Flexible Selection of Heterogeneous and Unreliable Services in Large-Scale Grids

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

School of Electronics and Computer Science, University of Southampton, Southampton, SO17 1BJ, UK.
{ss2,trp,nrj}@ecs.soton.ac.uk

As Grids become larger and more interconnected in nature, scientists can benefit from a growing number of distributed services that may be invoked on demand to complete complex computational workflows. However, it also means that these scientists become dependent on the cooperation of third-party service providers, whose behaviour may be uncertain, failure-prone and highly heterogeneous. To address this, we have developed a novel decision-theoretic algorithm that automatically selects appropriate services for the tasks of an abstract workflow and deals with failures through redundancy and dynamic re-invocation of functionally equivalent services. In this article, we summarise our approach, describe in detail how it can be applied to a real-world bioinformatics workflow and show that it offers a significant improvement over current service selection techniques.

Keywords: Service selection, Grids, reliability, service-oriented computing

1. Introduction

Grid computing is a key enabling technology for the field of e-Science. As scientists rely increasingly on computational support for their work, Grids allow research organisations to pool their resources, spread the cost of expensive hardware and utilise idle machines (Coveney, 2005). However, as Grids become larger, more open and distributed, new challenges need to be addressed (Foster et al., 2004). In particular, the behaviour of services may be highly uncertain, as manifested by frequent failures and execution delays. This uncertainty may be due to network delays, power cuts, competition for resources or machine failures. Moreover, services will increasingly be owned by agents that act autonomously and cannot be assumed to always behave cooperatively or deterministically (Jennings, 2001). Rather, such agents will follow their own decision-making procedures and may even defect when this is beneficial. For example, the owner of a computational cluster may decide to temporarily suspend all jobs in its queue in order to run an important internal job.

Clearly, this uncertainty must be addressed, and this is particularly important when consumers execute large workflows of interdependent tasks, when service providers demand remuneration and when workflows have deadlines. To this end, we concentrate on the process of deciding which service instances to select for the constituent tasks of an abstract workflow in an automated fashion. This allows the consumer to consider non-functional quality-of-service (QoS) parameters (such as the cost or reliability) of functionally equivalent services and to react to failures by dynamically selecting alternate services when previous ones have failed.

Current work does not address the selection problem satisfactorily. Typically, QoS parameters are only considered locally for each task, either to meet user-imposed quality constraints (Thain & Livny, 2003), or to optimise some of the parameters, e.g., to select the cheapest or most reliable service (Vazhkudai et al., 2001). Other work optimises a weighted sum of global QoS parameters, such as the workflow cost and reliability (Zeng et al., 2004). However, it is often unclear how weights are set and both the local and global approaches select only a single service for each task, thus producing brittle workflows that are vulnerable to failures.

A number of existing Grid technologies do consider service failures explicitly. In particular, failures are sometimes handled by submitting the failed task to a different service provider (Thain & Livny, 2003; Oinn et al., 2004). In other systems, fixed redundancy is employed to increase the probability of success (Anderson et al., 2002). However, neither of these approaches explicitly models the impact of failures on the overall workflow (e.g, whether failures will result in missing the overall deadline and whether the cost of redundancy outweighs its benefit).

To address these shortcomings, we have previously developed a decision-theoretic service selection algorithm (Stein et al., 2007). This reasons explicitly about the value of choosing particular service instances and balances the overall workflow cost with its expected reward. Furthermore, the algorithm uses both service redundancy and dynamic re-selection, but applies them in a principled manner based on their impact on the expected performance of the workflow. While our previous work concentrated on the abstract description of the algorithm, in this article, we examine its applicability to e-science by considering a bioinformatics workflow. Furthermore, we present novel experimental results that show an average 46.8% improvement in utility over existing selection approaches.

The remainder of this article is organised as follows. In section 2, we provide a high-level description of our algorithm. Then, in section 3, we show how it can be applied to a real Grid problem. In section 4, we evaluate our work experimentally and conclude in section 5.

2. Flexible Service Selection

In this section, we summarise our assumptions about how services are provided and consumed in a Grid system and we provide a high-level description of our selection algorithm. The full technical details of our approach and associated assumptions can be found in our previous work (Stein et al., 2007).

Throughout our work, we consider a generic Grid system that is based on service-oriented technologies (Foster et al., 2002). Here, providers advertise their services with a registry (which could be potentially federated), which can then be discovered and invoked by consumers when required. Specifically, we assume that there is no centralised resource or workload manager, but rather that each consumer is responsible for selecting and communicating with the service providers directly. We believe that such a view is more appropriate for open, large-scale, highly heterogeneous systems, where a centralised solution would represent a potentially unacceptable communication and performance bottleneck, create a single point of failure and significantly decrease the autonomy of both consumers and providers.

Figure 1 shows the typical activities of a service consumer in our model, which consist of four steps. First, during *workflow selection*, a consumer selects an appropriate workflow to meet its current high-level goal. Such a workflow describes

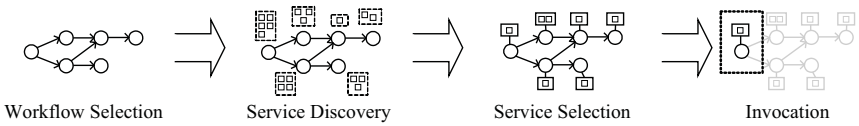


Figure 1. Activities of a service consumer.

the types of services required by the consumer and related ordering constraints (for an example, see section 3). In practice, it may be chosen from a repository of workflows, automatically synthesised by a planner or submitted by a human user. In more detail, we assume that a workflow is a directed acyclic graph, where nodes represent *tasks*, each mapped to a specific *service type*, and edges represent ordering constraints. Furthermore, we assume that a consumer derives a time-dependent utility value when completing the workflow (or none if a fixed deadline is reached).

Given such an abstract workflow, the consumer next engages in a *service discovery* phase. Here, it discovers candidate services by searching a public registry or by contacting an appropriate service broker. This discovery step uses the service types provided by the workflow, mapping each type to a set of suitable services.

This is followed by *service selection*, which constitutes the main focus of this article. Here, the consumer makes predictions about the performance of the workflow and explores the space of service allocations. In doing so, we assume that the consumer has access to some QoS information about the matching services:

- **Cost**: The invocation cost, expressed in the same units as the workflow utility. This is paid on invocation and regardless of the eventual outcome.
- **Failure probability**: The probability that it will fail to complete a task. In this case, a service is not generally assumed to send explicit failure messages.
- **Duration function**: A probability distribution for the time between invoking the service and receiving notification of a successful outcome (as observed by the consumer and including any data transmission or queueing delays).

As described in section 1, we take a decision-theoretic approach and are therefore interested in selecting services that maximise the consumer's expected utility, as given by the difference between the reward of completing the workflow and the overall cost incurred (we assume that the consumer is risk-neutral). Formally, the aim of our algorithm is to find α^*:

$$\alpha^* = \operatorname*{argmax}_{\alpha} \mathrm{E}(R(\alpha) - C(\alpha)), \tag{2.1}$$

where α is a service allocation, which maps tasks to selected services, $R(\alpha)$ and $C(\alpha)$ are random variables representing, respectively, the reward when following allocation α and the associated cost, and $\mathrm{E}(\cdot)$ is the expectation operator.

Now, while existing selection approaches typically map each task to a single service, we are interested in introducing redundancy where necessary and in planning for potential failures by selecting and invoking multiple services in series. Hence, we represent the service allocation α as a mapping that associates each task with the selected services for that task and their respective invocation time steps (conditional on the task not having been completed successfully). An example of such an allocation for a single task is shown in figure 2. Here, three cheap and unreliable services are invoked immediately when the task first becomes executable. If they have not been successful after 15 time steps, more services are gradually invoked to ensure that the task is completed eventually.

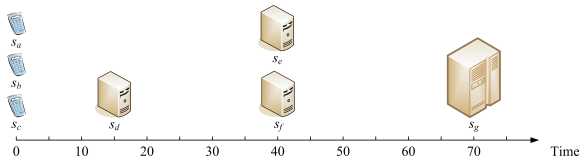


Figure 2. An example service allocation for a single task.

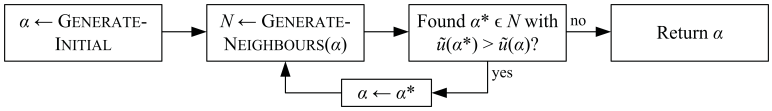


Figure 3. Service selection algorithm.

Unfortunately, the optimisation problem shown in equation 2.1 is NP-hard due to its combinatorial nature. This means that it is generally infeasible to compute a globally optimal solution for larger workflows with many candidate services (as is common in large Grid systems). For this reason, we carry out a local search, as summarised by figure 3. This starts with a random candidate solution and then iteratively improves this until no more improvements can be made. As calculating the expected utility of a given allocation is intractable, we use a heuristic approximation for this ($\tilde{u}(\alpha)$). This first calculates a number of performance parameters for each workflow task (i.e., a success probability, expected cost, mean duration and variance), then aggregates these over the entire workflow and finally uses them to obtain an estimated expected utility. Such a heuristic typically converges quickly to a good solution, but is not guaranteed to find the global optimum α^*.

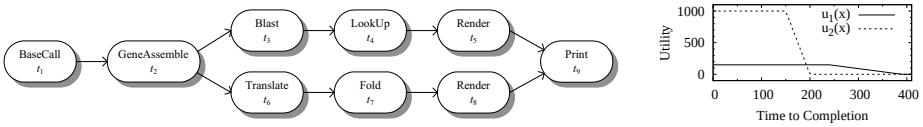
In the following, we illustrate our approach using a real-world Grid workflow.

3. Illustrative Example

To demonstrate how our work can be employed in practice, we use a simple workflow from the bioinformatics domain — an area that relies heavily on computationally intensive services and that has increasingly seen the establishment of large distributed Grid systems for sharing resources (Oinn et al. (2004)). In our example, a scientist has just sequenced a previously unknown gene of a bacterium, and is now interested in visualising the shape of the associated protein. For this, she has to carry out a number of tasks, which are shown as a workflow in figure 4(a).

To summarise this briefly†, the scientist first uses a base-calling service (*BaseCall*) to annotate DNA chromatograms and attach quality values to each base. These are then assembled into a single continuous DNA sequence by a sequence-assembling service (*GeneAssemble*), which also identifies the coding region of the gene. Next, a translation service is used to derive a corresponding amino acid sequence (*Translate*), and a particularly computationally extensive folding service (*Fold*) predicts the 3-dimensional shape of the coded protein. This is then rendered in high resolution using a graphics service (*Render*). In parallel with the folding service, the scientist is also interested in comparing the gene to known sequences. To this end, she searches through public collections of proteins to find the closest match using a specialised service (*Blast*), and then accesses database services to retrieve structural information about this protein (*LookUp*). This is rendered again, and both images are printed as part of a report on a local printer (*Print*).

† Full details can be found in (Stein, 2008).



(a) Bioinformatics workflow. (b) Utility functions.

Figure 4. Illustrative example.

Service Type	**ID**	**Fail. Pr.**	**Cost** (£)	**Num.**	**Duration**	**Mean (min)**	**Var. (min^2)**
BaseCall	P_0	0.2	1	20	Gamma(1.5,2)	3	6
(t_1)	P_1	0.1	3	10	Gamma(1,2)	2	4
	P_2	0.1	1	1	Gamma(1,2)	2	4
GeneAssemble	P_3	0.1	5	25	Gamma(5,2)	10	20
(t_2)	P_4	0	10	1	Gamma(5,2)	10	20
	P_5	0.3	1	10	Gamma(10,2)	20	40
Blast	P_6	0.3	2	50	Gamma(5,3)	15	45
(t_3)	P_7	0.8	0.1	50	Gamma(10,10)	100	1000
	P_8	0.05	10	5	Gamma(2,1)	2	2
LookUp	P_9	0.5	5	10	Gamma(1.5,1.5)	2.25	3.375
(t_4)	P_{10}	0.5	4	2	Gamma(1.5,1.5)	2.25	3.375
	P_{11}	0.75	5	10	Gamma(0.5,0.5)	0.25	0.125
Render	P_{12}	0.1	10	25	Gamma(30,3)	90	270
(t_5, t_8)	P_{13}	0.01	100	5	Gamma(20,2)	40	80
	P_{14}	0.9	1	25	Gamma(30,3)	90	270
Translate	P_{15}	0.7	0.5	50	Gamma(1,1)	1	1
(t_6)	P_{16}	0.7	0.1	50	Gamma(10,2)	20	40
	P_{17}	0	25	10	Gamma(1,1)	1	1
Fold	P_{18}	0.2	10	5	Gamma(3,30)	90	2700
(t_7)	P_{19}	0.05	50	1	Gamma(3,5)	15	75
	P_{20}	0.75	1	1	Gamma(50,2)	100	200
Print	P_{21}	0.2	2	20	Gamma(2,3)	6	18
(t_9)	P_{22}	0.05	2	10	Gamma(5,5)	25	125
	P_{23}	0.9	0.1	30	Gamma(2,3)	6	18

Table 1. *Service types used in the example workflow.*

Now, some service types in this example require a significant computational effort and may take a considerable amount of time to complete. In this context, the workflow utility function allows the scientist to succinctly encode the overall value of the workflow and how this relates to the time taken. For example, if the scientist needs the results later in the day, but is not overly concerned about waiting a bit longer, a utility function that decreases slowly over time, such as u_1 in figure 4(b), is appropriate. If, on the other hand, the results are critical for a presentation she is giving to a funding committee just hours later, a utility function such as u_2 expresses the urgency and high value of the workflow more suitably.

To illustrate how our approach deals with many heterogeneous services, we assume that each workflow task has associated with it a number of services, as shown in table 1. Here, each row of the table describes a group of services that share the same performance characteristics. For example, the population P_0 in the first row represents 20 services that each cost £1, fail 20% of the time and follow a gamma distribution with shape $k = 1.5$ and scale $\theta = 2$. Generally, we have chosen these service populations to offer certain trade-offs — e.g., services in P_1 are more reliable and faster than those in P_0, but also three times as expensive, while services in P_{22} are slower, but more reliable than those in P_{21}.

To demonstrate the types of decisions our algorithm makes, figure 5 shows the final service allocation that is found by local search when the less urgent utility function u_1 is used (this rewards a constant utility of 150 if the workflow is completed

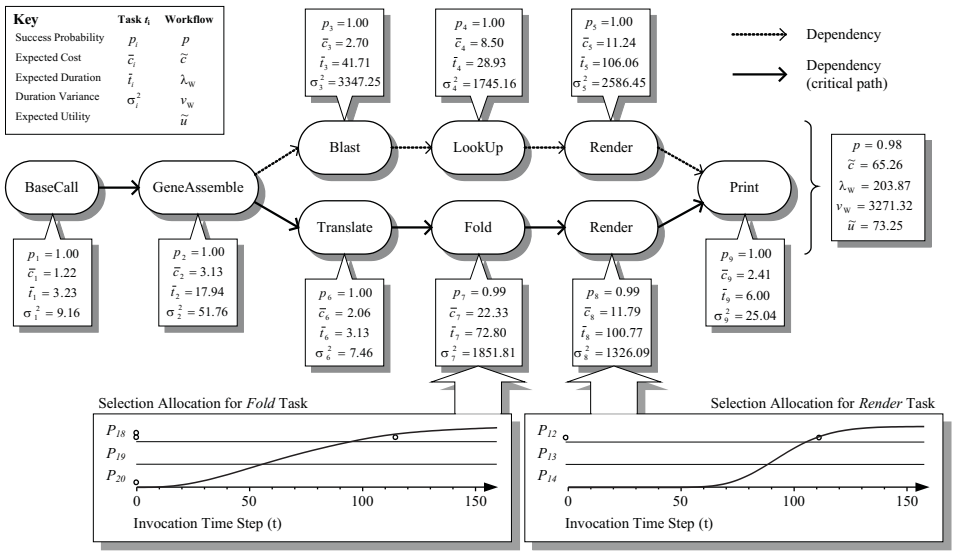


Figure 5. Final service allocation when using u_1 (with detailed allocations for t_7 and t_8).

within four hours, then deducts 1 from this for every minute that the workflow is late, until no more utility is gained after 390 minutes). Here, services are selected redundantly for each task, such that their respective success probabilities are between 0.99 and 1. However, the algorithm generally chooses cheaper and slower services where possible, resulting in low expected costs and large mean durations and variances for each workflow task. Two example task allocations are shown here for the *Fold* and the lower *Render* tasks. For the former, the algorithm initially chooses two services from P_{18} and one from P_{20}. Although each of these has a non-negligible failure probability, their parallel invocation, followed some time later by another service from P_{18}, results in an overall success probability of 0.99. However, despite the redundancy, the services are slow and the overall completion time has a high variance (as shown by the cumulative success probability of the task over time, which is plotted along with the allocation). Due to the lower failure probability for P_{12}, the algorithm proceeds more cautiously for the *Render* task, invoking a single service only, followed almost two hours later by a second one.

Given the performance characteristics for each task, these are aggregated to yield an overall success probability for the workflow (0.98) and an expected cost (65.26). To estimate the overall duration of the workflow, we sum all mean durations and variances along the critical path of the workflow (i.e., the longest path when considering the mean duration of each task), and then use a normal distribution d_W with the resulting summed mean and variance to approximate the completion time. This allows us to estimate the overall expected utility as $\tilde{u} = 0.98 \cdot \int_0^\infty d_W(x)u_1(x)\mathrm{d}x - 65.26 = 73.25$.

Next, figure 6 shows the outcome of the algorithm when the more urgent utility function u_2 is used. Here, it is clear that the strategy adapts appropriately to the changed deadline and higher workflow reward. Generally, the algorithm here relies on higher redundancy, as exemplified by the lower *Render* task, where the consumer now starts with two services from P_{12}, followed just minutes later by a third. In

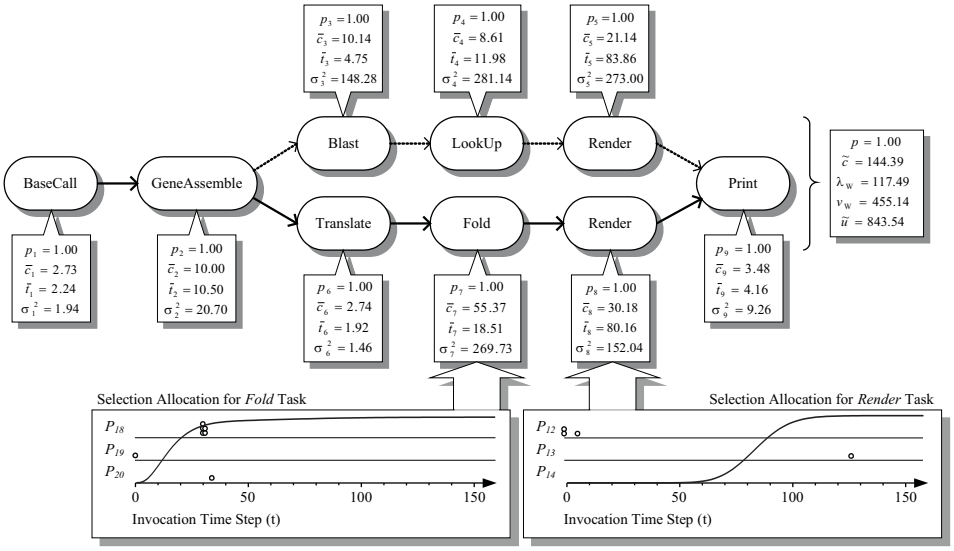


Figure 6. Final service allocation when using u_2 (with detailed allocations for t_7 and t_8).

other cases, the consumer also switches to entirely different populations. For the *Fold* task, it now selects a fast and reliable, but expensive, service from P_{19}, followed by more services from P_{18} and P_{20}. This indicates that the algorithm now proceeds more aggressively, incurring higher service costs, but also significantly reducing the mean workflow duration, so that the overall deadline can be met.

In the following section, we evaluate our approach experimentally.

4. Empirical Evaluation

In order to evaluate the performance of our strategy, we implemented it using the Java programming language and applied it to a range of realistic problems, such as the one described in the previous section. To aid the scientist and visualise the workflow before and during execution, we also developed an interactive user interface, as shown in figure 7. This is divided into three main sections: the top part shows the workflow and its current progress, the middle panel shows global workflow statistics, including the predicted success probability and an estimated completion time distribution, while the lower part gives detailed information about a particular task and its service allocation over time. Moreover, to generalise our results over a range of environments with varying degrees of uncertainty, we decided to conduct an empirical analysis using a simulated Grid system.

In more detail, we first generate twelve random service types and attach an average cost, duration shape and scale (of a gamma distribution) to each, which are drawn from the continuous uniform distributions $U_c(1, 80)$, $U_c(1, 40)$ and $U_c(1, 10)$, respectively. Next, we generate a random number of service populations for each type (drawn from the discrete uniform distribution $U_d(3, 10)$), and add to each a random number of services (drawn from $U_d(1, 200)$). These populations represent groups of services with the same performance characteristics. Then, we vary the characteristics between the populations of each type, based on the type-specific values above. More specifically, we generate the failure probability of a population by sampling from $B(\Phi, 0.005)$, where Φ is the overall average failure probability of

Figure 7. Service selection user interface.

services and $B(m, v)$ is a beta distribution with mean m and variance v. Furthermore, we determine the cost of each service as the product $2 \cdot y \cdot z$, where y is the type-specific average cost and z is sampled from $B(0.5, 0.05)$. This is repeated for the duration parameters, resampling z for each. Finally, workflows always consist of 50 tasks, with precedence constraints that are generated by randomly populating an adjacency matrix until at least 25% of all possible edges are added, ignoring any that would create a cycle. We define the utility function u by awarding a utility of $u_{\max} = 25000$ up to time step $t_{\max} = 2000$ and then deduct $\delta = 37.5$ for each further time step until the utility is 0. Finally, we map each task to a random type.

For each run, we record the net profit our strategy achieves by selecting and then executing a workflow. To obtain statistical significance, we repeat this 250 times with different services and workflows, and carry out ANOVA and two-sample t-tests at the 99% confidence level. We have also repeated our experiments using a large range of controlled variables and obtained the same broad trends as described below. To evaluate our strategy, we compare it to the following benchmarks:

- **local(n,w):** This represents approaches that select services myopically for each task by optimising a weighted sum of QoS parameters. In more detail, we use the technique described by Zeng et al. (2004), weighing the cost, reliability and expected service duration equally†. To include approaches that consider redundancy and service time-outs, the strategy selects the n best services and waits w time-steps until selecting more services.
- **global(w):** This strategy selects a single service for each task, maximising a weighted sum of global performance metrics. Here, we consider the overall workflow cost, reliability and duration, as outlined in Zeng et al. (2004) (weighing all parameters equally). Furthermore, we impose a budget constraint $u_{\max}$ and deadline 2666. To allow the strategy to react to failures, we include a time-out w, after which a new service allocation is computed.

† We have experimented with other weight distributions, but obtained similar or worse results.

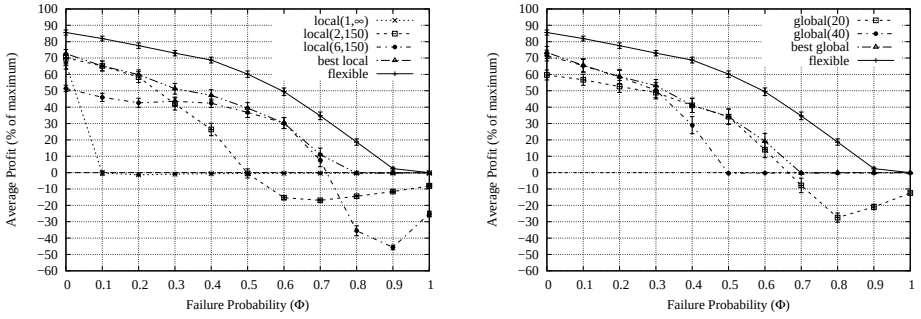
Using this setup, our experimental results are shown in figure 8. In particular, figure 8(a) contains the results of our *flexible* approach and a number of representative *local(n,w)* strategies. The first of these, *local(1,∞)*, represents selection approaches that do not consider failures and so select only the single best service that is available. Clearly, this strategy is unsuitable where there are services failures: at $\Phi = 0.1$, it fails to complete most of its workflows, making an overall loss.

Next, *local(2,150)* and *local(6,150)* are shown as two of the overall best-performing local strategies. These deal with failures by including redundancy and time-outs. For this reason, they perform better than the *local(1,∞)* in most environments and the *local(6,150)* strategy is able to achieve a high positive profit even when the failure probability reaches $\Phi = 0.6$. However, two disadvantages of these strategies are immediately obvious. First, neither of them clearly dominates — at low failure probabilities, *local(2,150)* performs better as it makes a smaller investment by only invoking two services at a time, but when the failure probability rises above $\Phi = 0.3$, *local(6,150)* begins to performs better as higher redundancy is needed to complete the workflow on time. Second, both of them make a significant loss when the failure probability rises to high levels. This is due to their inability to identify infeasible workflows and stop executing when they are likely to miss their deadline or when the cost of repeatedly invoking services begins to outweigh the benefit. To give an indication of the best possible performance achievable by any *local(n,w)* strategy, we also plot an upper bound (a hypothetical *best local* strategy), which was obtained by exhaustively testing a large range of parameters for n and w.

Finally, the figure also contains the average profit of our proposed *flexible* strategy. Clearly, it outperforms all *local(n,w)* strategies. This is due to its ability to automatically vary the levels of redundancy it employs and to select different services, depending on the current environment and the characteristics of its workflow. In some cases, it makes an overall positive profit when no other *local(n,w)* is able to do so. Furthermore, by explicitly predicting the outcome of the workflow, it avoids an overall loss in any of the environments tested here.

Figure 8(b) highlights similar trends for the performance of the *global(w)* strategies. Again, they are able to cope with service uncertainty to some extent, but are also sensitive to the right choice of parameter and make an overall loss when the failure probability rises to a higher level. In this case, the loss is not as severe as that of the *local(n,w)* strategies, due to the overall budget and deadline constraints which eventually cause the consumer to stop executing.

Summarising these results, the *best local* strategy achieves a profit of $8550.74 \pm$



(a) local(n,w) strategy (b) global(w) strategy

Figure 8. Average profit of the examined strategies (all with 95% confidence intervals).

324.79 (averaged over all values for Φ shown in figure 8), the *best global* achieves a profit of 7841.02 ± 365.00, while the *flexible* strategy achieves 12552.12 ± 311.44. Therefore, in the environments tested here, our approach results in an average improvement of 46.8% over current techniques.

5. Conclusions

In this article, we described a novel decision-theoretic service selection algorithm which uses service QoS information to choose between heterogeneous, but functionally equivalent, services and to introduce redundancy when they are particularly failure-prone. Using an example from the bioinformatics domain, we demonstrated how this algorithm can be used by scientists to select uncertain service providers. Finally, by conducting a thorough empirical evaluation over a range of environments, we showed that our approach consistently outperforms current techniques, without requiring manually supplied redundancy and time-out parameters.

Our algorithm is most applicable in large-scale Grid systems where consumers are able to discover and invoke the services of a large number of heterogeneous providers. Hence, it is not directly suitable for closed systems where service selection and invocation is typically performed by centralised middleware, but rather for the open, highly-distributed systems that are supported by emerging service-oriented frameworks, such as the Open Grid Services Architecture and the Semantic Grid (Foster et al., 2002; De Roure et al., 2005). In such systems, our algorithm can execute as an additional decision-making layer on top of existing workflow engines, requiring no additional human input (apart from an appropriate utility function u to encode the value of workflow), and thereby removing the burden of detecting and recovering from most failures from the user.

This work was carried out as part of the Hyperion DIF DTC project.

References

Anderson, D. P., Cobb, J., Korpela, E., Lebofsky, M. & Werthimer, D. 2002 SETI@home: an experiment in public-resource computing. *Communications of the ACM* **45**, 56–61.

Coveney, P. V. 2005 Scientific grid computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* **363**, 1707–1713.

De Roure, D., Jennings, N. R. & Shadbolt, N. R. 2005 The semantic grid: Past, present, and future. *Proceedings of the IEEE* **93**, 669–681.

Foster, I., Jennings, N. R. & Kesselman, C. 2004 Brain meets brawn: Why Grid and agents need each other. In *Proceedings of the 3rd International Conference on Autonomous Agents and Multi-Agent Systems*, pp. 8–15.

Foster, I., Kesselman, C., Nick, J. & Tuecke, S. 2002 The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Tech. rep., Open Grid Service Infrastructure WG.

Jennings, N. R. 2001 An agent-based approach for building complex software systems. *Communications of the ACM* **44**, 35–41.

Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M. R., Wipat, A. & Li, P. 2004 Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics* **20**, 3045–3054.

Stein, S. 2008 *Flexible Service Provisioning in Multi-Agent Systems*. Ph.D. thesis, School of Electronics and Computer Science, University of Southampton.

Stein, S., Jennings, N. R. & Payne, T. R. 2007 Provisioning heterogeneous and unreliable providers for service workflows. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, Vancouver, Canada*, pp. 1452–1458. AAAI Press.

Thain, D. & Livny, M. 2003 *The Grid 2: Blueprint for a New Computing Infrastructure*, chap. Building Reliable Clients and Services, pp. 285–318. San Francisco: Morgan Kaufmann, 2nd edn.

Vazhkudai, S., Tuecke, S. & Foster, I. 2001 Replica selection in the globus data grid. In *Proceedings of the 1st IEEE/ACM International Symposium on Cluster Computing and the Grid*, pp. 106–113.

Zeng, L., Benatallah, B., Ngu, A. H., Dumas, M., Kalagnanam, J. & Chang, H. 2004 QoS-aware middleware for web services composition. *IEEE Trans. Softw. Eng.* **30**, 311–327.

List of Figures

Short Title: Flexible Grid Service Selection