Robust Scheduling and Runtime Adaptation of Multi-agent Plan Execution

Mingzhong Wang      Kotagiri Ramamohanarao      Jinjun Chen
Dept. of Computer Science and Software Engineering  The University of Melbourne  Swinburne University of Technology
The University of Melbourne  Victoria 3010, Australia  Victoria 3122, Australia
{minwang, rao}@csse.unimelb.edu.au  jchen@swin.edu.au

Abstract

Robustness and reliability with respect to the successful completion of a schedule are crucial requirements for scheduling in multi-agent systems because agent autonomy makes execution environments dynamic and non-deterministic. We introduce a model to incorporate trust which indicates the probability that an agent will comply with its commitments into scheduling, thus improving the predictability and stability of the schedule. To deal with exceptions during execution, we adapt and evolve the schedule at runtime by interleaving the processes of evaluation, scheduling, execution and monitoring in the life cycle of a plan. 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 plan evaluation approach conforms with the simulation result, thus being helpful for plan selection.

1. Introduction

Agents need to cooperate to achieve business goals because individuals usually possess limited capabilities and resources. To produce correct and reliable results, execution plans that constrain and define the allowed business scenarios among agents act as a blueprint to guide and organize the autonomous participants.

A goal can usually be achieved by several alternative plans. As well, for the same task in a plan, there are several eligible agents to accomplish its requirements. During execution, the system needs to evaluate eligible plans and then select the most suitable one (based on certain criteria) for execution. A scheduler which assigns each task in the selected plan to an appropriate agent is also required in this framework. Much research has been carried out in scheduling and one can assume that good schedules can be found using heuristic-based methods, as in general it is an intractable (NP-Hard) problem.

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 [16]. However, in multi-agent system, autonomous agents may no longer be 100% trustworthy. They may behave dishonestly and sometimes maliciously, especially when pursuing their own interest and benefit; this is the price of autonomy. The untrustworthy environment brings uncertainty to the scheduler to deal. 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 [2].

Since trust provides a method to measure and minimise the uncertainty associated with interactions in an open distributed system [12], we propose to incorporate it as the third dimension into the scheduling process, in addition to cost and time, therefore improving the reliability and robustness of the schedule in terms of participating agents that are more likely to fulfill their commitments.

Usually, plan selection can be achieved by comparing the best possible schedule for each eligible plan. However, this approach becomes impractical in systems consisting of autonomous agents since frequent changes in the environment will force repeated rescheduling. Consequently, although the cost of scheduling a plan once can usually be neglected with respect to the overall duration of the plan, it is both time and space consuming to explore and schedule all feasible plans repeatedly, thus degrading system performance. Therefore, we introduce a new evaluation mechanism prior to scheduling to discover the plan which is likely to result in the most robust schedule, thus avoiding (re-)scheduling on all alternative plans.

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 plan. 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 gives the formal definition for the optimization crite-
ria and Section 3 describes details incorporating trust into the life cycle of a plan. The design of experiment and results are presented in Section 4. Section 5 provides an overview of the state of the art. Finally, in Section 6 we present our conclusions and perspective for future work.

2. Trust Model and Problem Definition

In this paper, a special Coordinator agent is used to manage and execute the plan. It is in charge of plan selection, task scheduling, runtime monitoring and schedule updating.

2.1. Trust-aware Platform for Agents

In multi-agent systems, an agent usually needs the services of others for completion of a plan because of its limited capacity or resources. 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 multi-agent systems. Trust is a measure composed of many different attributes, including reliability, dependability, honesty, truthfulness, security, 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 [12]. In the 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 use them interchangeably.

Many models and mechanisms are proposed to model the trust value of an agent [8]. For the purpose of this paper, a centralized trust management model like eBay [4] 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 the trust value of an agent.

2.2. Problem Definition

An execution plan, denoted as $p$, specifies a task network to compose and organize a group of tasks to achieve a well-defined goal, such as trip planning or order handling in an enterprise. A plan can be modelled as a directed acyclic graph (DAG) in which each node represents a task and each arc is a precedent relation between two tasks. The scheduling of the plan involves discovering resources or services and delegating tasks to suitable agents to meet users’ requirements and constraints. To simplify the discussion and without loss generality we assume that each agent is only designed for a specific task.

To achieve the goal of the system, the Coordinator is responsible for goal decomposition, plan selection, plan scheduling and plan monitoring.

**Definition 1.** The Coordinator can be represented as a tuple $CO = \langle G, A, I, R \rangle$.

- $G$ is the goal of the system, which can be decomposed into a set of tasks $\mathcal{P}$. As a result, $T_p$, the set containing all tasks that appear in plan $p \in \mathcal{P}$, can be derived.
- $A$ is the set of service agents in the system. $A$ can be divided into subsets $A_i$, whose elements are capable of achieving task $T_i$. Let $A_{ij} \in A_i$ be the $j$th element.
- $I$ is the solution to achieve $G$. It contains two parts: a selected plan $p \in \mathcal{P}$; and a mapping function $\delta : T_i \rightarrow A_{ij}$ for each $T_i \in T_p$.
- $R$ is the set of event-condition-action (ECA) rules to adapt the schedule to changes in the environment.

The ECA rules in $R$, which will be further discussed in the next section, are applied to enable the Coordinator to operate in a dynamic environment.

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 2.** Agent $A_{ij}$ is a triplet $<time_{ij}, cost_{ij}, trust_{ij}>$, 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 $I$, which contains a selected plan $p$ from $\mathcal{P}$ and an assignment of every $T_i \in T_p$ onto a suitable $A_{ij}$ to achieve the multi-objective optimization criteria below. In $I$, 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 [9] of the plan, which is the path with the longest overall duration, and $T_{cp}^p$ is the set containing all tasks on the critical path. $B$ is the cost constraint (budget) and $D$ is the time constraint (deadline) required by the user as part of the goal.

\[
\begin{align*}
Cost(I) &= \sum_{T \in T_p} cost(i) \\
Time(I) &= \sum_{T \in T_{cp}^p} time(i) \\
P(Success(I)) &= \prod_{T \in T_p} trust(i)
\end{align*}
\]

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

\[
\begin{align*}
Cost(I) &< B \\
Time(I) &< D
\end{align*}
\]

$P(\text{Success}(I))$ 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.

3. Trust-based Plan Management

Robustness is considered as an important measurement for a good schedule, and has different definitions [1]. In this research, it is defined as the probability that a schedule will complete successfully.

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. 1 shows the behaviour of the Coordinator, which consists of three interleaved activities: plan evaluation, scheduling, and execution monitoring.

![Figure 1. Execution flow of Coordinator](image)

3.1. Plan Evaluation

To select a more suitable plan, some scheduling methods rely on scheduling every eligible plan 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 plans 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 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 $\overline{cost(i)}$ from the allowed cost. Finally, the cost margin is converted into the measure of trust according to the unit price of changing $T_i$’s trust.

\[
\eta(i) = \overline{trust(i)} + \left( \frac{B \cdot \overline{cost(i)} - \overline{cost(i)} \cdot \overline{trustPrice(i)}}{\overline{Cost(p)}} \right) + \left( \frac{D \cdot (\overline{time(i)} + \overline{allocatedSlack(i)}) - \overline{time(i)} \cdot \overline{trustPrice(i)}}{\overline{Time(p)}} \right)
\]

The trade of $T_i$’s time for trust follows the same way except that the allocated slack time for $T_i$, denoted as $\overline{allocatedSlack(i)}$, should also be considered as part of its allowed time. $\overline{allocatedSlack(i)}$ is derived from $T_i$’s slack time [9], denoted as $\overline{slack(i)}$, which represents the maximum 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, $\overline{slack(i)}$ 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 $\overline{slack(i)}$, 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 \( allocatedSlack(i) \), which is the allocated 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 \( MAD(time(i)) = 0 \) and \( MAD(cost(i)) = 0 \), 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.

### 3.2. Plan Scheduling

After plan \( p \) is selected in the plan evaluation stage, the Coordinator refines the solution \( I \) 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) [6] to design and implement the scheduling process.

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. 2, 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 \( \Psi \), measures the quality of a particular solution according to the given optimization objectives. \( \Psi \) contains two parts: the fitness function, denoted as \( F(I) \), to encourage the higher robustness of the schedule, and the penalty function, denoted as \( P(I) \), to enforce the constraints of budget \( B \) and deadline \( D \). In our definition, the higher the value of \( \Psi \), the better the solution.

\[
F_{\text{cost}}(I) = 1 - \frac{\text{Cost}(I)}{B}
\]

\[
F_{\text{time}}(I) = 1 - \frac{\text{Time}(I)}{D}
\]

\[
F(I) = P(\text{Success}(I)) \cdot (F_{\text{cost}}(I) + F_{\text{time}}(I))
\]

\[
P_{\text{budget}}(I) = \begin{cases} 
F_{\text{cost}}(I) & \text{if } \text{Cost}(I) \geq B \\
0 & \text{otherwise}
\end{cases}
\]

\[
P_{\text{deadline}}(I) = \begin{cases} 
F_{\text{time}}(I) & \text{if } \text{Time}(I) \geq D \\
0 & \text{otherwise}
\end{cases}
\]

\[
P(I) = P_{\text{budget}}(I) + P_{\text{deadline}}(I)
\]

\[
\Psi = F(I) + P(I)
\]

In \( F(I) \), the multiplication of \( P(\text{Success}(I)) \) and \( F_{\text{cost}}(I) + F_{\text{time}}(I) \) indicates that the saving on execution time or cost is only valuable if the schedule succeeds. \( P(I) \) is design to enforce the bias against ineligible schedules which exceed the budget or go beyond the deadline.

After the chromosome and the evaluation function are defined, standard GA operators can be applied to find the optimized schedule. However, none of the evolutionary algorithms can guarantee that the resulting schedule is feasible (satisfying all the constraints) [14]. 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.

### 3.3. Execution Monitoring

Since the environment keeps on evolving, 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. 1, are represented in the form of \( \text{event} \leftarrow \text{condition} \mid \text{action} \) and can easily be translated into an existing agent programming language, such as 3APL or JADE [3]. To simplify the notation, the delegatee of \( T_i \) in the schedule \( I \) is denoted as \( A_{0i} \) and \( T_i \) is used to denote the current task being monitored by the Coordinator.

![Figure 3. Monitoring rules of Coordinator](image-url)
trigger further actions. There are two main types of exception the Coordinator needs to consider: (1) \( \text{fail}(A_{i0}) \) that the execution of \( T_i \) failed. (2) unavailable\( (A_{i0}) \) that \( A_{i0} \) is not available to start \( T_i \). 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 [15].

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 \( T_i \):

\[
\zeta(j) = \text{trust}_{ij} \cdot \left( 1 - \frac{\text{cost}_{ij} - \text{cost}_0}{\text{costMargin}(I)} \right) + \left( 1 - \frac{\text{time}_{ij} - \text{time}_0}{\text{timeMargin}(I) + \text{slack}(i)} \right)
\]

The assessment is based on the cost and time margin between the schedule and system constraints. \( \text{costMargin}(I) \) represents the surplus between budget \( B \) and the schedule \( I \) and \( \text{timeMargin}(I) \) is the surplus of time from deadline \( D \). \( \text{slack}(i) \), 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 \( \text{findSubstitution}(T_i) \) applies the utility function to find the most promising substitute agent \( A_{cj} \) for \( A_{i0} \) to complete \( T_i \). 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 \( \text{outOfControl} \), which means a plan reevaluation is necessary when arriving at the next point with more than one eligible subsequent plan. Finally, the solution \( I \) is updated by replacing \( A_{i0} \) with \( A_{ij} \). Consequently, \( \text{costMargin}(I), \text{timeMargin}(I) \) and \( \text{slack}(i) \) are all updated accordingly.

In case of agent failure, an extra step \( \text{update}(I) \) needs to be performed before the function \( \text{findSubstitution}() \), since the failed agent has already consumed time and money. In this case, \( \text{update}(I) \) 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 \( I \) to synchronize with all the changes.

The Coordinator can also provide support to control the computational cost of 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 cost for the coordinator. If this value is exceeded, schedule \( I \), including \( \text{costMargin}(I) \) and \( \text{timeMargin}(I) \), should be updated to account for the coordinator’s cost.

\begin{verbatim}
Input: Schedule I; task Ti which requires a substitute agent
Func findSubstitution(Ti) {
    costMargin(I) = B - Cost(I);
    timeMargin(I) = D - Time(I);
    get slack(i);
    foreach Aij \in \mathcal{A}_i do
        compute \( \zeta(j) \);
    end
    sort \( \zeta(j) \) in descending order;
    foreach \( \zeta(j) \) do
        if (\( \text{cost}(j) - \text{cost}(0) < \text{costMargin}(I) \))
            \&\&(\( \text{time}(j) - \text{time}(0) < \text{timeMargin}(I) + \text{slack}(i) \))
            then
                select Aij;
                break;
            end
        if none is selected then
            select Aij with highest \( \zeta(j) \);
            outOfControl = true;
        end
    I = I' which replace A_{i0} by A_{ij};
}
\end{verbatim}

Algorithm 1: Find the feasible substitution agent

4. Experiments and Evaluations

In this section, experiments on plan scheduling and plan evaluation are performed. To the best of our knowledge, applying agent trust into the plan 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.

The environment for the experiments was created by adopting the DAG dataset of the Resource-Constrained Project Scheduling Problem, provided in the Project Scheduling Problem Library [11].

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.

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 ob-
At 0 and the deadline \( D \) tries to maximize participants’ trust value.

\[ B = \maxCost - k_1(\maxCost - \minCost) \]
\[ D = \maxTime - k_2(\maxTime - \minTime) \]

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 [10]. The population for each generation is 50 and the total generation is 1000 for each experiment. All other parameters, such as crossover and mutation probability, are the system default.

**Experimental result for 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. 4 shows the result about the correlation between \( B, D, \) and the robustness of the schedule, in a randomly chosen DAG with 32 nodes. Altering the experiments by changing resource settings and DAG definitions yields similar results to Fig. 4. 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. However, it will keep stable when the constraint levels are low. 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.

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

Fig. 5 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.

5. Related Work

When scheduling is applied to environments composed of autonomous agents, 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, [5] provide each task with time slack to execute, thus resulting in more exception-tolerant schedules. [1] 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 agent 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 [12, 8], such as peer-to-peer networks, e-commerce, and online auctions, since they encourage individuals to behave as promised. [17] 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 multi-agent environments systematically.

The runtime execution monitoring of our Coordinator is similar to the concept of reactive scheduling [13] which revises or reoptimizes the schedule when an unexpected event occurs in dynamic environments. However, our use of trust in plan scheduling and schedule updating can reduce the frequency of rescheduling and reoptimization.

6. Conclusion and Future Work

In this paper, we incorporate agent trust to manage the life cycle of an agent plan, thus improving the robustness and predictability of the overall system. The approach can be classified into three main scenarios: plan evaluation, scheduling, and monitoring. During the evaluation process, the system goal is first decomposed into a set of plans, each of which is capable of achieving the 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 plan 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 investigate the relationship between the auction, trust, and scheduling. Another interesting work is to see how our approach and redundancy-based methods can be combined to improve system robustness. Finally, fraud detection methods [17] will be further studied and applied to reduce the effects of malicious agents in scheduling.

Acknowledgement: We would like to express our thanks to the reviewers for their valuable comments. We also thank William E. Webber for his detailed and insightful advices on the paper.

References