Provisioning Heterogeneous and Unreliable Providers for Service Workflows

Sebastian Stein
ss04r@ecs.soton.ac.uk

Nicholas R. Jennings
nrj@ecs.soton.ac.uk

Terry R. Payne
trp@ecs.soton.ac.uk

School of Electronics and Computer Science,
University of Southampton,
Southampton, SO17 1BJ, UK.

ABSTRACT

In this paper, we address the problem of provisioning unreliable and heterogeneous service providers for the constituent tasks of abstract workflows. Specifically, we deal with unreliable providers by provisioning multiple service providers redundantly for specific tasks, and we employ a local search mechanism to choose among many heterogeneous providers that offer the same type of service. We empirically show that our strategy can achieve significant improvements over current approaches, and we demonstrate that it works well over a range of environments.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence

General Terms

Algorithms, Reliability

Keywords

Service-Oriented Computing, Service Provisioning, Workflows

1. INTRODUCTION

Service-oriented computing enables software applications to discover distributed resources and provision them dynamically for abstract workflows, in order to meet complex high-level goals. However, most of the recent work in this area ignores the fact that services in distributed systems are offered by autonomous agents, which follow their own decision-making procedures. Therefore, they are not guaranteed to provide the services they advertise or may take uncertain amounts of time to execute them. Furthermore, many providers may offer similar services at varying levels of quality and price. Such unreliability and heterogeneity must be addressed, especially when executing large workflows with time constraints, where service failures and delays can easily jeopardise the overall outcome.

In this paper, we address this problem by describing a strategy that uses service redundancy in a flexible manner to deal with unreliable providers. In particular, we extend our previous work in this area [3], by considering the case where multiple heterogeneous providers offer the same type of service. Moreover, we formalise the provisioning problem and show that it is NP-hard.

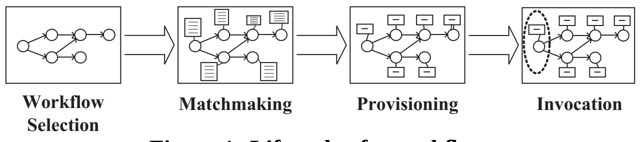

Figure 1: Lifecycle of a workflow.

2. WORKFLOW LIFECYCLE

Figure 1 shows the typical lifecycle of a workflow in our system model. Based on its current objective, a consumer first selects an abstract workflow (e.g., from a plan repository or a planner). Formally, we represent this as a directed acyclic graph $W = (T, E)$, where $T = \{t_1, t_2, \dots, t_{|T|}\}$ is a set of tasks and E is a set of precedence constraints. We also assume that the value of completing the workflow depends on the time of completion. More specifically, the consumer receives a maximum reward u_{max} if all tasks have been finished no later than a deadline d. For each full time step after that, a penalty δ is deducted from the reward until it reaches 0 (no negative reward is given — rather, we assume that the workflow has failed).

Next, the consumer uses a matchmaking process to discover suitable service providers for each task of the workflow (e.g., using a service registry or broker agent). We represent this using a matching function $m \in T \rightarrow \wp(S)$ that maps each task t_i to a subset of the set of all service providers, $S = \{s_1, s_2, \dots, s_{|S|}\}$. We also assume that some performance information is known about each service provider s_i: $f(s_i) \in [0, 1]$ is the *failure probability* of s_i; $D(s_i, t) \in [0, 1]$ is the (cumulative) *duration distribution* of s_i (i.e., the probability that the provider will take no longer than $t \in \mathbb{Z}^+$ time steps to complete a service invocation); and $c(s_i) \in \mathbb{R}$ is the *invocation cost* of s_i.

To cover the case where some providers share the same performance characteristics (e.g., when providers use a common implementation or when there is only limited information about groups of providers), we also assume that S is partitioned into service populations $P = \{P_1, P_2, \dots, P_{|P|}\}$, whose members are disjoint subsets of S with $\bigcup_i P_i = S$. Any two members s_x and s_y of a given population P_i always have the same failure probability, duration distribution and cost, and each task that is mapped to s_x by m is also mapped to s_y.

Once the consumer has identified matching service providers, it then provisions service providers for the workflow. Here, it makes a decision on which providers to invoke for each task, allowing it to make predictions about the outcome and feasibility of the workflow, and to invest additional resources in more reliable providers, or even multiple providers where necessary. This stage is the focus of our work, and it is further detailed in the following sections.

After provisioning, the consumer begins invoking service providers according to the ordering constraints of the workflow and its decisions made during provisioning. When invoking an appropriate provider $s_i \in m(t_j)$ for task t_j, the consumer first pays the cost, $c(s_i)$. The provider then successfully completes the task with probability $1 - f(s_i)$ in a random amount of time that is distributed according to $D(s_i, t)$. However, the outcome of an invocation is not revealed to the consumer until the service has been completed successfully (and failures are never explicitly communicated). A workflow is completed when all tasks have been successfully executed, and we evaluate the overall outcome using the difference of the reward and the total costs incurred (the net profit). In the following section, we formalise the service provisioning problem based on this measure.

3. SERVICE PROVISIONING PROBLEM

First, we define a provisioning allocation $\alpha \in (T \rightarrow \wp(S \times \mathbb{Z}_0^+))$ to be a mapping from workflow tasks to sets of service providers and associated invocation times. These times are relative to the time a task first becomes executable and indicate in what order different service providers will be invoked for the task, given that it has not already been completed. With this, we formulate service provisioning as follows:

Definition 1. (PROVISIONING): Given a workflow W and other information discussed above (u_{max}, d, δ, m, f, D and c), find a provisioning allocation α^* that maximises the expected net profit.

THEOREM 1. PROVISIONING *is NP-hard.*

PROOF. (Sketch). We prove Theorem 1 by giving a polynomial time transformation from a KNAPSACK instance to a PROVISIONING instance. To this end, for each item of the KNAPSACK instance, a new task with two matching providers (s_i and s_0) is created as part of a sequential workflow. These two providers never fail and represent, respectively, the choice between adding the item to the knapsack or not. Their duration functions are defined so that choosing s_i over s_0 results in a longer duration, corresponding to the weight of the item, and their costs are defined so that choosing s_i over s_0 results in a saving, corresponding to the value of the item. The overall capacity and target value of the knapsack are then mapped to the reward function u, so that any providers are provisioned if and only if the original knapsack instance can be satisfied. This transformation is performed in a time linear to the number of items, and as KNAPSACK is NP-hard, so is PROVISIONING. $\square$

4. HEURISTIC STRATEGIES

To deal with this intractability, we now present two heuristic strategies. The first provisions service providers in a static manner (Section 4.1), while the second varies the provisioning flexibly, using a novel method for predicting the workflow outcome (Section 4.2).

4.1 Fixed(n,w) Strategy

The first strategy, *fixed(n,w)*, is based closely on the *serial* and *parallel* strategies in [3] and uses static redundancy to deal with unreliable providers. More specifically, for each task t_i, the *fixed(n,w)* strategy always provisions n randomly chosen providers given by $m(t_i)$ for the time the task is started, and then again every w time units (if there providers left). A special case is the *naïve* strategy, which we define as *fixed(1,∞)*. This strategy models current approaches that do not consider service non-determinism and thus provision a single service provider for each task, assuming it will complete the task successfully.

However, *fixed(n,w)* requires a human user to specify n and w manually, and it does not consider provider heterogeneity. To address this, we present the *flexible* strategy in the following section.

4.2 Flexible Provisioning

The *flexible* strategy uses a hill-climbing algorithm to search the space of all provisioning allocations for a good solution. In particular, the algorithm begins with a random choice for α, and then systematically applies a number of random changes to the provisioned providers for each task (e.g., removing or adding single providers, or changing the invocation times). Whenever such a random change results in a higher predicted net profit, the algorithm adopts the new allocation and continues exploring the neighbours of that allocation. When all tasks of the workflow have been considered a fixed number of times without any improvements[1], the algorithm returns the current allocation.

Now, obtaining the expected net profit of a given allocation is central to this hill-climbing algorithm, but calculating it requires a distribution over the completion time. This is known to be an intractable problem [1], and so we use a heuristic estimate, $\tilde{u}$, of the expected net profit of a given allocation α (for brevity, we omit this parameter):

$$\tilde{u} = p \int_0^\infty d_W(x) u(x) \, \mathrm{d}x \; - \tilde{c} \qquad (1)$$

where p is the probability that all tasks will eventually finish successfully, d_W is a probability density function to estimate the overall duration of the workflow (conditional on success) and $\tilde{c}$ is the estimated expected cost (not conditional on success). To calculate these components, we first obtain a number of local parameters for each task t_i, and then, these are combined to solve Equation 1.

For the local calculations, we let $\hat{D}(s_x, t) = (1 - f(s_x)) \cdot D(s_x, t)$ be the probability that a service provider s_x has completed its service successfully within no more than t time steps (not conditional on overall success). We also let $S_i(t)$ be the set of provisioned service providers and associated times that are invoked at most t time steps after task t_i was started. Then, $T_i(t) = 1 - \prod_{(x,y) \in S_i(t)} (1 - \hat{D}(x, t - y))$ is the probability that task t_i is completed successfully within no more than t time steps. With this, we calculate the following four local parameters for each task:

- **Success Probability (p_i):** This is the probability that the task will be completed successfully no later than t_{max} time steps after the task was started[2]. Hence, $p_i = T_i(t_{\text{max}})$.
- **Mean Completion Time (λ_i):** This is the mean time until the task is completed, conditional on the task being successfully completed within t_{max} time steps: $\lambda_i = \frac{1}{p_i} \sum_{t=1}^{t_{\text{max}}} t \cdot (T_i(t) - T_i(t-1))$.
- **Variance (v_i):** This is the variance of the completion time, conditional on successful completion as above: $v_i = -\lambda_i^2 + \frac{1}{p_i} \sum_{t=1}^{t_{\text{max}}} t^2 \cdot (T_i(t) - T_i(t-1))$.
- **Expected Cost (c_i):** The total cost the consumer is expected to spend on this task: $c_i = \sum_{(x,y) \in \alpha(t_i)} (1 - T_i(y)) \cdot c(x)$.

Next, these parameters are aggregated to obtain the overall estimated net profit of the allocation. This process is similar to the one described in [3] with the notable difference that task duration variance is now taken into account to give a more accurate estimate of the duration distribution. To this end, we first calculate the overall success probability of the workflow as the product of all individual task success probabilities: $p = \prod_{\{i \mid t_i \in T\}} p_i$.

The estimated expected cost for the workflow is the sum of the expected cost of all tasks, each multiplied by the probability that it is reached (i.e., that all its predecessors are successful): $\tilde{c} =$

[1]In our work, we set this to 10 iterations.

[2]This is the first time step at which the consumer receives no more reward from the workflow.

$\sum_{\{i \mid t_i \in T\}} r_i c_i$, where $r_i = 1$ if t_i has no predecessor tasks, and $r_i = \prod_{\{j|(t_j \mapsto t_i) \in E\}} p_j$ otherwise.

Finally, to estimate the duration distribution of the workflow, we employ a technique commonly used in operations research [2], and evaluate only the critical path of the workflow (i.e., the path with the highest sum of all mean task durations along it). Using the central limit theorem, we then approximate d_W with a normal distribution that has a mean λ_W equal to the sum of all mean task durations along the path and variance v_W equal to the sum of the respective task variances.

This allows us to solve Equation 1. In order to write it in closed form, we let $D_W(x) = \int_{-\infty}^{x} d_W(y)\,\mathrm{d}y$ be the cumulative probability function of $d_W(x)$, we let $D_d = D_W(d)$ be the probability that the workflow will finish no later than the deadline d and $D_l = D_W(t_0) - D_W(d)$ the probability that the workflow will finish after the deadline but no later than time $t_0 = u_{\max}/\delta + d$ (both conditional on overall success), and write:

$$\tilde{u} = p \cdot (D_d \cdot u_{\max} + D_l \cdot (u_{\max} - \delta \cdot (t_{\text{late}} - d))) - \tilde{c} \quad (2)$$

$$\text{where } t_{\text{late}} = \lambda_W + (e^{\frac{-(d-\lambda_W)^2}{2v_W}} - e^{\frac{-(t_0-\lambda_W)^2}{2v_W}})\frac{\sqrt{v_W}}{D_l \cdot \sqrt{2\pi}} \quad (3)$$

5. EMPIRICAL EVALUATION

To evaluate our strategies, we test their performance empirically by simulating a service-oriented system. More specifically, we first generate a workflow consisting of 10 tasks by randomly populating an adjacency matrix until 25% of all possible edges have been added, and we set $u_{\max} = 1000$, $d = 100$ and $\delta = 50$. Then, we associate each task with one of five different service types, each of which contains a number of service populations drawn from the discrete uniform distribution $U_d(3, 10)$. We populate each of these populations with a number of providers drawn from $U_c(1, 100)$. Then, to generate random performance characteristics, we attach an average cost to each service type, drawn from the continuous uniform distribution $U_c(1, 10)$, as well as a gamma distribution with a shape from $U_c(1, 10)$ and scale from $U_c(1, 5)$ (as the duration distribution). We then introduce further variance to the costs and duration distributions of different populations by randomly varying these based on the type-specific average values[3].

The main variable we control during our experiments is the average failure probability of providers, which allows us to test the performance of our strategies in environments where providers display varying levels of unreliability. Again, we vary this randomly between different service types and populations. To obtain statistical significance and to cover a large set of randomly generated environments, we repeat all experiments for a particular setup 1000 times. Hence, all results are given with 95% confidence intervals and we have carried out hypothesis tests where appropriate (we use two-sample t-tests and ANOVA at the 99% confidence level).

Figure 2 shows the performance of our strategies as we vary the average failure probability. Here, it is clear that the *naïve* strategy performs very poorly. Its net profit is less than 50% of the *flexible* strategy even when service providers never fail (this is because they still take uncertain amounts of time to complete their services). As soon as providers begin to fail, the net profit drops quickly, and the *naïve* strategy makes an overall loss when the failure probability is only 0.3 or higher. Averaged over the failure probabilities, this strategy achieves a total net profit of only 48.33 ± 4.24.

The figure also shows the *fixed(5,25)* strategy as a representative of the general *fixed(n,w)* strategy (we chose these parameters, because they yield the best overall performance). The results of

[3]Due to space restrictions, we omit the full details here.

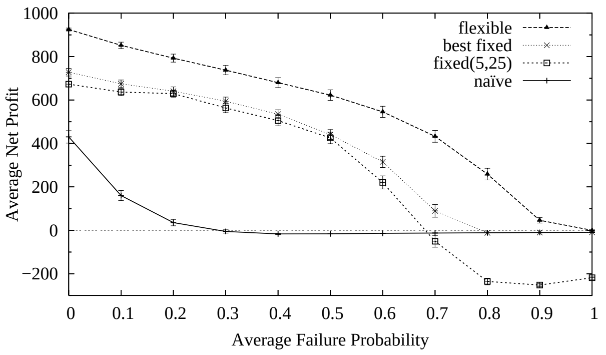

Figure 2: Performance of our heuristic strategies.

this strategy are promising, and clearly indicate that simple service redundancy can achieve a good performance. However, it is also clear that the strategy is not well suited for the whole range of environments tested here. When the average failure probability reaches 0.7, *fixed(5,25)* begins to make an overall loss (as the strategy makes a large investment and still fails to complete the workflow). To give an upper bound for the performance achievable by any *fixed(n,w)* strategy, the hypothetical *best fixed* strategy here demonstrates the best results we could obtain by varying the parameters n and w for each given failure probability. Overall, the *fixed(5,25)* and *best fixed* strategies achieve an average net profit of 263.4 ± 9.1 and 362.49 ± 7.74, respectively.

Finally, the figure shows the results of the *flexible* strategy. These are promising and demonstrate that the strategy outperforms all other strategies. Even at high failure probabilities, the strategy is able to achieve good results and continues to make a net profit when the average failure probability in the system is 0.9. It is also significantly better than the *best fixed*, because it is able to flexibly adapt its provisioning allocation according to the performance characteristics of each task. Moreover, the *flexible* strategy avoids making an overall net loss, because its prediction mechanism allows it to ignore infeasible workflows. Averaged over all failure probabilities, the *flexible* strategy achieves a net profit of 535.66 ± 8.29.

6. CONCLUSIONS

In this paper, we have presented a flexible strategy that provisions multiple heterogeneous providers for the tasks of a workflow, and we have shown that it outperforms the standard naïve approach and various strategies that use fixed redundancy. As the strategy provisions providers automatically, using some performance information about providers, it is suitable for building autonomous software agents that execute workflows in large service-oriented systems, such as computational Grids or the Web.

7. ACKNOWLEDGEMENTS

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

8. REFERENCES

[1] J. N. Hagstrom. Computational complexity of PERT problems. *Networks*, 18:139–147, 1988.

[2] 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.

[3] S. Stein, N. R. Jennings, and T. R. Payne. Flexible provisioning of service workflows. In *Proc. 17th European Conf. on AI (ECAI-06), Riva del Garda, Italy*, pages 295–299. IOS Press, 2006.