An Effective Strategy for the Flexible Provisioning of Service Workflows

Sebastian Stein, Terry R. Payne, Nicholas R. Jennings

School of Electronics and Computer Science,
University of Southampton,
Southampton, SO17 1BJ, UK.
{ss04r,trp,nrj}@ecs.soton.ac.uk

Abstract. Recent advances in service-oriented frameworks and semantic Web technologies have enabled software agents to discover and invoke resources over large distributed systems, in order to meet their high-level objectives. However, most work has failed to acknowledge that such systems are complex and dynamic multi-agent systems, where service providers act autonomously and follow their own decision-making procedures. Hence, the behaviour of these providers is inherently uncertain — services may fail or take uncertain amounts of time to complete. In this work, we address this uncertainty and take an agent-oriented approach to the problem of provisioning service providers for the constituent tasks of abstract workflows. Specifically, we describe an algorithm that uses redundancy to deal with unreliable providers, and we demonstrate that it achieves an 8-14% improvement in average utility over previous work, while performing up to 6 times as well as approaches that do not consider service uncertainty. We also show that our algorithm performs well in the presence of inaccurate service performance information.

1 Introduction

Fuelled by technological advances and the pervasiveness of communication networks, such as the Internet, modern computing devices are increasingly part of complex distributed systems, allowing unprecedented access to a wide range of resources, information sources and remote computing facilities. In this context, service-oriented computing is emerging as a popular approach for allowing autonomous agents to discover and invoke such distributed resources, encapsulated as computer services [1]. In most application scenarios, from computational Grids to automated business processes, multiple services are often combined as part of workflows, thus enabling the consumer to achieve complex goals [2].

Now, a key feature of large distributed systems is that participants are often autonomous agents that follow their own decision-making procedures [3]. This means that the behaviour of service providers is beyond the control of the consumer, and is thus inherently uncertain. This might be manifested by uncertain service durations (e.g., when a provider prioritises service requests from some consumers over others, or when it allocates a variable fraction of its available

resources to each request), or by unpredictable failures that might occur when a provider is unable (or unwilling) to honour a service request. Such uncertainty is a critical issue for service consumers that need to execute large workflows of interdependent tasks in distributed systems, especially when there are time constraints and when service providers demand remuneration.

So far, this issue has received relatively little attention in the current literature. Instead, most work is concerned with matchmaking techniques that simply identify any single service provider able to fulfil a given goal, based on service descriptions that are assumed to be accurate and truthful [4]. While some workflow languages contain static exception handling methods for dealing with service failures [5], these are inflexible, as they: deal only reactively with problems; need to be specified manually; and usually rely on cooperative services that signal failures and can be rolled back.

To address failures more proactively, some approaches have used quality-of-service measures to place constraints on services or to optimise a weighted sum of various parameters [6], but these require appropriate constraints and weights to be specified by a user. Other research uses utility-theory to choose optimal services in the presence of uncertainty [7]. However, these approaches generally provision only single providers for each task of a workflow, which leads to brittle workflows that are highly vulnerable to single failures. An exception to this is work on highly unreliable services in public-resource computing and peer-to-peer systems, where services are invoked redundantly to increase the overall reliability [8]. While dealing with uncertainty to some extent, such work assumes services to be provided for free, and uses a fixed level of redundancy regardless of the actual reliability of services, time constraints or the value of the workflow.

To deal with this, we previously described a strategy that uses stochastic quality-of-service information about providers to flexibly provision multiple providers for particularly failure-prone tasks in a workflow [9]. In that work, we showed that this redundancy allows the consumer to deal proactively with service uncertainty, and we demonstrated that the strategy worked well even if all available providers were highly unreliable. However, the strategy used a simple estimate of the overall workflow completion time based only on the mean time of each task. Furthermore, we assumed accurate performance information to be available about every task, which may be unrealistic in open and dynamic systems, where such information is often noisy and inaccurate.

In this paper, we substantially extend our previous work in the the following three ways. First, we describe how to calculate and use task duration variance in order to obtain a better estimate for the overall workflow completion time. Second, we investigate the sensitivity of our strategy in the presence of inaccurate service performance information. Third, we show how our strategy performs in more complex environments than previously considered.

The remainder of this paper is structured as follows. In Section 2, we describe the model of a service-oriented system. Then, in Section 3, we outline our extended provisioning strategy. In Section 4, we empirically evaluate the strategy and compare it to our previous work. We conclude in Section 5.

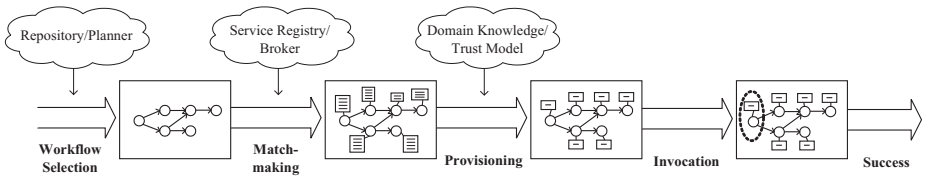


Fig. 1. Lifecycle of a workflow.

2 Service-Oriented System Model

In order to provide a formal basis for our work, we briefly describe our system model in this section and relate it to the lifecycle of a workflow, as shown in Fig. 1. As is common in related work, we assume that a service consumer has selected an abstract workflow, describing the types of tasks and their ordering constraints required to achieve some high-level goal (the first stage in Fig. 1). This may originate from a workflow repository or a planner that operates on service types. More formally, we represent such an abstract workflow as a directed acyclic graph $W = (T, E)$, where $T = \{t_1, t_2, \ldots, t_{|T|}\}$ is the set of nodes (the tasks) and $E \in T \leftrightarrow T$ is the set of edges (the ordering constraints). Furthermore, we use a reward function $u \in \mathbb{Z}_0^+ \rightarrow \mathbb{R}$ that denotes the utility of completing the workflow at a given time step t. This is based on a maximum reward $u_{\max}$ that is awarded if the workflow is completed no later than a deadline d. If the workflow is late, a cumulative penalty δ is deducted at each time step, until the consumer no longer receives a reward (it does not receive a negative reward — rather, we assume that the workflow has simply failed). Formally, we define u as:

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

Once a workflow has been selected, a consumer discovers appropriate service providers for each of the tasks by submitting the abstract task descriptions to a matchmaking process [10] (as shown in the matchmaking stage of Fig. 1). This might be achieved by contacting a service registry or communicating with a broker. We represent this process using a matching function $m \in T \rightarrow \wp(S)$, where $S = \{s_1, s_2, \ldots, s_{|S|}\}$ is the set of all service providers. This maps each task t_i to a subset of S to represent the providers that are able to satisfy t_i.

In the next stage, the agent provisions service providers for each task. Here, it makes a decision about how to allocate individual service providers to the constituent tasks of the workflow. We focus on this particular stage, because it allows the agent to use appropriate domain knowledge or information provided by a trust model [11] to make predictions about the feasibility of a workflow, and, where necessary, invest additional resources to provision providers redundantly for particularly failure-prone tasks. In this context, we assume that some limited

performance information about the population of providers for each task t_i is known to the consumer[1]:

- $S_i \subseteq S$ is the set of suitable service providers (as given by $m(t_i)$).
- $f_i \in [0, 1]$ is the probability that a randomly chosen provider from S_i will default or otherwise fail to deliver a satisfactory result.
- $D_i(t) \in [0, 1]$ is the cumulative probability distribution, representing the execution duration of a single service provider (randomly chosen from S_i and assuming it is successful). In particular, $D_i(t)$ is the probability that the provider will take t time steps or less to complete the service.
- $c_i \in \mathbb{R}$ is the cost of invoking a provider from S_i. This may be a financial remuneration or a communication cost.

After provisioning, the consumer starts invoking providers for the tasks of the workflow according to the ordering constraints given by E (the fourth stage in Fig. 1). Here, we assume that providers are only invoked at integer time steps, and, as is common in the Web services domain, this is done *on demand* when providers are required. Hence, the cost for each is paid only at the time of invocation (but regardless of the eventual outcome). When invoked, a provider successfully completes the assigned task t_i with probability $1 - f_i$. The duration of a successful service execution is distributed according to D_i, after which the service consumer is notified of success. When a provider fails, we assume that it does so silently (i.e., no response is given to the consumer). This is realistic in distributed systems, where service providers generally do not reveal their internal state, and where network or machine failures can cause communication losses.

Furthermore, as there may be several matching providers for a task, the consumer can invoke more than one service for this task at the same time. In this case, the consumer has to pay each provider separately, and the task is completed when the earliest service has executed successfully (if any). When all providers seem to have failed, the consumer may invoke new providers for this task. In this case, the consumer will ignore the previously invoked providers and assign the task to the newly provisioned set of services.

Having outlined our basic system model and workflow lifecycle, we now continue to describe our extended provisioning strategy.

3 Flexible Provisioning

In this section, we present a heuristic provisioning strategy that decides dynamically how to provision providers based on the information that is available about each task of the workflow. This strategy is an extension of the work described in [9], and now includes mechanisms for calculating the duration variances of individual tasks and using them to estimate the overall duration distribution of the workflow. In Section 3.1, we begin by giving a high-level overview of our approach, and then, in Sections 3.2 and 3.3, we detail the calculations at the task and workflow level, respectively.

[1] In [12] we consider the case where more detailed information about individual providers is available.

3.1 Provisioning Problem

As discussed in more detail in our previous work [9], we use two forms of redundancy to deal with uncertainty: parallel and serial provisioning. More specifically, for each task t_i, we determine a number of providers (n_i) to invoke in parallel. Provisioning multiple providers in such a way increases the probability of success for that task, and also decreases its expected duration (as the consumer can proceed as soon as one of the providers is successful). Furthermore, to deal with the case where none of the providers is successful, we determine a maximum waiting time (w_i) before the consumer invokes a new set of n_i service providers. However, using redundancy in such a way increases the cost for the overall workflow, and, hence, we aim to maximise the expected net profit (denoted $\bar{u}(\boldsymbol{n}, \boldsymbol{w})$):

$$\max_{\boldsymbol{n},\boldsymbol{w}\in\mathbb{N}^{|T|}} \bar{u}(\boldsymbol{n}, \boldsymbol{w}), \tag{2}$$

where $\boldsymbol{n}$ and $\boldsymbol{w}$ are vectors, whose ith elements correspond to the number of parallel providers (n_i) and the waiting time (w_i) of each task t_i of the workflow. Furthermore, we define the expected net profit $\bar{u}(\boldsymbol{n}, \boldsymbol{w})$ as the difference of the expected reward $\bar{r}(\boldsymbol{n}, \boldsymbol{w})$ and the expected cost $\bar{c}(\boldsymbol{n}, \boldsymbol{w})$ when provisioning the workflow using the vectors $\boldsymbol{n}$ and $\boldsymbol{w}$:

$$\bar{u}(\boldsymbol{n}, \boldsymbol{w}) = \bar{r}(\boldsymbol{n}, \boldsymbol{w}) - \bar{c}(\boldsymbol{n}, \boldsymbol{w}). \tag{3}$$

However, even the calculation of this objective function for any given vectors $\boldsymbol{n}$ and $\boldsymbol{w}$ is intractable, because it requires the calculation of the overall duration distribution (this is known to be a $\#P$-complete problem [13]). For this reason, we use a heuristic strategy that estimates the expected reward and cost of a random initial choice for $\boldsymbol{n}$ and $\boldsymbol{w}$, and then performs steepest-ascent hill-climbing to find a good solution. More specifically, at each iteration, our hill-climbing algorithm generates a large set of neighbours of the current best choice for $\boldsymbol{n}$ and $\boldsymbol{w}$ by performing small changes to it. To generate each neighbour, the algorithm picks one component of either vector (n_i or w_i), then increases or decreases it by either 1 or a random amount. This is done systematically for each task, thus producing $8 \cdot |T|$ neighbours in total. Each neighbour is then evaluated, the best is chosen and this process is repeated until no more improvements can be made.

Now, in our previous work, we used a simple calculation for estimating the expected reward $\bar{r}(\boldsymbol{n}, \boldsymbol{w})$. In particular, our mechanism assumed task durations to be deterministic and then used the longest path in the workflow to calculate an overall, deterministic duration $\tilde{t}$ (conditional on success). Hence, it estimated the expected reward as $\tilde{r} = p \cdot u(\tilde{t})$, where p is the overall success probability of the workflow, and $u(\tilde{t})$ is the reward at time $\tilde{t}$ (omitting the parameters $\boldsymbol{n}$ and $\boldsymbol{w}$ for brevity). A major shortcoming of this approach is that it does not consider the variance of task durations at all, and so often overestimates the expected reward. To address this, we artificially increased our estimate for $\tilde{t}$ by a constant 20%, but such an approach still does not consider variance, and is not guaranteed to work well in all environments.

To overcome this, we now use an improved heuristic that does not assume a deterministic duration, but rather estimates the probability distribution of the workflow duration. To this end, we estimate the expected profit as:

$$\tilde{u} = p \int_0^\infty d_W(x) u(x) \, \mathrm{d}x \; - \tilde{c}, \tag{4}$$

where $d_W(x)$ is a probability density function that estimates the overall duration of the workflow, and $\tilde{c}$ is an estimate of the expected overall cost (using a continuous probability function allows us to derive a simple and concise solution in closed form). This equation is central to our heuristic strategy, and, in the following two sections, we describe how to calculate its components.

3.2 Local Task Calculations

In order to solve Equation 4, we first calculate four parameters for each task t_i in the workflow: its success probability, expected cost, expected duration and duration variance. These are discussed in more detail below.

Success Probability (p_i): This is the probability that the task will be successful, regardless of the eventual finishing time. Because we assume that the consumer will continue invoking providers until the task is completed (with the given waiting time w_i between invocations), it is the probability that at least one provider from the set S_i is successful within its allocated time:

$$p_i = 1 - (1 - (1 - f_i) \cdot D_i(w_i))^{|S_i|}. \tag{5}$$

Expected Cost ($\bar{c}_i$): This is expected overall cost that the consumer will spend on the task. To calculate it, we let $\hat{f}_i = (1-(1-f_i) \cdot D_i(w_i))^{n_i}$ be the probability that a single invocation of n_i parallel providers is unsuccessful within their given waiting time, $m = \lfloor |S_i|/n_i \rfloor$ the maximum number of such invocations (with n_i providers), and $r = |S_i| \bmod n_i$ the number of service providers available for the last invocation. The expected cost is then:

$$\bar{c}_i = n_i c_i \cdot \frac{1 - \hat{f}_i^m}{1 - \hat{f}_i} + \hat{f}_i^m c_i r. \tag{6}$$

Expected Duration ($\bar{t}_i$): This is the expected time the providers will take to complete the task (conditional on overall success). Here, we let $\mu_i = \hat{D}_i(w_i)^{-1} \sum_{k=1}^{w_i} k \cdot (\hat{D}_i(k) - \hat{D}_i(k-1))$ be the mean time to success of a single successful invocation of n_i service providers (where $\hat{D}_i(x) = 1 - (1 - (1 - f_i) \cdot D_i(x))^{n_i}$ is the cumulative non-conditional probability that at least one out of n_i services has finished successfully by time x), λ_i the corresponding mean time for the final invocation of r providers, and $\check{f}_r$ the failure probability of that invocation (calculated analogously to $\hat{f}_i$). Hence, we calculate the expected duration as:

$$\bar{t}_i = \frac{1}{p_i}(\mu_i(1-\hat{f}_i^m) + w_i \frac{\hat{f}_i - m\hat{f}_i^m + (m-1)\hat{f}_i^{m+1}}{1-\hat{f}_i} + \hat{f}_i^m(1-\check{f}_r)(\lambda_i + m w_i)). \tag{7}$$

Variance of Duration (σ_i^2): This is the variance of the task duration. To calculate it, we let C_i be a random variable representing the duration of the task, conditional on its success. Then, $\mathrm{E}(C_i)$ is its expected value (note that $\mathrm{E}(C_i) = \bar{t}_i$), $\mathrm{E}(C_i^2)$ is its expected square, and $\mathrm{VAR}(C_i)$ is its variance. Hence,

$$\sigma_i^2 = \mathrm{VAR}(C_i) = \mathrm{E}(C_i^2) - \mathrm{E}(C_i)^2 = \mathrm{E}(C_i^2) - \bar{t}_i^2. \tag{8}$$

In order to obtain $\mathrm{E}(C_i^2)$, we consider two cases: (1) the task is completed successfully within the first m invocations, and (2) it is completed during the last invocation of r providers (if $r \neq 0$). We denote the durations in each case by the random variables A_i and B_i, respectively. This allows us to treat both cases separately, and, letting P_A and P_B be the respective probabilities of each case occurring (conditional on overall success), we write $\mathrm{E}(C_i^2)$ as:

$$\mathrm{E}(C_i^2) = P_A\mathrm{E}(A_i^2) + P_B\mathrm{E}(B_i^2) = \frac{1-\hat{f}_i^m}{1-\check{f}_r\hat{f}_i^m}\mathrm{E}(A_i^2) + \frac{\hat{f}_i^m(1-\check{f}_r)}{1-\check{f}_r\hat{f}_i^m}\mathrm{E}(B_i^2). \tag{9}$$

Next, we note that each duration is divided into a time period spent waiting during any unsuccessful invocations (we denote these as A_{Wi} and B_{Wi}), and a period spent during the final invocation until the first provider is successful (we denote these as A_{Di} and B_{Di}). We note that these are independent in our model, and treat case (1) first:

$$\begin{aligned}\mathrm{E}(A_i^2) &= \mathrm{VAR}(A_i) + \mathrm{E}(A_i)^2 \\ &= \mathrm{VAR}(A_{Wi}) + \mathrm{VAR}(A_{Di}) + (\mathrm{E}(A_{Wi}) + \mathrm{E}(A_{Di}))^2 \\ &= \mathrm{E}(A_{Wi}^2) + \mathrm{E}(A_{Di}^2) + 2\mathrm{E}(A_{Wi})\mathrm{E}(A_{Di}).\end{aligned} \tag{10}$$

Now, $\mathrm{E}(A_{Di}) = \mu_i$, and $\mathrm{E}(A_{Di}^2)$ can be similarly calculated by multiplying the term inside the summation by k^2 instead of k. Furthermore, we obtain $\mathrm{E}(A_{Wi})$ from Equation 7, and then derive $\mathrm{E}(A_{Wi}^2)$ in a similar way:

$$\mathrm{E}(A_{Wi}) = \frac{w_i(\hat{f}_i - m\hat{f}_i^m + (m-1)\hat{f}_i^{m+1})}{(1-\hat{f}_i)(1-\hat{f}_i^m)} \tag{11}$$

$$\begin{aligned}\mathrm{E}(A_{Wi}^2) &= \frac{(1-\hat{f}_i)w_i^2}{1-\hat{f}_i^m}\sum_{k=0}^{m-1} k^2\hat{f}_i^k \\ &= w_i^2(\hat{f}_i + \hat{f}_i^2 - m^2\hat{f}_i^m - (2m+1-2m^2)\hat{f}_i^{m+1} \\ &\quad + (2m-1-m^2)\hat{f}_i^{m+2})(1-\hat{f}_i^m)^{-1}(1-\hat{f}_i)^{-2}.\end{aligned} \tag{12}$$

Treating case (2) next, we calculate $\mathrm{E}(B_i^2)$ analogously to Equation 10. This is easier, as the waiting time B_{Wi} is constant (mw_i). Hence,

$$\begin{aligned}\mathrm{E}(B_i^2) &= \mathrm{E}(B_{Wi}^2) + \mathrm{E}(B_{Di}^2) + 2\mathrm{E}(B_{Wi})\mathrm{E}(B_{Di}) \\ &= (mw_i)^2 + \mathrm{E}(B_{Di}^2) + 2mw_i\mathrm{E}(B_{Di}).\end{aligned} \tag{13}$$

The remaining terms, $\mathrm{E}(B_{Di})$ and $\mathrm{E}(B_{Di}^2)$, are calculated as $\mathrm{E}(A_{Di})$ and $\mathrm{E}(A_{Di}^2)$, discussed above. Combining all terms to solve Equation 8 gives us a

closed form for calculating the variance of the duration of a given task. With these parameters for each task, we now continue to describe how they are combined over the entire workflow to solve our heuristic function (Equation 4).

3.3 Workflow Calculations

In order to solve Equation 4, we require the overall success probability p, a distribution for the workflow duration $d_W(x)$, and the estimated overall cost $\tilde{c}$. The success probability p is simply the probability that all tasks are eventually successful, and the estimated total cost $\tilde{c}$ is the sum of all task costs, each multiplied by the probability that the task is ever reached (denoted r_i):

$$p = \prod_{\{i|t_i \in T\}} p_i. \tag{14}$$

$$\tilde{c} = \sum_{\{i|t_i \in T\}} r_i \bar{c}_i \tag{15}$$

$$r_i = \begin{cases} 1 & \text{if } \forall t_j \cdot ((t_j \mapsto t_i) \notin E) \\ \prod_{\{j|(t_j \mapsto t_i) \in E\}} p_j & \text{otherwise.} \end{cases} \tag{16}$$

To estimate the workflow duration function $d_W(x)$, we use a technique from operations research [14]. In particular, we consider the *critical path* of the workflow (i.e., the path that maximises the sum of all mean task durations along it) and obtain the sum of all mean task durations (λ_W) and variances on it (v_W). Exploiting the central limit theorem, we then approximate the duration of the workflow using a normal distribution with mean λ_W and variance v_W:

$$d_W(x) = \frac{1}{\sqrt{v_W 2\pi}} \, \mathrm{e}^{-\frac{(x-\lambda_W)^2}{2v_W}}. \tag{17}$$

Now, to solve Equation 4, we let $D_W(x) = \int_{-\infty}^{x} d_W(y)\,\mathrm{d}y$ be the cumulative probability function[2] of $d_W(x)$, we let $D_{\max} = D_W(t_{\max})$ be the probability that the workflow will finish within the deadline $t_{\max}$ and $D_{\text{late}} = D_W(t_0) - D_W(t_{\max})$ the probability that the workflow will finish after the deadline but no later than time $t_0 = u_{\max}/\delta + t_{\max}$ (both conditional on overall success).

Next, we consider three distinct cases, based on the form of $u(t)$ (Equation 1). First, the workflow may finish within the deadline $t_{\max}$ — this happens with probability $D_{\max}$ and results in the full reward, $u_{\max}$. Second, the workflow may finish after t_0 — this happens with probability $1 - D_W(t_0)$, and here the consumer receives no reward (and so we can ignore it). Finally, the workflow may finish between these two times, which happens with probability D_{late}. Because $u(t)$ is linear on this interval, we calculate the expected reward in this case by

[2] This common function is usually approximated numerically. In our work, we use the SSJ library (`http://www.iro.umontreal.ca/~simardr/ssj`).

applying $u(t)$ to the mean time on the interval, which we denote by $\bar{t}_{\text{late}}$. Hence, we re-write Equation 4, concluding our heuristic utililty function:

$$\tilde{u} = p \cdot (D_{\max} \cdot u_{\max} + D_{\text{late}} \cdot u(\bar{t}_{\text{late}})) - \tilde{c} \tag{18}$$

$$\text{with} \quad \bar{t}_{\text{late}} = \frac{1}{D_{\text{late}}} \int_{t_{\max}}^{t_0} d_W(x) x \, \mathrm{d}x$$

$$= \lambda_W + (e^{\frac{-(t_{\max}-\lambda_W)^2}{2v_W}} - e^{\frac{-(t_0-\lambda_W)^2}{2v_W}}) \cdot \frac{\sqrt{v_W}}{D_{\text{late}} \cdot \sqrt{2\pi}}. \tag{19}$$

Using this heuristic function, it is now possible to use steepest-ascent hill-climbing as described at the beginning of this section. Through observations, we have seen that our hill-climbing algorithm quickly converges to a good solution[3]. In particular, the heuristic function $\tilde{u}$ can be solved efficiently in quadratic time. The bottleneck here is the calculation for Equations 15 and 16. However, after the initial calculation, only small adjustments need to be made at each iteration of the hill-climbing procedure, further reducing the run-time of calculating $\tilde{u}$. In this case, it is bounded by the critical path problem required to obtain λ_W and v_W used in Equation 19, which has a run-time in $O(|T| + |\mathcal{E}|)$ where $|T|$ is the number of tasks in the workflow and $|\mathcal{E}|$ the number of non-transitive edges[4].

Having concluded our discussion of the improved provisioning strategy, we now describe a set of empirical experiments to evaluate our work.

4 Empirical Evaluation

To investigate the performance of our strategy and compare it to our previous work, we conduct a set of empirical experiments using the same methodology as described in [9]. To summarise briefly, we test our strategy by randomly generating a set of service providers and a single workflow according to a set of controlled variables that specify a particular environment. Then, we provision the workflow using our strategy and simulate its execution, recording the overall net profit achieved. This is repeated 1000 times for each environment to achieve statistical significance (we give all results with 95% confidence intervals and carry out ANOVA and two-sample t-tests as appropriate, both at the 99% confidence level). Throughout all experiments, we vary the failure probability of providers (as given by f_i) to evaluate our strategy in environments with varying levels of service unreliability, keeping all other variables constant for consistency. To provide a benchmark, we compare our results to a *naïve* strategy that provisions a single provider for each task of the workflow. This strategy represents currently prevalent approaches that ignore any service uncertainty and thus rely only on functional service descriptions to find any suitable provider for each task.

[3] On average, around six iterations are needed per task in the workflow. During the empirical evaluation of our algorithm (see Section 4), a solution was typically found within 250ms (10 tasks) or 5s (50 tasks) on a 3GHz Pentium 4 with 1GB RAM.

[4] We also assume that the probability density functions of service durations and expected values (e.g., μ_i in Equation 7) can be efficiently calculated or approximated.

In the remainder of this section, we first test our improved strategy using the same experimental setup as in our previous work to allow a direct comparison (Section 4.1). Because that setup considers relatively small workflows with homogeneous tasks, we then examine a more complex scenario with heterogeneous tasks and larger workflows in Section 4.2. Finally, in Section 4.3, we evaluate the performance of our strategy in the presence of inaccurate information.

4.1 Small Workflows

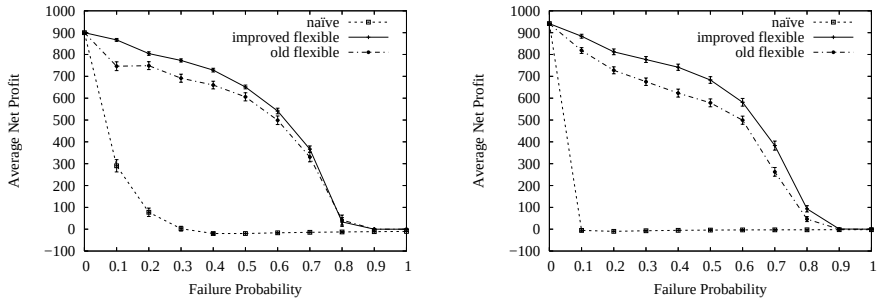
In order to compare the improved strategy to our previous work, we first consider the same environments as described in [9] (however, for consistency, we repeat all experiments). Here, workflows consist of 10 tasks in a strict sequence, they have a deadline of $d = 400$ time steps, a maximum utility $u_{\max} = 1000$ and a penalty of $\delta = 10$ per time step. Furthermore, there are 1000 providers able to satisfy each task. Each provider has a cost $c_i = 10$ and its duration distribution is a gamma distribution with parameters $k = 2$ and $\theta = 10$ (hence, the duration distribution for each provider is $D_i(t) = \gamma(k, t/\theta)\Gamma(k)^{-1}$).

Fig. 2(a) shows the results of our improved strategy (*improved flexible*), our previous strategy (*old flexible*) and the *naïve* strategy. Clearly, the performance of all three strategies is significantly different. Averaged over all failure probabilities, the *naïve* strategy achieves a net profit of only 106.00 ± 5.95, and, in fact, begins making an overall loss when the failure probability drops to 0.3 or lower. As discussed in previous work, our *old flexible* strategy clearly improves on this, yielding an average net profit of 474.91 ± 7.77. The results also indicate that our modified *improved flexible* strategy further yields a significant improvement in overall performance with an average net profit of 515.02 ± 4.02. At all failure probabilities from 0.1–0.7, the strategies achieve significantly different amounts of net profit, with the *improved flexible* strategy outperforming all others. At 0.8 and beyond, the difference between the two flexible strategies becomes less pronounced due to the overall low net profit, and it is no longer significant. However, they still outperform the *naïve* strategy. When the failure probability is 0, all strategies perform equally well, as a single provider for each task is always sufficient to complete the workflow in time.

To summarise, averaged over the failure probabilities discussed here, our *improved flexible* strategy achieves a $385.87\% \pm 8.69\%$ improvement over the *naïve* strategy, and an $8.44\% \pm 2.21\%$ improvement over the *old flexible* strategy.

4.2 Large Workflows

To show that our strategy performs well in more complex environments, we perform another set of experiments with workflows consisting of 50 tasks and heterogeneous task parameters. More specifically, for each experimental run, we first generate a workflow by randomly populating an adjacency matrix until 25% of all possible edges have been added (ignoring any that would result in a cyclic graph). This ensures that the workflow contains a considerable number of



(a) Small workflows (10 tasks). (b) Large workflows (50 tasks).

Fig. 2. Net profit of flexible strategies.

parallel tasks. We also assume there are 100 providers for each task, the deadline is $d = 1000$, the maximum reward $u_{\max} = 1000$ and the penalty $\delta = 1$.

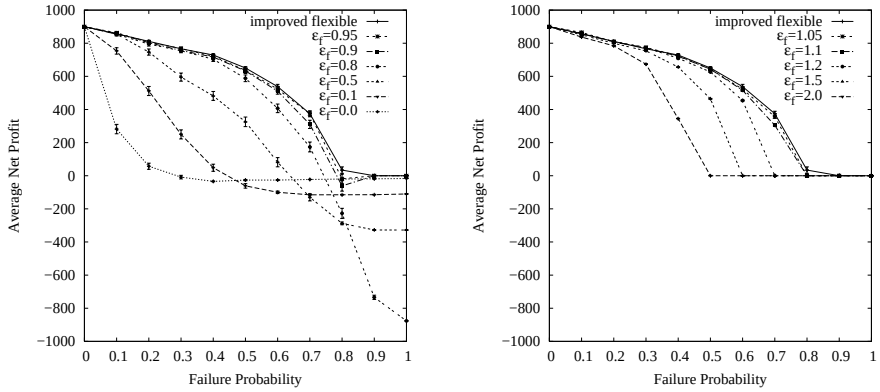
In order to vary the performance characteristics of the tasks in the workflow, we randomly assign each to one of seven different types of tasks with varying costs (c_i) and duration distributions (D_i), as shown in Table 1. For every experimental run, we also attach a failure probability to each type (to determine f_i) that is drawn from a beta distribution with parameters $\alpha = 10 \cdot f$ and $\beta = 10 - \alpha$, where f is the average failure probability of the environment (unless $f = 0$ or $f = 1$, in which case all tasks have the same failure probability). This adds further variance to the tasks, while ensuring that the overall average failure probability over all runs is close to f.

Fig. 2(b) shows similar results to those described in the previous section. When providers are always reliable (the failure probability is 0), there is no significant difference between the strategies. However, as soon as providers begin to fail, the *naïve* strategy begins to perform poorly and make an overall loss. The two flexible strategies achieve far better results and avoid making a loss in any environment. As before, our *improved flexible* strategy outperforms the *old flexible* strategy in most environments, except when the overall failure probability reaches 0.9, when there is no more significant difference between them.

When averaging over all failure probabilities, the *naïve* strategy achieves an average net profit of 81.53 ± 5.12, the *old flexible* achieves 470.21 ± 7.44 and the *improved flexible* achieves 536.12 ± 7.46. This means that our *improved flexible*

Table 1. Service types used to test complex workflows.

Type	**Cost**	**Duration**	**Mean**	**Variance**
T_1	0.1	Gamma(1,0.1)	0.1	0.01
T_2	0.1	Gamma(1,10)	10	100
T_3	1	Gamma(5,1)	5	5
T_4	1	Gamma(5,10)	50	500
T_5	2	Gamma(10,1)	10	10
T_6	2	Gamma(10,5)	50	250
T_7	2	Gamma(100,0.1)	10	1



(a) Underestimating f_i ($\epsilon_f < 1$). (b) Overestimating f_i ($\epsilon_f > 1$).

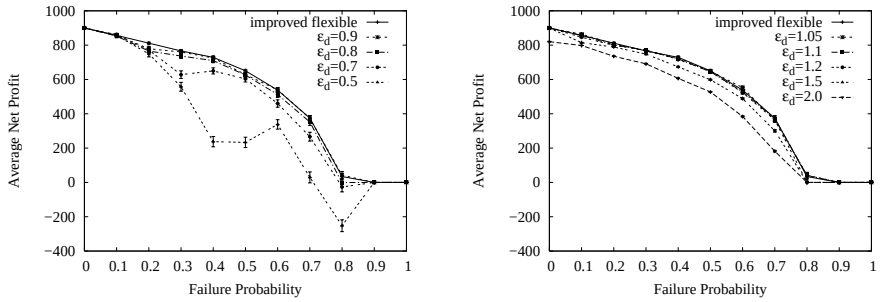
Fig. 3. Effect of incorrect failure probabilities.

strategy achieves a $557.60\% \pm 11.10\%$ improvement over the *naïve* strategy and a $14.02\% \pm 2.24\%$ improvement over the *old flexible* strategy.

4.3 Sensitivity Analysis

In order to evaluate the performance of our strategy in the presence of inaccurate information, we follow the same experimental setup as in Section 4.1, but now systematically introduce errors into the information that is available to a service consumer following the *improved flexible* strategy. To this end, we first evaluate the effect of relying on inaccurate failure probabilities, and then examine the impact of inaccurate service duration information. In both cases, we expect the performance of our strategy to decrease as the information becomes less accurate. However, because we rely on heuristic estimates, we anticipate that small inaccuracies will have little overall impact on the performance of the strategy.

In our first set of experiments, we consider the case where the consumer *underestimates* the failure probability of service providers. Hence, we multiply the actual values for the failure probabilities f_i by a scalar $\epsilon_f < 1$ to provide an inaccurate input to the *improved flexible* strategy. The results for various values of ϵ_f are shown in Fig. 3(a). In most cases, the average net profit gained by the strategy degrades gracefully as the performance information becomes more inaccurate. In fact, when the (true) failure probability is low in the environment (up to around 0.3), the strategy does well even if the information is up to 90% inaccurate (i.e., $\epsilon_f = 0.1$). However, when the failure probability rises to 0.7 and beyond, the impact of inaccurate information becomes more detrimental to the performance of the strategy. This is particularly evident when $\epsilon_f = 0.8$, which results in a large net loss at high failure probabilities. This is because the strategy provisions a large number of providers in parallel without detecting that the workflow is infeasible (and thus, it loses its high investment). Perhaps surprisingly, when information becomes even more inaccurate at high failure probabilities, the consumer begins to make smaller losses again. This is due to



(a) Underestimating duration ($\epsilon_d < 1$). (b) Overestimating duration ($\epsilon_d > 1$).

Fig. 4. Effect of incorrect failure probabilities.

the strategy provisioning less providers in parallel and therefore losing less of its investment when the workflow eventually fails. Despite the special case when $\epsilon_f = 0.8$, the results are promising and show that small inaccuracies in the information (up to 10%) have little or no effect on our strategy. In most other cases, performance degrades gracefully as the information becomes less accurate.

Next, we are interested in the trends resulting from *overestimating* the failure probability of service providers. Hence, we now multiply the failure probabilities by a scalar $\epsilon_f > 1$ to provide an inaccurate input to our strategy (using a failure probability of 1 whenever $f_i \cdot \epsilon_f > 1$). The results of this are shown in Fig. 3(b). Not surprisingly, the performance of the strategy simply degrades as the perceived failure probability rises. Because its behaviour is more conservative when it overestimates the failure probability of providers (it will provision unnecessarily many providers), it never makes a long-term loss. These results show that our strategy performs well, even when it significantly overestimates failure probabilities. In fact, the overall performance degrades only slightly when the failure probability is overestimated by 10% ($\epsilon_f = 1.1$). Even at 20% ($\epsilon_f = 1.2$), the performance is extremely good, and at 50% the strategy still performs reasonably well compared to the case with accurate information.

Apart from the failure probabilities, our strategy also relies on probability density functions for the duration of a service execution. Because these will most likely be based on past observations and can be subject to noise, we now examine the effect of inaccurate information about these functions. Here, we multiply the scale parameter θ of the underlying gamma distribution by a scalar ϵ_d to yield an inaccurate duration distribution. By varying the scale parameter, we ensure that the mean of the distribution is varied proportionally with ϵ_d (e.g., when $\epsilon_d = 0.5$, the consumer estimates the mean service execution time to be half of the true value), while the overall shape of the distribution stays the same.

Again, we first consider the case of *underestimating* the duration of service providers ($\epsilon_d < 1$). The results are shown in Fig. 4(a). Here, the strategy handles an error of up to 20% ($\epsilon_d = 0.8$) very well with only a marginal performance decrease. Even when the error rises to 30% ($\epsilon_d = 0.7$), the performance comes close to the accurate information case. However, as the information becomes even more inaccurate, the strategy performs increasingly badly. Also, the strategy

behaves more erratically at the same time — occasionally, the average net profit at a given level of inaccuracy increases as the failure probability rises (this is because the strategy constantly varies the balance between parallel and serial invocations, the latter of which is more susceptible to wrong duration estimates).

Finally, Fig. 4(b) shows the corresponding results when the consumer *overestimates* the service duration. Here, the performance again degrades slowly as the error rises. This is because the agent allocates unnecessarily long waiting times to the providers or provisions parallel providers when this is not needed. However, the loss in performance is clearly very small. This is because the consumer will occasionally wait longer than required or incur extra expenditure by provisioning parallel providers, but in many cases, the providers will simply complete their services earlier than anticipated and the consumer will be able to continue the workflow immediately and without penalty.

To conclude the sensitivity analysis, the results presented in this section show that our strategy is robust to small and moderate inaccuracies. In all cases, it performs well when the information provided is within 10% of the true value, and often errors up to 20% and 30% lead to only marginal decreases in performance, especially when the consumer is overly pessimistic (i.e., when it overestimates the failure probability or duration of services). Overall, performance generally degrades gracefully as larger errors are introduced into the information that is known about providers (until they are too large to be of any value to the consumer — e.g., as ϵ_d reaches 0.5).

We also identified one case where underestimating the failure probability of providers can lead to poor performance. However, this only occurs in very specific scenarios when providers are highly unreliable and when the error in information is a significant 20%. Hence, our strategy may benefit from identifying these conditions in advance, and we will consider this in future work. Nevertheless, the overall results presented here are promising, showing that our strategy is applicable even in environments where completely accurate performance information is unavailable (as will be typical in any large dynamic multi-agent system).

5 Conclusions

In this paper, we have extended our previous work by taking into consideration the variance of service duration times. This is important to consider when predicting the overall completion time of a workflow, and in empirical experiments, we have shown that our new strategy performs significantly better than our previous work. We have also given more extensive results in a variety of settings and shown that our strategy copes well in environments where performance information is not accurate.

Considering these results, our work is highly applicable in realistic scenarios, where software agents autonomously execute large workflows in service-oriented systems. Examples of such scenarios include: the Grid domain, where expensive and time-intensive data processing services are often required as part of complex workflows; peer-to-peer systems, where a vast amount of cheap, but

usually unreliable providers offer their services; and e-commerce applications, where the consumers often face highly valued workflows with strict deadlines (e.g., extensive supply-chains or large commercial projects).

In future work, we will extend our strategy to handle more advanced negotiation mechanisms rather than the fixed price model we use at the moment. Such mechanisms have been widely used in the context of multi-agent systems, and are now increasingly being employed in service-oriented systems to allow consumers and providers to reach service-level agreements prior to invocation. We also plan to improve the adaptivity of our strategy, utilising information about workflow progress to dynamically alter the provisioning of services.

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

References

1. Huhns, M.N., Singh, M.P.: Service-oriented computing: Key concepts and principles. IEEE Internet Comput. **9**(1) (2005) 75–81
2. Mandell, D.J., McIlraith, S.A.: Adapting BPEL4WS for the semantic web: The bottom-up approach to web service interoperation. In: Proc. 2nd Int. Semantic Web Conf. (ISWC03), Sanibel Island, USA, Springer (2003) 227–241
3. Jennings, N.R.: An agent-based approach for building complex software systems. Comm. ACM **44**(4) (2001) 35–41
4. McIlraith, S.A., Son, T.C.: Adapting Golog for Composition of Semantic Web Services. In: Proc. 8th Int. Conf. on Knowledge Representation and Reasoning (KR2002), Toulouse, France, Morgan Kaufmann (2002) 482–493
5. Curbera, F., Khalaf, R., Mukhi, N., Tai, S., Weerawarana, S.: The next step in Web services. Comm. ACM **46**(10) (2003) 29–34
6. Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., Sheng, Q.Z.: Quality driven web services composition. In: Proc. 12th Int. World Wide Web Conf. (WWW'03), Budapest, Hungary, ACM Press (2003) 411–421
7. Collins, J., Bilot, C., Gini, M., Mobasher, B.: Decision processes in agent-based automated contracting. IEEE Internet Computing **5**(2) (2001) 61–72
8. Anderson, D.P., Cobb, J., Korpela, E., Lebofsky, M., Werthimer, D.: SETI@home: an experiment in public-resource computing. Comm. ACM **45**(11) (2002) 56–61
9. Stein, S., Jennings, N.R., Payne, T.R.: Flexible provisioning of service workflows. In: Proc. 17th Eur. Conf. on AI (ECAI-06), Riva, Italy, IOS Press (2006) 295–299
10. Decker, K., Williamson, M., Sycara, K.: Matchmaking and brokering. In: Proc. 2nd Int. Conf. on Multi-Agent Systems (ICMAS'96), Kyoto, Japan. (1996) 432–433
11. Teacy, W.T.L., Patel, J., Jennings, N.R., Luck, M.: TRAVOS: Trust and reputation in the context of inaccurate information sources. JAAMAS **12**(2) (2006) 183–198
12. Stein, S., Jennings, N.R., Payne, T.R.: Provisioning heterogeneous and unreliable providers for service workflows. In: Proc. 6th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS07), Honolulu, Hawai'i. (2007)
13. Hagstrom, J.N.: Computational complexity of PERT problems. Networks **18** (1988) 139–147
14. Malcolm, D.G., Roseboom, J.H., Clark, C.E., Fazar, W.: Application of a technique for research and development program evaluation. Oper. Res. **7**(5) (1959) 646–669