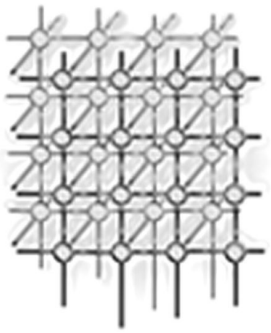
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2009; **00**:1–17 *Prepared using* cpeauth.cls *[Version: 2002/09/19 v2.02]*

Trust-based robust scheduling and runtime adaptation of scientific workflow



Mingzhong Wang[1] *,†, Kotagiri Ramamohanarao[1] and Jinjun Chen[2]

[1] *Department of Computer Science and Software Engineering, The University of Melbourne, Victoria 3010, Australia*
[2] *Faculty of Information and Communication Technologies, Swinburne University of Technology, Victoria 3122, Australia*

SUMMARY

Robustness and reliability with respect to the successful completion of a schedule are crucial requirements for scheduling in scientific workflow management systems because service providers are becoming autonomous. We introduce a model to incorporate trust which indicates the probability that a service agent will comply with its commitments to improve the predictability and stability of the schedule. To deal with exceptions during the execution of a schedule, we adapt and evolve the schedule at runtime by interleaving the processes of evaluating, scheduling, executing and monitoring in the life cycle of the workflow management. Experiments show that schedules maximizing participants' trust are more likely to survive and succeed in open and dynamic environments. The results also prove that the proposed approach of workflow evaluation can find the most robust execution flow efficiently, thus avoiding the need of scheduling every possible execution path in the workflow definition.

KEY WORDS: robustness; trust; scheduling; adaptation

1. INTRODUCTION

Scientific workflow is an important unifying mechanism to support modelling, management and execution of large-scale scientific computing in many complex e-science applications such as climate modelling, astrophysics, medical surgery and disaster recovery [1]. To achieve their goal, domain users define a scientific workflow by composing a large number of computing or data intensive tasks, as well as specifying dependencies between them [2]. For example, large scale scientific computing can usually be achieved by several alternative workflows. As well, for the same task in a workflow, there

*Correspondence to: Department of Computer Science and Software Engineering, The University of Melbourne, Victoria 3010, Australia
†E-mail: minwang@csse.unimelb.edu.au

 Revised 8 February 2009

are several eligible service providers or agents to accomplish its requirements. During execution, the system needs to evaluate eligible workflows and then select the most suitable one (based on certain criteria) for execution. A scheduler which assigns each task in the selected workflow to an appropriate service agent is also required in this framework. Much research has been carried out in scheduling and one can assume that the good schedules can be found using heuristic-based methods, as in general it is an intractable (NP-Hard) problem [3].

Current research on scheduling mainly concerns the trade-off between the execution cost and time of a task under the assumption that all participants are reliable and trustworthy [4]. However, the execution environment of scientific workflows, such as grid or cloud, is dynamic and heterogeneous in nature. In other words, service providers are autonomous agents which may behave dishonestly and sometimes maliciously, especially when pursuing their own interest and benefit [5, 6]. The untrustworthy environment brings uncertainty to the scheduling. That is, the scheduler needs to deal with an uncertain environment. For example, a company may lie about its capability, such as processing time and cost, to attract the attention and contracts. Although a customer may avoid loss when dealing with a certain agent via a carefully designed legal contract, the failure may still ruin other parts of the plan and result in unexpected completion time and/or explosion in cost [7].

Since trust provides a method to measure and minimise the uncertainty associated with interactions in an open distributed system [8], we propose to incorporate it as the third dimension in the scheduling process, in addition to cost and time, therefore improving the reliability and robustness of the schedule.

Usually, selection of the execution flow can be achieved by comparing the best possible schedule for each eligible flow. However, this approach becomes impractical in scientific workflow systems because of the fully dynamic and complex operating environment, and the scale of the workflow itself. Since agents, such as participating nodes in peer-to-peer environments, are no longer 100% trustworthy and can join and leave the system at will, frequent rescheduling may become necessary to reflect those changes. Consequently, although the cost of scheduling an execution flow once can usually be neglected with respect to the overall duration of the flow, it is both time and space consuming to explore and schedule all feasible flows repeatedly, thus degrading system performance. Therefore, we introduce a new evaluation mechanism prior to scheduling to discover the flow which is likely to result in the most robust schedule, thus avoiding (re-)scheduling every possible execution path.

To avoid the need for rescheduling when an exception occurs, we adapt and evolve the existing schedule at runtime by interleaving the processes of scheduling, execution and monitoring in the life cycle of a flow. A set of event-condition-action rules is applied to guide and manage the interaction between the schedule and the environment.

The remainder of this paper is organized as follows. Section 2 introduces the background and motivation. Section 3 gives the formal definition for the optimization criteria and Section 4 describes details incorporating trust into the life cycle of a workflow. The design of experiment and results are presented in Section 5. Section 6 provides an overview of the state of the art. Finally, in Section 7 we present our conclusions and perspective for future work.

2. TRUST IN SCIENTIFIC WORKFLOW MANAGEMENT

In this paper, the workflow management system is called as Coordinator which is in charge of flow selection, task scheduling, runtime monitoring and schedule updating. An important point is that

the Coordinator should not be regarded as a centralized controller because service providers are autonomous to behave on behalf of themselves. For example, they can decide whether to accept requests from the Coordinator or not. They are also free to quit the system in spite of the existence of the Coordinator.

2.1. Trust and workflow execution

A workflow specifies a task network to compose and organize a group of tasks to achieve a well-defined goal, such as weather forecasting or gene alignment in e-science application. The scheduling of the workflow involves discovering resources or services and delegating tasks to suitable service agents to meet users' requirements and constraints. Since the terms of service provider and agent are both commonly used for the executor of a task, we will use them interchangeably in the paper.

The actual execution flow, or plan, is a sub-graph of the workflow definition (when alternative execution paths exist), and can be modelled as a directed acyclic graph (DAG) in which each node represents a task and each arch is a precedent relation between two tasks. Tasks in the graph are delegated to service agents for completion. If the participating agents' execution capacities and costs are guaranteed to remain the same throughout the scheduling and execution of the workflow, the resources of the system are considered static; otherwise they are dynamic. Correspondingly, problems of *static scheduling* and *dynamic scheduling* are recognized respectively.

Research in high performance computing has developed various static scheduling algorithms to help systems optimize execution time and cost to satisfy users' requirements on quality of service (QoS) [9]. However, these methods become restrictive in scientific workflows which are expected to operate in open, dynamic and internetworked information environments [10].

Addressing this realistic application domain, including the execution of scientific workflow in service-oriented or grid environments, this paper focuses on the issue of dynamic scheduling. In fact, there are two fundamental issues making static scheduling infeasible.

- Existing work in workflow scheduling assumes that all participants will adhere to their promises. However, the participating agents may no longer be 100% trustworthy. They may break their commitment when they feel doing so is profitable.
- There is usually more than one feasible execution flow with different needs of resources to achieve the same goal. The selection process of existing work which compares the scheduling result of each alternative flow is both time-consuming and expensive. As resources become dynamic, frequent changes of the environment will force repeated rescheduling on all possible flows, which is expensive and impractical.

Since a service agent may behave differently from its promises, the scheduler can not merely choose the one with the least cost and/or shortest execution time, because the agent may deny the request at run time, thus making the whole schedule obsolete. To address this issue, we propose to integrate the concept of trust in commerce into the scientific workflow management system, thus increasing the reliability and robustness of the schedule in terms of participating agents that are more likely to fulfil their commitments.

To respond to the environment changes at runtime, the processes of scheduling, execution and monitoring are interleaved in the life cycle of an execution flow. When related changes are detected,

the schedule is adapted and evolved accordingly. We also introduce a new evaluation mechanism prior to scheduling to discover the execution flow which could probably result in the most robust schedule, thus avoiding (re-)scheduling on alternative flows as much as possible.

2.2. Trust-aware workflow platform

Each task in the workflow definition needs to be delegated to service agents for completion. However, since individuals run autonomously to pursue and maximise their own interest and utility, the delegator (Coordinator) faces the risk that the delegatee may not behave as it has advertised or agreed. For example, the delegatee may delay the outcome because of low processing efficiency or deliver unsatisfactory results with a lower quality standard. Even worse, the delegatee might not honour the obligation as soon as it gets paid, disregarding the contract with the delegator.

Therefore, trust becomes a fundamental concern in open and dynamic environment. Trust is a measure composed of many different attributes, including reliability, dependability, honesty, truthfulness, competence, and timeliness. In particular, trust is a belief an agent has that the other party will *do what it says it will*, given an opportunity to default to get higher payoffs [8]. In this paper, trust is applied to indicate the probability that a service will be delivered by an agent as it has advertised. Since trust and reputation are closely related concepts, we will use them interchangeably [11].

Many models and mechanisms are proposed to model the trust value of an agent [12]. For the purpose of this paper, a centralized trust management model like eBay [13] is assumed to simplify the discussion. In this case, the trust value of each agent can be obtained by querying the trust manager which models the trust value based on the feedback it receives from the service users. It should be noted that distributed trust models can also be applied in our model to provide agent's trust value.

3. PROBLEM DEFINITION

This paper models a scientific workflow as an AND/OR graph, which is common applied to represent the execution structure of applications. We refine and extend the existing representation of workflows to suit our purpose of dynamic scheduling.

Definition 1. *A* task *is an atomic unit of work from the Coordinator's point of view.*

Workflows are defined by composing individual tasks according to the rules shown in Definition 2, which are in EBNF notation. Three operators are introduced to specify the goal composition: ";" for sequential composition, in which each elements is executed one after the other; "||" for parallel composition, in which all elements are launched together; and "," for selective composition, in which one of the alternative choices is chosen for execution.

Definition 2. *Task composition serves to organize a group of atomic tasks according to their functionalities to build a workflow for a certain goal. Let $\mathcal{T}$ be the set of tasks and $T_i \in \mathcal{T}$.*

$$< atomicTask > ::= \text{“}\epsilon\text{”} \mid \text{“}T_i\text{”} \mid \text{“}start\text{”} \mid \text{“}end\text{”}$$
$$< Workflow > ::= \text{“}start\text{”}\ \text{“;”}\ < Branch >\ \text{“;”}\ \text{“}end\text{”}$$
$$< Branch > ::= < atomicTask > \mid < sequence > \mid < parallel > \mid < selection >$$
$$< sequence > ::= < Branch >\ \text{“;”}\ < Branch >$$
$$< parallel > ::= < Branch >\ \text{“}\|\text{”}\ < Branch >$$
$$< selection > ::= < Branch >\ \text{“,”}\ < Branch >$$

ϵ is a dummy task in the system, which does nothing and always succeeds. It acts as a bridge to connect branches in task composition. *start* and *end* are special ϵ tasks serving as the starting and ending points of the workflow graph respectively.

The selection of a task for execution must enforce the precedent relationship specified by the graph to preserve the application logic. For sequential composition and parallel composition, the Coordinator simply selects both branches together for execution. However, for selection composition, the Coordinator needs to decide which branch to follow, thus making the actual execution path non-deterministic and only known at run time.

However, the Coordinator can decompose the workflow into a set of execution flows or plans, denoted as $\mathcal{P}$, containing all possible execution paths from *start* to *end*, by combining different branches at each selective composition. Each plan, denoted as $p \in \mathcal{P}$, is capable of achieving the goal of the workflow and can be represented as a DAG which specifies the runtime execution flow. Since both task numbers and task types may differ from plan to plan, the resources requirements of each plan will also differ. Let $\mathcal{T}_p$ be the set containing all tasks in p.

Usually, there are several agents with different processing capacities eligible to complete each T_i in the system. Let the set of candidate agents for T_i be denoted as $\mathcal{A}_i$ and $A_{ij} \in \mathcal{A}_i$ be the jth element. For every $T_i \in \mathcal{T}_p$, the Coordinator will choose an agent from $\mathcal{A}_i$ and delegate the work to it. To simplify the discussion and without loss generality we assume that each agent is only designed for a specific task.

To operate in a dynamic environment, the Coordinator should also be able to detect changes and exceptions and then act accordingly. A set of reactive rules, which will be further discussed in the next section, is applied to make the Coordinator environment-aware.

Definition 3. *The Coordinator can be represented as a tuple $CO =< \mathcal{W}, \mathcal{A}, \mathcal{I}, \mathcal{R} >$.*

- *$\mathcal{W}$ represents the graph definition of the workflow, which can be decomposed into a set of plans $\mathcal{P}$. As a result, $\mathcal{T}_p$, the set containing all tasks that appear in plan $p \in \mathcal{P}$, can be derived.*
- *$\mathcal{A}$ is the set of service agents in the system. $\mathcal{A}$ can be further divided into subsets $\mathcal{A}_i$ whose elements are capable of achieving T_i.*
- *$\mathcal{I}$ is the solution to achieve $\mathcal{W}$. It contains two parts: a selected plan $p \in \mathcal{P}$; and a mapping function $\delta : T_i \rightarrow A_{ij}$ which assigns each task $T_i \in \mathcal{T}_p$ to an eligible agent $A_{ij} \in \mathcal{A}_i$.*
- *$\mathcal{R}$ is the set of event-condition-action (ECA) rules to monitor system changes and adapt the schedule accordingly.*

As an example, the AND/OR graph in Fig. 1 represents a simplified travel workflow $\mathcal{W}$. $\mathcal{W}$ can be decomposed into the plan set $\mathcal{P}$ containing four elements:

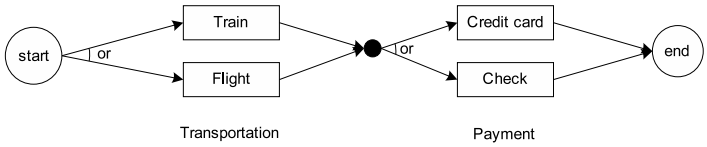


Figure 1. Travel workflow example

- $p_1 : start \rightarrow Train \rightarrow Credit\ card \rightarrow end$,
- $p_2 : start \rightarrow Train \rightarrow Check \rightarrow end$,
- $p_3 : start \rightarrow Flight \rightarrow Credit\ card \rightarrow end$,
- $p_4 : start \rightarrow Flight \rightarrow Check \rightarrow end$.

The solution $\mathcal{I}$ of $\mathcal{W}$ is a plan p_i from $\mathcal{P}$ as well as the service assignment for each task in p_i. Taken p_3 as the solution, $\mathcal{I}$ also indicates the selection of the airline and credit card company.

Since the Coordinator is only concerned about the outcome of an agent for T_i, but not its internal implementation details, each A_{ij} can be abstractly modelled as a tuple representing its processing time, cost, and trust.

Definition 4. *A service agent is a triplet $A_{ij} = < time_{ij}, cost_{ij}, trust_{ij} >$, which means A_{ij} can be trusted with $trust_{ij}$ to complete T_i for the price of $cost_{ij}$ within time $time_{ij}$.*

The aim of the Coordinator is to generate a solution $\mathcal{I}$, which contains a selected plan p from $\mathcal{P}$ and an assignment of every $T_i \in \mathcal{T}_p$ onto a suitable A_{ij} to achieve the multi-objective optimization criteria below. In $\mathcal{I}$, if T_i is assigned to A_{ij}, the completion time of T_i is then denoted as $time(i) = time_{ij}$. As well, $cost(i) = cost_{ij}$ and $trust(i) = trust_{ij}$. CP_p is used to denote the critical path [14] of the plan, which is the path from *start* to *end* with the longest overall duration, and $\mathcal{T}^p_{CP}$ is the set containing all tasks on the critical path.

$$Cost(\mathcal{I}) = \sum_{T_i \in \mathcal{T}_p} cost(i)$$

$$Time(\mathcal{I}) = \sum_{T_i \in \mathcal{T}^p_{CP}} time(i)$$

$$P(Success(\mathcal{I})) = \prod_{T_i \in \mathcal{T}_p} trust(i)$$

Maximize $P(Success(\mathcal{I}))$ subject to:

$$Cost(\mathcal{I}) < B$$

$$Time(\mathcal{I}) < D$$

B is the cost constraint (budget) and D is the time constraint (deadline) required by the user as part of the workflow definition.

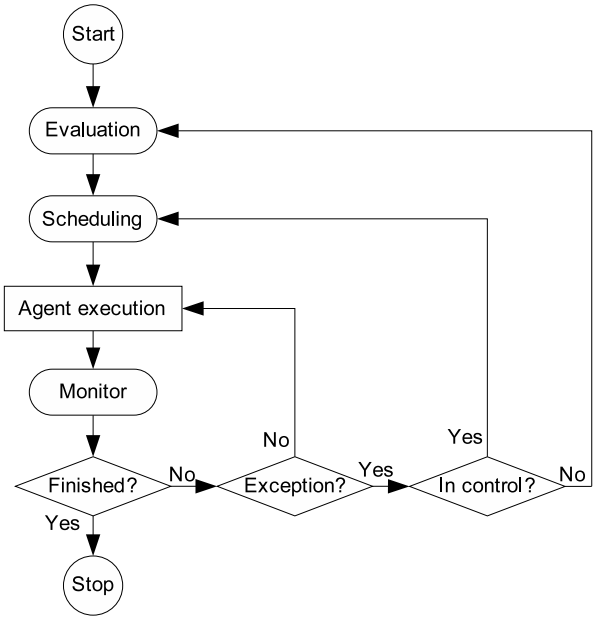


Figure 2. Execution flow of Coordinator

$P(Success(\mathcal{I}))$ indicates the probability that the solution will finally succeed. It is a product of $trust(i)$, which indicates the probability that each task will succeed, because the failure of any task will cause the whole schedule to fail. We assume that each agents' trusts or probabilities of success is independent. A robust schedule has to maximize the solution's probability to succeed.

4. TRUST-BASED SCIENTIFIC WORKFLOW MANAGEMENT

In most practical environments, scheduling is an ongoing, reactive process because of evolving and changing circumstances [15]. Therefore, robustness is considered as an important measurement for a good schedule [16], since higher robustness indicates the schedule is likely to remain valid in the dynamic, changing, and non-deterministic environment, thus reducing the possibility of subsequent rescheduling and making the outcome more predictable.

We propose to incorporate agent trust into plan management, thus improving the robustness and predictability of the system. The inspiration comes from the fact that in economic and social activities the player with higher trust value is more likely to comply with its promises.

Fig. 2 shows the behaviour of the Coordinator, which consists of three interleaved activities in the life cycle of a plan: plan evaluation, scheduling, and execution monitoring.

The evaluation process accepts an AND/OR graph of workflow definition as input and decomposes it into a set of execution plans, each of which is represented as a DAG, by selecting different choice at each selection branch. It then evaluates each plan and chooses the most promising one with respect to the schedule robustness as the input of the next stage. The scheduling stage assigns each task in the selected plan onto a service agent to find an optimal schedule which maximize the robustness of the system while meeting user's deadline and budget constraints. Along with the task execution, Coordinator monitors the environment changes and task result to decide if a reevaluation or rescheduling is necessary. The three scenarios are further explained in the following sections.

4.1. Flow evaluation and selection

To deal with the non-determinism caused by selection branches, some scheduling methods rely on scheduling every possible path combination and then selecting the optimal one. However, since scheduling with cost and time constraints is NP-hard, this approach becomes too costly to deploy in practice. Other methods try scheduling different possible combinations one by one until a satisfactory result is found. However, this usually leads to a poor schedule.

We devise a heuristic plan evaluation method, based on the median values of cost, time and trust of each task, to select the most preferable plan, thus reducing the efforts needed by the scheduler. The motivation comes from the observation that median values are more stable and robust in depicting the sampled feature of the system [17].

Our approach first breaks down the workflow definition into a set of plans, each of which is capable of achieving the coordinator's goal. Then a utility function is applied to each plan to calculate its probability of succeeding within the budget and deadline constraints. Finally, the plan with the highest utility value is selected as solution $\mathcal{I}$.

For a task T_i in plan p, $\widetilde{cost}(i)$, the median cost for achieving T_i, and $MAD(cost(i))$, the median absolute deviation, can be calculated as follows. In the formula, $cost_{ij}$ is assumed to be sorted such that $cost_{i1} \leq cost_{i2} \leq \cdots \leq cost_{im}$ and m is the number of agents in $\mathcal{A}_i$. $MAD(cost(i))$ is the median of $cost_{ij}$'s absolute deviations from the median of $cost_i$. That is, after $\widetilde{cost}(i)$ is computed, the set of each agent's absolute deviation from the median cost can be derived as $\{|cost_{i1} - \widetilde{cost}(i)|, |cost_{i2} - \widetilde{cost}(i)|, \cdots |cost_{im} - \widetilde{cost}(i)|\}$ and $MAD(cost(i))$ is the median value of the derived set.

$$\widetilde{cost}(i) = \begin{cases} \frac{1}{2}(cost_{i\frac{m}{2}} + cost_{i(\frac{m}{2}+1)}) & \text{if } m \bmod 2 = 0 \\ cost_{i\frac{m}{2}} & otherwise \end{cases}$$

$$MAD(cost(i)) = median(|cost_{ij} - \widetilde{cost}(i)|)$$

In the same way, $\widetilde{time}(i)$, $MAD(time(i))$, $\widetilde{trust}(i)$ and $MAD(trust(i))$ can be calculated. Therefore, the median execution cost and time of p are estimated as follows.

$$\widetilde{Cost}(p) = \sum_{T_i \in \mathcal{T}_p} \widetilde{cost}(i)$$

$$\widetilde{Time}(p) = \sum_{T_i \in \mathcal{T}_{CP}^p} \widetilde{time}(i)$$

An estimation function $\eta(i)$ for task T_i is introduced to predict the best possible trust that can be obtained by T_i within the constraints of budget B and deadline D. Generally, $\eta(i)$ tries to trade the spare time and cost of T_i for additional trust. As defined in the function, the allowed cost for T_i can be obtained by distributing B to each T_i according to its share in the total cost of the plan. Then the marginal cost for T_i is derived by deducting $\widetilde{cost}(i)$ from the allowed cost. At last, the cost margin is converted into the measure of trust according to the unit price of changing T_i's trust.

$$
\begin{aligned}
timePrice(i) &= MAD(time(i))/MAD(cost(i)) \\
trustPrice(i) &= MAD(trust(i))/MAD(cost(i)) \\
\eta(i) &= \widetilde{trust}(i) + \Big(\frac{B \cdot \widetilde{cost}(i)}{\widetilde{Cost}(p)} - \widetilde{cost}(i)\Big) \cdot trustPrice(i) \\
&\quad + \Big(\frac{D \cdot (\widetilde{time}(i) + slackable(i))}{\widetilde{Time}(p)} - \widetilde{time}(i)\Big) \cdot \frac{trustPrice(i)}{timePrice(i)}
\end{aligned}
$$

The trade of T_i's time for trust follows the same way except that the allowed slack time for T_i, denoted as $slackable(i)$, should also be considered as part of its allowed time. $slackable(i)$ is derived from T_i's slack time [14], denoted as $slack(i)$, which represents the amount of time that T_i is allowed to be delayed without affecting the whole schedule and is computed while searching for the critical path. However, $slack(i)$ is in fact the total allowed delay for a non-critical path and usually shared by several continuous tasks. For example, let T_i be the parent of T_{i+1} and both of them have slack time greater than 0. If T_i is delayed by t, then the start time of T_{i+1} is also delayed by t. As a result, the allowed delay for T_{i+1} becomes $slack(i+1) - t$. Therefore, we introduce $slackable(i)$, which is the allowed delay for T_i on a per-task basis, to record the result of splitting and distributing $slack(i)$ among involved tasks.

Two special cases are not represented in the formula: (1) If $MAD(trust(i)) = 0$, there is no need to consider the margin because all agents have the same trust; (2) If $MAD(time(i)) = 0$ or $MAD(cost(i)) = 0$, the corresponding margin should not be included. Since the result of $\eta(i)$ is measured in terms of trust, which is the probability that T_i will be fulfilled, its value should be restricted into $[0, 1]$ by defining $\eta(i) = 1$ when the result is greater than 1 and $\eta(i) = 0$ when it is smaller than 0.

Finally, the utility function $\eta(p)$, which heuristically represents the best probability that plan p will succeed, is defined by multiplying the best possible trust of each participating task.

$$\eta(p) = \prod_{T_i \in \mathcal{T}_p} \eta(i)$$

Given a set of eligible plans, the utility function is applied to each of them and the one with the highest value is selected for scheduling and execution.

4.2. Flow scheduling

After plan p is selected in the plan evaluation stage, the Coordinator refines the solution $\mathcal{I}$ to a multi-objective scheduling problem. To find an optimized task assignment for p that maximizes the possibility that the system will eventually succeed, we adopt a genetic algorithm (GA) [18] to design and implement the scheduling process.

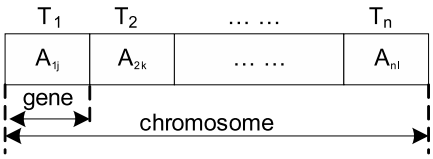


Figure 3. Chromosome encoding for DAG

Typically, a GA contains two main problem-dependent components: the encoding schema and the evaluation function. The encoding scheme breaks a potential solution into discrete parts, that can vary independently. In GA, the discrete parts are called "genes" and the solution is called a chromosome. Since a plan can be modelled as a DAG, a natural encoding for it, as shown in Fig. 3, is a vector of task-agent assignments in which element i, $A_{ij} \in \mathcal{A}_i$, is the agent that T_i is assigned to.

Task dependencies are not encoded into the chromosome. Instead, they are modelled into the calculation of the execution time and are dealt with by the evaluation function.

The evaluation function, denoted as Ψ, measures the quality of a particular solution according to the given optimization objectives. Ψ contains two parts: the fitness functions to encourage the higher robustness of the schedule, and the penalty functions to enforce the constraints of budget B and deadline D. In our definition, the higher the value of Ψ, the better the solution.

In the fitness function $F(\mathcal{I})$, the multiplication of $P(Success(\mathcal{I}))$ and $F_{cost}(\mathcal{I}) + F_{time}(\mathcal{I})$ indicates that the saving on execution time or cost is only valuable if the schedule succeeds. The penalty function $P(\mathcal{I})$ is developed to enforce the constraints. Whenever the cost exceeds the budget or the time goes beyond the deadline, the schedule is not acceptable.

$$
\begin{aligned}
F_{cost}(\mathcal{I}) &= 1 - \frac{Cost(\mathcal{I})}{B} \\
F_{time}(\mathcal{I}) &= 1 - \frac{Time(\mathcal{I})}{D} \\
F(\mathcal{I}) &= P(Success(\mathcal{I})) \cdot (F_{cost}(\mathcal{I}) + F_{time}(\mathcal{I})) \\
P_{budget}(\mathcal{I}) &= \begin{cases} F_{cost}(\mathcal{I}) & \text{if } Cost(\mathcal{I}) >= B \\ 0 & otherwise \end{cases} \\
P_{deadline}(\mathcal{I}) &= \begin{cases} F_{time}(\mathcal{I}) & \text{if } Time(\mathcal{I}) >= D \\ 0 & otherwise \end{cases} \\
P(\mathcal{I}) &= P_{budget}(\mathcal{I}) + P_{deadline}(\mathcal{I})
\end{aligned}
$$

Therefore, Ψ can be defined as,

$$\Psi = F(\mathcal{I}) + P(\mathcal{I})$$

After the chromosome and the evaluation function are defined, standard GA operators can be applied to find the optimized schedule. Processes of selection, crossover and mutation are repeated for a number of times or generations, and then the solution with best evaluation value is chosen as the

$$fail(A_{c0}) \leftarrow canRetry(c) \mid update(\mathcal{I}); findSubstitution(T_c)$$
$$fail(A_{c0}) \leftarrow cannotRetry(c) \mid askUsers()$$
$$unavailable(A_{c0}) \leftarrow true \mid findSubstitution(T_c)$$
$$true \leftarrow isSelectionPoint(T_c) \wedge outOfControl \mid reevaluate(); reschedule()$$

Figure 4. Monitoring rules of Coordinator

schedule. However, none of the evolutionary algorithms can guarantee that the resulting schedule is feasible (satisfying all the constraints) [19]. In practice, a feasible schedule can always be found if the constraints are lessened. Since the purpose of this paper is achieving the robustness and reliability of the schedule by integrating agent trust, the experiments are not performed on extremely constrained cases, therefore all solutions are feasible.

4.3. Flow execution adaptation

Since the environment keeps on evolving, for instance because service agents join and leave the system at will, it is a crucial requirement for the Coordinator to monitor the progress of the schedule execution and to respond reactively in a timely fashion. To achieve this purpose, a set of ECA rules is devised for the Coordinator to manage and adapt the schedule. The rules, as shown in Fig. 2, are represented in the form of *event* $\leftarrow$ *conditions* | *actions*. To simplify the notation, the delegated agent for T_i in the schedule $\mathcal{I}$ is denoted as A_{i0} and T_c is used to denote the current task being monitored by the Coordinator. The meanings of the functions are explained in the rest of the section.

The meaning of the rules can be better comprehended when combined with Fig. 2. During the schedule execution, the Coordinator keeps on monitoring whether there is an exception occurring, which is then treated as an event to trigger further actions. There are two main types of exception the Coordinator needs to consider:

- $fail(A_{c0})$: The execution of T_c failed.
- *unavailable*(A_{c0}): A_{c0} is not available to start T_c. This also includes the situation that the agent modifies its promised execution cost, time, or trust.

Dealing with task failure can be very complicated. Some applications require a recovery process to restore the state of the system. The general approach is to generate a semantic compensation plan to "undo" the failure, and then reevaluate and reschedule the new available execution graph. However, this issue is outside the scope of this paper. For further details, we refer to [20, 21].

However, if the failure can be resolved by reassigning the task to another agent for completion, the Coordinator can modify the schedule by finding a substitute agent, and continue the execution. Agent unavailability can also be dealt with by substitution. Therefore, the utility function $\zeta(j)$ is introduced to measure the soundness of using A_{ij} as a substitute for achieving T_i.

$$\zeta(j) = trust_{ij} \cdot \Big((1 - \frac{cost_{ij} - cost_{i0}}{costMargin(\mathcal{I})}) + (1 - \frac{time_{ij} - time_{i0}}{timeMargin(\mathcal{I}) + slack(i)})\Big)$$

The assessment is based on the cost and time margin between the schedule and system constraints. $costMargin(\mathcal{I})$ represents the surplus between budget B and the schedule $\mathcal{I}$ and $timeMargin(\mathcal{I})$ is the surplus of time from deadline D. $slack(i)$, the slack time of T_i, is added because tasks not on the critical path have an allowed delay without affecting the whole schedule. These three values need to be updated whenever there is a change to the existing schedule.

The function $findSubstitution(T_c)$ applies the utility function to find the most promising substitute agent A_{cj} for A_{c0} to complete T_c. Details are given in Algorithm 1.

In the algorithm, after $\zeta(j)$ is computed for every $A_{ij} \in \mathcal{A}_i$, they are sorted in descending order. The Coordinator then iterates through and selects the first agent which does not violate the system budget and deadline constraints. If none is found, it will simply select the agent with the highest utility, and mark the schedule as *outOfControl*, which means a plan reevaluation is necessary when arriving at the next point of selective composition. At last, the solution $\mathcal{I}$ is updated by replacing A_{i0} with A_{ij}. Consequently, $costMargin(\mathcal{I})$, $timeMargin(\mathcal{I})$ and $slack(i)$ are all updated accordingly.

In case of agent failure, an extra step $update(\mathcal{I})$ needs to be performed before the function $findSubstitution()$, since the failed agent has already consumed time and money. In this case, $update(\mathcal{I})$ at first inserts a dummy task at the place of failure and assigns the exact amount of consumed time and money to it. And then it updates $\mathcal{I}$ to synchronize with all the changes.

After the execution flow is adapted for run time exceptions, it might be necessary to verify that the properties and constraints of the workflow are satisfied as well. The topics on workflow verification and validation are out of scope of this paper; therefore, refer to [22, 23] for details.

The Coordinator can also provide support to control the computational cost of workflow verification and the scheduling itself. It is generally assumed that scheduling cost is comparatively low and can be ignored. However, repeated rescheduling, especially when performed at runtime, has direct impact on the system performance. The solution is to define a threshold for the accumulated scheduling and verification cost for the coordinator. If this value is exceeded, schedule $\mathcal{I}$, including $costMargin(\mathcal{I})$ and $timeMargin(\mathcal{I})$, should be updated to account for the coordinator's cost.

It is possible that during the execution, a better service agent for T_i may appear. In this case, the Coordinator may simply assign T_i to it without considering the existing delegation relationship. However, this may reduce the trust of itself for breaching of contract.

5. EXPERIMENTS AND EVALUATIONS

In this section, experiments on plan scheduling and plan evaluation are performed. To the best of our knowledge, applying trust into the scientific workflow scheduling and execution management to improve the predictability and stability of the schedule has not been addressed in previous research. Therefore, we can only provide the evaluation and comparison between the scheduling strategies with and without utilizing trust.

Input: Schedule I; task T_i which requires a substitute agent
Func findSubstitution(T_i) {
  $costMargin(I) = B - Cost(I)$;
  $timeMargin(I) = D - Time(I)$;
  get $slack(i)$;
  foreach $A_{ij} \in \mathcal{A}_i$ **do**
    compute $\zeta(j)$;
  end
  sort $\zeta(j)$ in descending order ;
  foreach $\zeta(j)$ **do**
    if $(cost_i(j) - cost_i(0) < costMargin(I)) \wedge (time_i(j) - time_i(0) < timeMargin(I) + slack(i))$ **then**
      select A_{ij};
      break;
    end
  end
  if *none is selected* **then**
    select A_{ij} with highest $\zeta(j)$;
    outOfControl = true;
  end
  $\mathcal{I} = \mathcal{I}^j$ which replace A_{i0} by A_{ij};
}

Algorithm 1: Find the feasible substitution agent

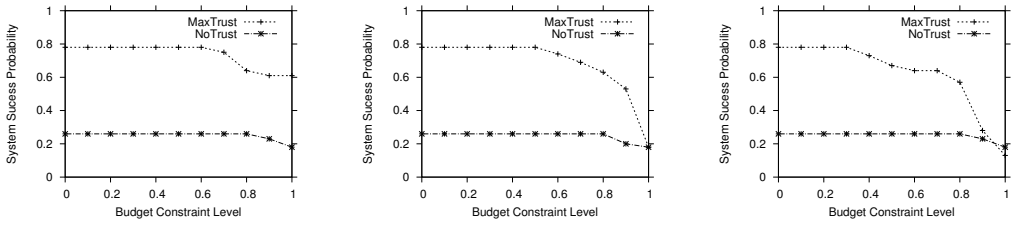
5.1. Experiment design

The environment for the experiments was created by adopting the DAG dataset of the Resource-Constrained Project Scheduling Problem (RCPSP), provided in the Project Scheduling Problem Library (PSPLIB) [24]. The standard sets j30, j60 and j90 consist of 480 DAGs with 30, 60 and 90 non dummy activities, respectively. These graph instances are generated by the generator ProGen [24].

To transform the test sets into our experiments on trust-based scheduling, we have simulated a set of different types of tasks with various cost, time and trust levels (mean values). In simulation, the cost level of each task is randomly selected between 100 and 2000, the time level is between 100 and 5000, and the trust level is set to 1.0. Each node in the graph is then randomly linked with a task type.

Each task is supported by a random number (between 1 and 10) of agent providers with varied processing capabilities. The exact cost, time and trust value for A_{ij} are obtained by randomly fluctuating about 20% around the cost, time and trust level of T_i. To make the simulated environment more practical, the fluctuation follows the observation that in commerce the execution time of an agent is typically inversely proportional to its cost, while its reputation is in direct proportion to the cost.

To simulate different levels of user constraints, the deadline D and the budget B are evaluated at a wide range of possible values between the maximum and minimum allowed time and cost for the plan. k_1 and k_2 in the formulas, ranging from 0 to 1, indicate the constraint levels of D and B respectively, the higher the value, the tighter the constraints.



(a) Deadline Constraint Level = 0.2 (b) Deadline Constraint Level = 0.5 (c) Deadline Constraint Level = 0.8

Figure 5. System success probability with variant constraint levels

$$B = maxCost - k_1(maxCost - minCost)$$
$$D = maxTime - k_2(maxTime - minTime)$$

In the experiments, $maxCost$ and $minCost$ are obtained by always selecting the service agent with the most or least processing cost for each task respectively, while $maxTime$ and $minTime$ are obtained by selecting the slowest/fastest agent for each task.

The implementation of the GA is based on [25]. The population for each generation is set to 50 and the total generation is 1000 for each experiment. All other parameters, such as crossover and mutation probability, are the system default.

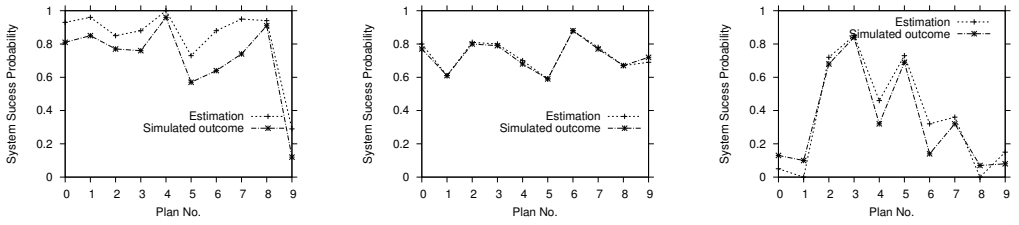
5.2. Result of plan scheduling

The experiment is carried out by setting k_2 at 0.2, 0.5 and 0.8 to represent the relaxed, medium and tight constraint of D respectively, and then increasing k_1 from 0 to 1 by 0.1. Fig. 5 shows the result about the correlation between B, D, and the robustness of the system, in a randomly chosen DAG with 32 nodes. Altering the experiments by changing resource settings and DAG definitions yields similar results to Fig. 5. $NoTrust$ represents scheduling without considering agent trust, while $MaxTrust$ tries to maximize participants' trust value.

The results show that the robustness of the schedule will drop as the constraint levels become tighter since the scheduler needs first to consider the limitation of the cost and time. In order to meet the deadline or budget, the scheduler sometimes has to choose less trustable agents for their cheaper price or shorter processing time. On the contrary, the robustness will keep stable when the constraint levels are low because the scheduler has abundant budget and time to change for services from the most trustable agents. It is also shown that scheduling with the consideration of agent trust will result in much better reliability of the schedule, which is less likely to be revised during execution.

5.3. Result of plan evaluation

Ten testing plans with the same task number are randomly selected from the DAG dataset, and each node in the graph is linked with a random selected task. These plans share the same budget and deadline, which are derived from another randomly selected plan. For each testing plan, the estimated



(a) $k_1 = k_2 = 0.2$ (b) $k_1 = k_2 = 0.5$ (c) $k_1 = k_2 = 0.8$

Figure 6. Comparison of estimation by plan evaluation and simulated outcome by plan scheduling

and simulated outcome of the best success probability within the constraints are computed by plan evaluation and plan scheduling respectively.

Fig. 6 represents the results achieved on 32 node DAGs under different constraint levels. Since task types differ from plan to plan, the resource requirements of each plan will also differ. As a result, the estimated outcome, as well as the simulated outcome, of each plan differs.

It shows that the estimation made by plan evaluation conforms to the simulated outcome computed by plan scheduling. Therefore, the proposed method for plan evaluation can be used as guidance to help find the most robust plan.

In the results, the estimation is closer to the simulated outcome in moderate-to-high constraint levels than in low levels because the conversion of spare time and budget into trust value optimistically assumes that the estimated trust value will be provided by some agents. However, since there are more spare time and budget to be converted in applications with low constrained level, the estimation becomes more likely to exceed the maximum available trust value provided by the agents in the system.

6. SURVEY AND FURTHER DISCUSSION

When scheduling is applied to environments composed of autonomous service providers, the robustness of the schedule, which reflects the probability that the schedule will finally succeed, becomes a crucial requirement. Many researchers have applied redundancy-based approaches to deal with the uncertainty in the environment. For example, [26] apply time slack to provide each task with extra time to execute, thus resulting in more exception-tolerant schedules. [16] use a more complex redundancy measure for robustness with respect to desired system performance features against multiple perturbations in various system and environmental conditions. However, we apply trust to improve the schedule robustness and reliability, thus reducing the probability of rescheduling.

Trust and reputation have been widely recognized as a crucial part of online applications and autonomous systems [8, 12], such as peer-to-peer networks, e-commerce, and online auctions, since they encourage individuals to behave as promised. [27] applied a Markov model to detect fraudsters from the participants based on the reputation system. However, there is no existing work in applying trust into plan management and plan scheduling in scientific workflow environments systematically.

The runtime execution monitoring of our Coordinator is similar to the concept of reactive scheduling [15, 28] which revises or reoptimizes the schedule when an unexpected event occurs in

dynamic environments. However, compared with their progressive trade-off between time and cost, our approach integrates a third parameter of agent trust to help improve the schedule robustness and reliability, thus reducing the probability of rescheduling.

7. CONCLUSION AND FUTURE WORK

In this paper, we incorporate trust to manage the life cycle of a scientific workflow, thus improving the robustness and predictability of the overall system. The approach, which is controlled by the Coordinator, can be classified into three main scenarios: plan evaluation, scheduling, and monitoring. During the evaluation process, the workflow definition of the system is first decomposed into a set of plans, each of which is capable of achieving the workflow goal. Then each plan is evaluated to find the best probability it can succeed by applying the median values of cost, time and trust. Finally, the plan with the highest value is selected for scheduling.

The scheduling stage applies a genetic algorithm to find the optimized assignment of each task in the plan to an autonomous service agent, thereby maximizing the robustness of the schedule while meeting the constraints of deadline and budget. During the schedule execution, the Coordinator monitors for any exceptions that occur. If one is detected, the Coordinator will actively respond by following the exception handling rules, which may result in task reassignment, plan rescheduling, or even reevaluation.

The experiments show that integrating trust into workflow management can greatly improve system robustness and reliability, especially when the constraint level is not very tight, meaning that the resulting schedule is more likely to succeed. The results also verify that the proposed plan evaluation method conforms to the scheduling result, thus being helpful for plan selection.

For future work, we intend to apply the auction model as the resource discovery and pricing mechanism. We want to investigate the relationship between the auction, trust, and scheduling. Another interesting part of future work is to see how our approach and redundancy-based methods can be combined to improve system robustness. Finally, fraud detection methods [27] will be further studied and integrated into our model to reduce the effects of malicious agents in scheduling.

ACKNOWLEDGEMENTS

A preliminary version of this paper appeared in the 2008 IEEE/WIC/ACM International Conference on Intelligent Agent Technology.

REFERENCES

1. Liu X, Dou W, Chen J, Fan S, Cheung SC, Cai S. On design, verification, and dynamic modification of the problem-based scientific workflow model. *Simulation Modeling Practice and Theory* 2007; **15**(9):1068–1088.
2. Yang C, Chen J. A datastream view for scientific workflow. *Grid Computing: Infrastructure, Services, and Applications*, Wang L (ed.). CRC Press: USA, 2009.
3. Ullman JD. Np-complete scheduling problems. *Journal of Computer and System Sciences* 1975; **10**(3):384–393.
4. Yu J, Buyya R. Workflow scheduling algorithms for grid computing. *Technical Report GRIDS-TR-2007-10*, University of Melbourne, Australia 2007.

5. Cheung WK, Liu J, Tsang KH, Wong RK. Towards autonomous service composition in a grid environment. *ICWS '04: Proceedings of the IEEE International Conference on Web Services*, IEEE Computer Society: Washington, DC, USA, 2004; 550, doi:http://dx.doi.org/10.1109/ICWS.2004.114.
6. Fox G, Zhuge H. Special issue: Autonomous grid computing. *Concurrency and Computation: Practice and Experience* 2007; **19**(7):943–944.
7. Babanov A, Collins J, Gini M. Asking the right question: Risk and expectation in multiagent contracting. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 2003; **17**(3):173–186, doi: http://dx.doi.org/10.1017/S0890060403173027.
8. Ramchurn SD, Huynh D, Jennings NR. Trust in multi-agent systems. *Knowl. Eng. Rev.* 2004; **19**(1):1–25, doi: http://dx.doi.org/10.1017/S0269888904000116.
9. Yu J, Buyya R. A taxonomy of scientific workflow systems for grid computing. *SIGMOD Rec.* 2005; **34**(3):44–49, doi: http://doi.acm.org/10.1145/1084805.1084814.
10. Zhao Z, Belloum A, Sloot P, Hertzberger B. Agent technology and scientific workflow management in an e-science environment. *ICTAI '05: Proceedings of the 17th IEEE International Conference on Tools with Artificial Intelligence*, IEEE Computer Society: Washington, DC, USA, 2005; 19–23.
11. Chang E, Dillon TS, Hussain FK. Keynote 2: Trust and reputation relationships in service-oriented environments. *Third International Conference on Information Technology and Applications (ICITA 2005), 4-7 July 2005, Sydney, Australia*, IEEE Computer Society, 2005; 4–14.
12. Jøsang A, Ismail R, Boyd C. A survey of trust and reputation systems for online service provision. *Decis. Support Syst.* 2007; **43**(2):618–644, doi:http://dx.doi.org/10.1016/j.dss.2005.05.019.
13. Boyd J. In community we trust: Online security communication at eBay. *J. Computer-Mediated Communication* 2002; **7**(3).
14. Kelley JJE. Critical-path planning and scheduling: Mathematical basis. *Operations Research* 1961; **9**(3):296–320. URL `http://www.jstor.org/stable/167563`.
15. Smith S. Reactive scheduling systems. *Intelligent Scheduling Systems*, Brown D, Scherer W (eds.). Kluwer Press, 1995.
16. Ali S, Maciejewski AA, Siegel HJ, Kim JK. Measuring the robustness of a resource allocation. *IEEE Trans. Parallel Distrib. Syst.* 2004; **15**(7):630–641.
17. Huber P. *Robust Statistics*. Wiley: New York, 1974.
18. Deb K ((ed.)). *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley & Sons, 2001.
19. Wall MB. A genetic algorithm for resource-constrained scheduling. PhD Thesis 1996. Supervisor-Mark Jakiela and Supervisor-Woodie C. Flowers.
20. Unruh A, Bailey J, Ramamohanarao K. A logging-based approach for building more robust multi-agent systems. *IAT '06: Proceedings of the 2006 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, IEEE Computer Society: Washington, DC, USA, 2006; 342–349, doi:http://dx.doi.org/10.1109/IAT.2006.12.
21. Wang M, Ramamohanarao K, Unruh A. Utilizing BDI features for transactional agent execution. *IAT '07: Proceedings of the 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, IEEE Computer Society: Washington, DC, USA, 2007; 215–221, doi:http://dx.doi.org/10.1109/IAT.2007.105.
22. Chen J, Yang Y. Multiple states based temporal consistency for dynamic verification of fixed-time constraints in grid workflow systems. *Concurrency and Computation: Practice and Experience* 2007; **19**(7):965–982, doi: http://dx.doi.org/10.1002/cpe.v19:7.
23. Chen J, Yang Y. A taxonomy of grid workflow verification and validation. *Concurrency and Computation: Practice and Experience* 2008; **20**(4):347–360, doi:http://dx.doi.org/10.1002/cpe.v20:4.
24. Project Scheduling Problem Library - PSPLIB. *http://129.187.106.231/psplib/* last visited 28 September 2008.
25. Meffert K, et al. *http://jgap.sourceforge.net/*. JGAP - Java Genetic Algorithms and Genetic Programming Package last visited 7 July 2008.
26. Davenport AJ, Gefflot C, Beck JC. Slack-based techniques for robust schedules. *Sixth European Conference on Planning*, Toledo, Spain, 2001; 7–18.
27. Zhang B, Zhou Y, Faloutsos C. Toward a comprehensive model in internet auction fraud detection. *Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, IEEE Computer Society: Washington, DC, USA, 2008, doi:http://dx.doi.org/10.1109/HICSS.2008.455.
28. Dorn J. Evaluating reactive scheduling systems. *IAT '04: Proceedings of the 2004 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, IEEE Computer Society: Washington, DC, USA, 2004; 458–461, doi: http://dx.doi.org/10.1109/IAT.2004.56.