does not expect to gain any utility from its current strategy or if the workflow is completed. To test the performance of this flexible strategy, we conduct a detailed empirical evaluation in the next section.

4. EMPIRICAL EVALUATION

As our methodology is a heuristic technique and due to the difficulty of finding an analytical solution (as described in Section 3.1) 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 favours certain provisioning approaches (e.g., where early provisioning is rewarded by more reliable services). In the following, we first describe how we simulate the market (Section 4.1), then we detail the strategies we test (Section 4.2) and finally describe our results (Section 4.3).

4.1 Market Setup

In our experiments, we assume that there are five different types of services ($S = \{s_1, s_2, s_3, s_4, s_5\}$). To simulate the market, we keep a list of currently available offers associated with each time step, from the current step \hat{t}_i to \hat{t}_{i+250} (hence, the consumer may provision 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 ($\hat{t}_i, \hat{t}_{i+1}, \dots, \hat{t}_{i+250}$), we generate offers using the distributions in each row of Table 5. 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 row⁴. 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 distributions⁵. 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 offers if the number exceeds the current supply).

Row	Type	Reserv. Cost	Exec. Cost	Time	Birth Rate	Death Rate
1	s_1	$\mathcal{U}_h(25)$	$\mathcal{U}_h(25)$	$\mathcal{U}_h(5)$	$r_b/2$	$r_d/2$
2	s_1	$\mathcal{U}_h(5)$	$\mathcal{U}_h(5)$	$\mathcal{U}_h(40)$	$r_b/2$	$r_d/2$
3	s_2	$\mathcal{U}_h(1)$	$\mathcal{U}_h(5)$	$\mathcal{U}_h(50)$	r_b	r_d
4	s_3	$\mathcal{U}_h(10)$	$\mathcal{U}_h(10)$	$\mathcal{U}_h(35)$	r_b	r_d
5	s_4	$\mathcal{U}_h(50)$	$\mathcal{U}_h(1)$	$\mathcal{U}_h(25)$	r_b	r_d
6	s_5	$\mathcal{U}_h(1)$	$\mathcal{U}_h(50)$	$\mathcal{U}_h(25)$	r_b	r_d

Table 5: Service type parameters.

In our simulations, a consumer is rewarded a maximum utility of $u_{\max} = 2000$ for completing a workflow, with penalty $\delta = 40$ and deadline $t_d = 200$. Each workflow consists of 8 tasks (with types chosen randomly from S) 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 two strategies that rely on integer programming techniques were unable to deal with larger cases. However, 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 in this paper.

⁴This is a common distribution for modelling random arrival events. We use $r_b = r_d = 0.005$, unless noted otherwise.

⁵We 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.

4.2 Strategies

We evaluate the performance of four strategies: the first three are based closely on the work presented in [9], but, more generally, represent common provisioning approaches that are widely used in the literature, such as [1]. The fourth is the flexible provisioning strategy proposed in this paper. We briefly describe each below.

4.2.1 Local Weighted Optimisation

This strategy provisions services completely on demand (i.e., when the respective task becomes available). During provisioning, 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 provisions the offer c^* that maximises a weighted sum:

$$c^* = \underset{c}{\operatorname{argmax}} \sum_{i=1}^3 w_i \cdot Q_i(c) \quad (6)$$

$$Q_i(c) = \begin{cases} 0 & \text{if } q_{\max,i} = q_{\min,i} \\ \frac{q_{\max,i} - q_i(c)}{q_{\max,i} - q_{\min,i}} & \text{otherwise} \end{cases} \quad (7)$$

where $q_1(c) = c_e(c) + c_r(c)$ is the combined total cost of the offer, $q_2(c) = 1 - P_s(c)$ is the probability that the task will not succeed and $q_3(c) = t(c) + d(c)$ 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 to re-provision any failed offers.

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 weight distributions).

4.2.2 Global Weighted Optimisation

This is perhaps the most widely adopted approach for provisioning services in the literature [1, 9]. Here, the agent observes the market once, then selects and provisions one offer for each task, so that a weighted sum similar to Equation 6 is maximised. 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 [9], 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 $t_{\text{zero}} - 1$.

4.2.3 Adaptive Global Weighted Optimisation

This strategy is similar to the previous, but it re-provisions services when they fail.

4.2.4 Flexible Provisioning

This is our flexible provisioning approach 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 (i.e., it knows the outcome probabilities of a given offer). We also build a task strategy library by taking 2000 independent observations of the market over time and recording the predicted outcomes of each of a set of possible strategies, which we generate by considering the combinations of the advance times $t_a(\omega) \in \{0, 10, 20, \dots, 250\}$, the provisioning intervals $t_i(\omega) \in \{1, 10, 20, \dots, 100\}$, the number of parallel providers $n(\omega) \in \{1, 2, 3, \dots, 10\}$ and all selection strategies. Considering that each strategy may be repeated, this results in 22880 possible high-level strategies for each service type.

Due to the time required to build the library, we do this once for every environment in this section and then re-use the same library when repeating our experiments (we have verified that there is no significant difference in our results when using a different library).

4.3 Results

In the following, we discuss the results of our experiments. Where appropriate, we have carried out ANOVA followed by pairwise t -tests to ascertain the statistical significance of the results (at the $p = 0.005$ level) and we give 95% confidence intervals for all data. We decided to test the four strategies over a number of different environments, where providers fail maliciously without paying compensation (Section 4.3.1), where providers give refunds when they fail (Section 4.3.2), and where providers offer better services when provisioned with varying advance notice periods (Section 4.3.3).

4.3.1 Environment 1: Malicious Providers

During our first set of experiments, we evaluated the performance of the four strategies in environments where service providers are increasingly unreliable. To this end, we varied an overall average defection probability \bar{d} across several experiments and used this to generate the defection probability of offers⁶. We also assume that services either succeed or defect (i.e., the failure penalty is irrelevant). This case is challenging for consumers, as they do not get compensation for failures, but it is realistic in highly dynamic distributed systems, where some providers may act maliciously and simply fail to perform the service they were paid to do. Examples include peer-to-peer systems, where providers may frequently leave the system and where it is difficult to enforce contracts.

The results of our experiments are shown in Figure 2, which plots the average defection probability of an environment against the average profit (as a proportion of u_{\max}) that each strategy gains⁷. When providers never defect ($\bar{d} = 0$), all strategies perform well, achieving between 70–90% of the maximum reward, and there is no significant difference between either of the global optimisation approaches and the flexible strategy. Intuitively, both global strategies are equivalent here, because there is no need to re-provision failed tasks, and they both perform well due to the certain information they have about the cost and duration of the complete workflow. The flexible strategy similarly performs well — although it does not provision the complete workflow in advance, it makes accurate predictions at the start (with little uncertainty) and provisions services as it proceeds through the workflow. The local optimisation approach performs worse than the other strategies, as it takes myopic decisions and therefore occasionally exceeds t_d or even t_{\max} .

As \bar{d} increases, all strategies generally perform worse, because they increasingly have to pay for services that do not perform as promised. The non-adaptive global optimisation 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. Hence, the performance trend closely follows the average success probability of a single execution, i.e., the probability that all eight workflow tasks succeed: $(1 - \bar{d})^8$.

In contrast to this, the adaptive optimisation strategy performs considerably better than the non-adaptive 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 re-

⁶Again, we draw from a distribution $U_h(\bar{d})$, where $h = \min(0.2, h')$ and h' is the largest real number with $(1 + h') \cdot \bar{d} \leq 1$.

⁷We average the profit over 750 runs for the flexible and the local approaches, while we average it over 250 runs for the global optimisation approaches due to their more time-intensive nature.

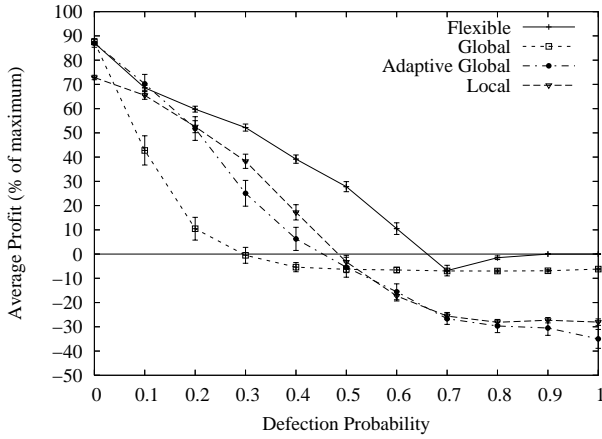


Figure 2: Performance of strategies in environments where providers increasingly defect.

provision the workflow to meet its deadline. However, at $\bar{d} = 0.5$, failures become too numerous (the consumer now fails to complete 69.0% of its workflows within t_{zero}) and the consumer begins to make an overall loss. As the defection probability rises further, this loss increases, eventually levelling off towards $\bar{d} = 1.0$. This considerable loss occurs because the consumer lacks the capability of predicting the overall cost it will incur by re-provisioning 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 failure probability.

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 a number of failures than the global approach, which may need to re-provision its workflow completely 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 the same reason. Beyond that, it follows a broadly similar trend to the adaptive global strategy, as it also invests heavily in services without ever completing the workflow.

Finally, we consider the performance of the flexible strategy. At low defection probabilities, it performs as well as the global approaches. However, at $\bar{d} = 0.2$, it begins to 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 higher failure probabilities, the flexible strategy is able to deal proactively with failures (e.g., by provisioning them redundantly or by favouring more reliable providers). In more detail, this means that the flexible approach is able to achieve an approximately 100% improvement over the best-performing non-flexible strategy at $\bar{d} = 0.4$ and it still makes a positive profit at $\bar{d} = 0.5$ and $\bar{d} = 0.6$ when all other strategies make a loss (in fact, the flexible strategy successfully completes over 96%, 93% and 84% of its workflows before t_{zero} in these environments, respectively).

However, at $\bar{d} = 0.7$, we notice that the flexible strategy makes a small net loss. This loss is not entirely surprising, due to our reliance on fast heuristic techniques to estimate the expected utility of a workflow. These techniques make some simplifying assumptions, which generally result in slightly optimistic utility estimates. For example, we use the critical path to estimate task durations, but this ignores tasks outside that path, which might become critical at run-time. Hence, the consumer will expect to achieve a small positive utility in these extremely challenging environments, but then discover at run-time that it needs to re-provision more often than expected in order to complete the workflow in time. Nevertheless, the

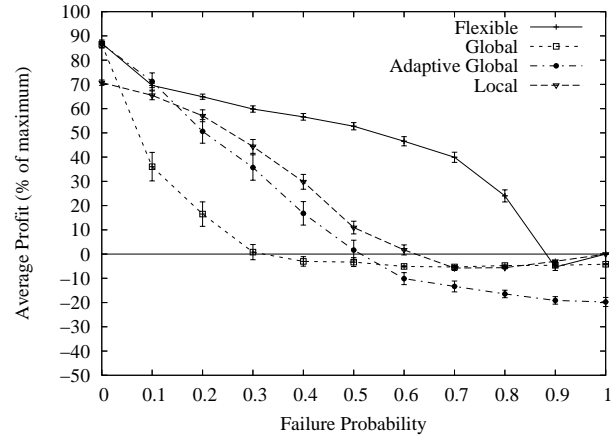


Figure 3: Performance of strategies in environments where providers fail, but give refunds.

flexible strategy outperforms or is comparable to the other strategies we tested. Averaged over all values for \bar{d} we tested, the flexible approach achieves a profit of 612.34 ± 16.09 , while the non-adaptive and adaptive global approaches achieve only 173.80 ± 26.85 and 183.60 ± 37.83 , respectively. The local strategy achieves an average profit of 212.30 ± 20.58 .

4.3.2 Environment 2: Failures with Refunds

In our next experiments, we were 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 experiment, 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 of such systems may include Web services or scientific Grids.

The results are shown in Figure 3 and clearly highlight similar trends as in the previous experiments for the non-flexible 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, and it 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 $\bar{f} = 0.8$. More specifically, at $\bar{f} = 0.6$, our strategy achieves an average profit of 930.86, with 95.7% of workflows executed successfully before t_{zero} , compared to the best non-flexible profit of only 34.34 with 19.1% of workflows successful (an approximately 27-fold improvement in average utility). At $\bar{f} = 0.8$, the flexible approach still completes 80.1% of workflows successfully, while the most successful non-flexible strategy completes 1.6%. This good performance is due to the considerably lower cost of invoking services redundantly, as now the consumer effectively pays for only those services that succeeded rather than all invoked services. Nevertheless, as before, we notice a small loss when the failure probability reaches $\bar{f} = 0.9$. For all values for \bar{f} tested, the flexible approach achieves an average profit of 901.31 ± 14.84 , the global approaches achieve 200.60 ± 26.07 (non-adaptive) and 335.80 ± 34.30 (adaptive), while the local approach achieves 473.68 ± 17.13 .

4.3.3 Environment 3: Different Market Conditions

Finally, we tested the performance of the strategies in environments

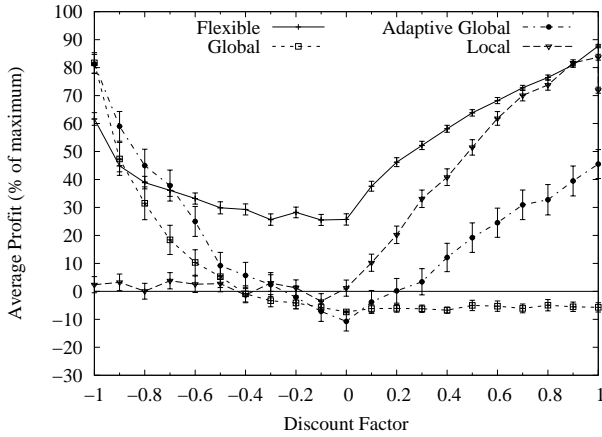


Figure 4: Performance of strategies when advance provisioning is preferred (negative discount) and when on demand is preferred (positive discount).

where either advance or on demand (just-in-time) provisioning is preferred and results in 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 under-utilised and therefore offer discounted services at the last minute. To express this preference, we vary a discount factor, \bar{d} , from -1 to 1. When negative, this indicates a preference for early (advance) provisioning and when positive, on demand provisioning 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 $|\bar{d}|$. We consider all offers generated for the current time step, t_i , as provisioned on demand, and any offers generated for t_{i+40} and beyond as provisioned in advance. Between these two, we vary the discount factor linearly. For example, when $\bar{d} = -0.6$, $\bar{f} = 0.5$ 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 our first experimental setup, but keep \bar{f} at 0.5, and now set $r_b = 0.5$ and $r_d = 5$, to ensure that discounted offers are available only at their respective time steps.

Figure 4 shows the results in these environments. Here, we note that the non-flexible strategies perform well only in extreme conditions — the global approaches excel when advance provisioning is preferred, while the local strategy performs well as \bar{d} tends to 1. When neither advance nor on demand provisioning is strongly preferred, none of the non-flexible strategies do well, as most of the services in the market are unreliable. In fact, at $\bar{d} = -0.1$, these strategies all make a net loss. In contrast to this, the flexible strategy manages to achieve a high profit over all environments, and, in most cases, significantly outperforms all other strategies. This is because the flexible strategy adjusts its provisioning strategies to the environment: at $\bar{d} = -1$, it provisions services, on average, 47.40 ± 1.46 time steps in advance, at $\bar{d} = 0$, this drops to 23.04 ± 0.67 and at $\bar{d} = 1$, it provisions only 6.36 ± 0.41 time steps ahead. However, we also note that the flexible strategy is now outperformed in two cases: at $\bar{d} = -0.9$ and $\bar{d} = -1$. In these cases, it suffers from not provisioning all offers in advance (and thereby producing a tight-fitting but reliable schedule). Instead, the strategy continues to provision only parts of the workflow (although now provisioning further ahead) and hence sometimes exceeds t_a . Nevertheless, when averaging over all values for \bar{d} considered here, the flexible strategy achieves an average utility of 953.87 ± 9.56 ,

while the global approaches achieve only 109.69 ± 17.78 (non-adaptive) and 428.40 ± 24.70 (adaptive), and the local approach achieves 516.37 ± 15.17 .

5. CONCLUSIONS AND FUTURE WORK

In this paper, we have proposed a novel provisioning strategy that extends the state of the art in several ways. First, our approach provisions only part of a workflow at a time and adapts its decisions at run-time, making the strategy more robust 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.

We believe 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. Although we use the contract net protocol, our approach should be applicable to other market mechanisms. In particular, the high-level decisions discussed in Section 3.2 could refer to strategies for participating in auctions, to carry out bilateral negotiations or simply for selecting services published on a registry.

There are several ways in which we plan to extend our work. First, we plan 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.

6. ACKNOWLEDGEMENTS

This work was funded by the Engineering and Physical Sciences Research Council (EPSRC) and a BAE Systems studentship.

7. REFERENCES

- [1] R. Aggarwal, K. Verma, J. Miller, and W. Milnor. Constraint driven web service composition in METEOR-S. In *Proc. IEEE Int. Conf. on Services Computing, China*, pages 23–30, 2004.
- [2] J. Collins, C. Bilot, M. Gini, and B. Mobasher. Decision processes in agent-based automated contracting. *IEEE Internet Comput.*, 5(2):61–72, 2001.
- [3] S. Kirkpatrick, J. C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.
- [4] D. G. Malcolm, J. H. Roseboom, C. E. Clark, and W. Fazar. Application of a technique for research and development program evaluation. *Oper. Res.*, 7(5):646–669, 1959.
- [5] E. Sirin, B. Parsia, and J. Hendler. Template-based composition of semantic web services. In *AAAI Fall Symposium on Agents and the Semantic Web, USA*, pages 85–92, 2005.
- [6] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Trans. Comput.*, 29(12):1104–1113, 1980.
- [7] S. Stein. *Flexible Service Provisioning in Multi-Agent Systems*. PhD thesis, School of Electronics and Computer Science, University of Southampton, 2008.
- [8] S. Stein, N. R. Jennings, and T. R. Payne. Provisioning heterogeneous and unreliable providers for service workflows. In *Proc. 22nd AAAI Conf. on AI, Canada*, pages 1452–1458, 2007.
- [9] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-aware middleware for web services composition. *IEEE Trans. Softw. Eng.*, 30(5):311–327, 2004.