Risk-aware Intermediate Dataset Backup Strategy in Cloud-based Data Intensive Workflows

Mingzhong Wang*, Liehuang Zhu, Zijian Zhang

School of Computer Science, Beijing Institute of Technology, Beijing 100081, China

Abstract

Data-intensive workflows are generally computing- and data- intensive with large volume of data generated during their execution. Therefore, some of the data should be saved to avoid the expensive re-execution of tasks in case of exceptions. However, cloud-based data storage services come at some expense. In this paper, we introduce the risk evaluation model tailored for workflow structure to measure and achieve the trade-off between the overhead of backup storage and the cost of data regeneration in failure, making the service selection and execution more efficient and robust. The proposed method computes and compares the potential loss with and without data backup to achieve the trade-off between overhead of intermediate dataset backup and task re-execution after exceptions. We also design the utility function with the model and apply a genetic algorithm to find the optimized schedule. The results show that the robustness of the schedule is increased while the possible risk of failure is minimized, especially when the volume of generated data is not large in comparison with the input.

Keywords: Data-intensive workflow, Risk evaluation, Robustness, Intermediate dataset, Checkpoint

*Corresponding author. Tel.: +86 10 6891 3536.

Email addresses: wangmz@bit.edu.cn (Mingzhong Wang), liehuangz@bit.edu.cn (Liehuang Zhu), zhangzijian@bit.edu.cn (Zijian Zhang)

URL: <http://cs.bit.edu.cn/wangmingzhong> (Mingzhong Wang)

1. Introduction

Data intensive workflow is an important unifying mechanism to support modelling, management, and process of big data applications such as astrophysics, medical management, and social computing [1]. Technically, a data intensive workflow is composed by a set of computing or data intensive tasks, as well as the dependencies between them. At runtime, each task needs to be instantiated for execution. Due to the cost-efficiency and flexibility features of service-oriented computing and cloud computing, the execution of a task in data intensive workflows is usually delegated to a certain web service in the cloud.

However, services are autonomous entities which are usually out of control of the workflow manager, and may join, quit, and lie about their capacity to pursue and maximize their own interest and utility, shifting the execution environment from being static and trustworthy to open, dynamic, and non-deterministic [2]. In case of data-intensive tasks, any failure will ruin all great computation effort thus far. Therefore, the critical issues are to minimize the possibility a service fails and to minimize the cost of restarting affected services.

Many works have been carried out to address the challenges of service failure avoidance and restarting cost reduction respectively. For the first issue, the general approach is to apply the concept of trustworthiness, which is an aggregated value computed from a service's historical behaviours to measure the probability that it will succeed, to filter out unqualified services. For the latter one, intermediate data generated by a service is commonly stored as a checkpoint to avoid the rerun of itself and its ancestor tasks.

In fact, these two aspects are highly correlated. On one side, if trustworthy services are selected, they are more likely to succeed, thus reducing the probability of restart. On the other side, if checkpoints are made frequently, failure of services will have less impact on the system. Unfortunately, both solutions are expensive in cloud economy. Users have to pay higher fees for services with higher reputation, and as the volume of intermediate data of data-intensive tasks is generally high, the storage is costly in the cloud with respect to both money and time.

To improve the workflow reliability while reducing the overall cost, a systematic approach is required to find the trade-off between expensive services and costly checkpoints. Therefore, in this paper we introduce the risk evaluation model tailored for workflow structure to measure and achieve the

trade-off between the overhead of backup storage and the cost of data regeneration in failure, making the service selection and execution more efficient and robust.

The remainder of this paper is organized as follows. Section 2 gives a concrete example to illustrate the problem. Section 3 provides an overview of the state of the art. Section 4 provides an introduction to the risk evaluation model. Section 5 illustrates the intermediate dataset backup strategy based on the risk evaluation model for computing- and data-intensive workflows to improved the efficiency and reliability of execution. The design of experiment and results are demonstrated and discussed in Section 6. Finally, in Section 7 we present our conclusions and perspective for future work.

2. Motivation

Figure 1 illustrates a general example of bioinformatics workflow adapted from [3]. In the workflow, each task is represented as a rectangle, while its output data are represented in ovals. In the flow, there are a set of functional eligible service candidates to complete each task, with the input from its parent tasks and the output to its children tasks. In the example, MAFFT (Multiple Alignment with Fast Fourier Transform) is a multiple sequence alignment program for amino acid or nucleotide sequences, PHYLIP (PHYLogeny Inference Package) is a package of programs for inferring phylogenies (evolutionary trees), and UNIPROT (Universal Protein Resource) stands for a comprehensive and freely accessible database of protein sequence and functional information.

As merchants in the market, services are usually with different execution duration, price, and reliability for the same task, making the selection of optimized service critical for the running of a workflow. We have proposed to take the task dependency into consideration for improved workflow reliability in [4]. For example, for a simplified workflow consisting of two tasks MAFFT and PHYLIP, the reliability of PHYLIP has higher priority than MAFFT with respect to minimizing the side effect of service failure. That is, the failure of MAFFT affects merely itself, but the failure of PHYLIP needs the re-execution of both tasks.

Since MAFFT and PHYLIP are expensive and time-consuming to execute, if the second one has high probability to fail, it would be wise to save the output of MAFFT to avoid its re-execution. Unfortunately, the call of a

Figure 1: Example of bioinformatics workflow

backup service incurs delay and expense in cloud-based data-intensive workflow for the consumption of bandwidth and storage. Specifically, to make the decision of backup cost-effective and reliable, the risk of failure for each task needs to be considered, and the trade-off between overhead of backup and recovery needs to be achieved. Therefore, optimally deciding whether to call a backup service after a task completion is the focus of this paper.

3. Related work

For the challenge of service failure avoidance, many work proposed to use the concept of trustworthiness or reputation to model and measure the probability that each service will fulfil its commitments and promises from its historical behaviours, and incorporate trustworthiness into the service selection process to improve the reliability and robustness of the execution. However, most existing work [5, 6] has not taken the workflow structure into consideration, thus losing the opportunity to find tailored solutions. For example, a task at the end of a workflow should get more attention or priority than a task at the beginning, since the failure of the ending one will ruin all previous effort. Correspondingly, the workflow manager should

allocate more resources (time or money) for the ending task to assure its successful completion.

Intermediate data backup service works like checkpoint for workflows [7], which makes a snapshot of the execution state for future recovery. However, the location of the checkpoint usually needs to be decided manually. [8] proposed to select checkpoints by analysing the dependency graph between intermediate data to reach the trade-off between the storage cost and regeneration cost. However, it has not consider the impact of service reliability. In comparison, this paper applies the concept of risk to measure the benefit of performing the backup, and make decisions automatically at runtime to achieve trade-off between the overhead of backup itself and the potential loss.

Risk management is a common business practice, and should be part of QoS (Quality of Service) management. However, the common strategy in QoS for risk management is rescheduling or retrying alternative services [9, 10], neglecting the precise calculation of cost of failure for each decision. Our previous work [4] has proposed such a risk evaluation model by assigning different impact factors or weights to tasks based on their ordering relationship. However, the model is designed for general workflow application without considering the features of data-intensive workflows. In comparison, this paper applies and extends the risk evaluation model to study the relation between service reputation and data backup, as well as their impact on system reliability and exception handling.

Our work can be categorized as a subset of data provenance, which aims to support reproducibility of data sets and processes. Most work in this area [11] focuses on the approach to capture, filter, query, and trace provenance data on demand. In contrast, we investigate the automatic decision mechanism for data backup of internal tasks.

4. Risk evaluation model for workflows

4.1. Workflow model

A workflow process can be specified as a directed acyclic graph (DAG). In case of a workflow with iterations, its corresponding cyclic graph can be converted into a DAG by loop unrolling. For the purpose of this paper, all loops are assumed to be terminable. [12] provides more discussions on the loop unrolling.

Definition 1 (Workflow graph definition). A workflow is abstracted as a DAG, which is a quadruple $G = \langle V, E, t_0, t_n \rangle$ consisting of following elements:

- V is a finite set of nodes. Each vertex is associate with an executable atomic task $t_i, 0 \leq i \leq n$. To simplify the discussion and without loss of generality, we assume that tasks associated with different nodes are different.
- $E \subset V \times V$ is a set of edges connecting nodes in the graph. An edge, which is a pair $e = \langle t_i, t_j \rangle \in E$, defines a precedent relation between two tasks. It means task t_j will start if and only if t_i completes successfully.
- $t_0 \in V$ is the starting node of the graph.
- $t_n \in V$ is the ending node, signifying the completion or termination of the workflow.

It is required that each workflow has only one entry point, and completes at a single end node. The DAG definition for a workflow in fact defines a partial order among tasks.

To simplify the discussion, we assume that workflow graphs are well-structured [13] in the sense that it consists of matching pairs of a node that splits the flow and a node that joins the flow. [13] proves with corresponding transforming algorithms that all workflow graphs can be converted to be well-structured.

Definition 2 (Precedence relation). The edge $e = \langle t_i, t_j \rangle \in E$ in a DAG defines a precedence relation between tasks, denoted as $t_i \rightarrow t_j$. Task t_j is the successor of t_i , and t_i is the predecessor of t_j .

$\rightarrow^*$ is the reflexive and transitive closure of $\rightarrow$, and this closure defines a partial order between tasks.

Definition 3 (Execution path). In a DAG, a set of tasks with the relation of $(t_i \rightarrow t_{i+1}) \wedge (t_{i+1} \rightarrow t_{i+2}) \wedge \dots \wedge (t_{j-1} \rightarrow t_j)$ forms an execution path $\pi_{i\dots j} = (t_i, t_{i+1}, t_{i+2}, \dots, t_j)$.

The paths between t_i and t_j may not be unique, and the set of paths between t_i and t_j is denoted as $S_{i\dots j}^\pi$.

Figure 2: Sequential execution of tasks

4.2. Risk evaluation model

Risk assessment, which evaluates quantitative or qualitative value of risk related to a concrete situation and a recognized threat, is a common practice in business and social management. This section provides a brief introduction to the concept and definition of the risk evaluation model we previous proposed in [4]. To the purpose of this paper, this section explains the concept and computation method of the cost of failure for workflow structure. Refer to [4] for the details of the definitions and proofs.

Definition 4 (Cost of failure). The risk for performing a certain activity or decision is evaluated as the cost of failure, which is the multiplication of the potential loss L , and the probability p that the loss will occur. Formally, $cost_f = Lp$.

For a given execution path as shown in Figure 2, failure of any task on the path will cause the whole sequence to fail. Definition 5 and 6 shows the formula to compute the cost of failure for the sequence, which helps developers to evaluate the risk and consequences of task failure.

Definition 5 (Failure cost of a task). In execution path $\pi_{1\dots n} = (t_1, t_2, \dots, t_n)$, if each $t_i (1 \leq i \leq n)$ has the probability p_i to be completed for the price of $cost_i$, then the failure of the k^{th} task will cost

$$cost_f^{t_k} = \prod_{i=1}^{k-1} p_i (1 - p_k) \sum_{j=1}^k cost_j$$

Definition 6 (Failure cost of a path). In execution path $\pi_{1\dots n} = (t_1, t_2, \dots, t_n)$, if each $t_i (1 \leq i \leq n)$ has the probability p_i to be completed for the price of $cost_i$, then the cost of failure for the path is

$$\begin{aligned} cost_f^{\pi_{1\dots n}} &= \sum_{k=1}^n cost_f^{t_k} \\ &= \sum_{k=1}^n \left(\prod_{i=1}^{k-1} p_i (1 - p_k) \sum_{j=1}^k cost_j \right) \end{aligned}$$

Figure 3: Concurrent execution of tasks

In Definition 5, the cost of failure for a task is accumulated along the sequence of successful tasks since its failure will ruin all previous efforts. Accordingly, Definition 6 provides the method to compute the cost of failure for sequential execution of tasks.

Compared with sequential task execution, concurrent tasks, as shown in Figure 3, are triggered simultaneously. Definition 7 provides the method to compute the cost of failure for parallel running tasks.

Definition 7 (Failure cost of concurrency). $\parallel_{t_1, \dots, t_k}$ represents that $t_1, t_2, \dots, t_k$ is a group of tasks running concurrently, satisfying $(t_0 \rightarrow t_1) \wedge (t_0 \rightarrow t_2) \wedge \dots \wedge (t_0 \rightarrow t_k)$ and $(t_1 \rightarrow t_n) \wedge (t_2 \rightarrow t_n) \wedge \dots \wedge (t_k \rightarrow t_n)$. If each task $t_i (1 \leq i \leq k)$ has the probability p_i to be completed for the price of $cost_i$, then the cost of failure for their concurrent execution is

$$cost_f^{\parallel_{t_1, \dots, t_k}} = \sum_{i=1}^k cost_i \left(1 - \prod_{i=1}^k p_i\right)$$

Applying the concept of graph reduction and sub-workflow substitution [14], the cost of failure for the workflow, denoted as $cost_f^G$, can be computed recursively with Definition 6 for sequential reduction and Definition 7 for parallel reduction. Figure 4 illustrates an example for the the process of workflow graph reduction.

Sequential reduction uses a single task to substitute sequential tasks on a sub-flow or sub-path containing no synchronization task with multiple in-

Figure 4: Recursively computing the cost of failure for a workflow

or out-edges. For example, the execution path $\pi_{9,10}$ is replaced by $t_{9,10}$, and $\pi_{11\dots14}$ is replaced by t_{1114} from Figure 4a to Figure 4b.

Parallel reduction uses a single task to substitute several parallel running paths, each of which contains only one normal task. For example, t_{56} , which is a substitution of execution path $\pi_{5,6}$, and t_7 are reduced to t_{57} from Figure 4a to Figure 4b, while t_{38} , which is a substitution of path $\pi_{3\dots8} = (t_3, t_4, t_{57}, t_8)$, and $t_{9,10}$ and t_{1114} are reduced to t_{314} from Figure 4b to Figure 4c.

The details and algorithms of using sequential reduction and parallel reduction to compute $cost_f^G$ are provided in [4].

5. Risk-aware intermediate dataset backup strategy

5.1. Risk evaluation with backup service

In this section, we first introduce a data backup service DBS_i , which is in charge of saving the output of task t_i for future usage. According to most cloud storage service providers, such as Amazon and Google, the service is charged for the storage space, duration, and the outwards network usage.

Let vol_i be the size of output by t_i , $price_s$ be the unit price for the storage, $duration_i$ be the time length of active storage, $price_{out}$ be the unit price for outwards network transfer, then

- for the backup process, the cost is $cost_i^{backup} = vol_i * price_s * duration_i$;
- for the retrieving process, the cost is $cost_i^{retrieve} = vol_i * price_{out}$.

Most existing cloud storage providers compute the storage cost on daily basis. Since the storage of intermediate data is temporarily meaningful, we can set $duration_i$ as the life cycle length of overall workflow to simplify the discussion. For a workflow, $duration_i$ can be treated as a constant. Moreover, these providers do not charge for inward network usage and it is ignored in the paper as well. However, all discussions can be easily extended to include it.

If DBS_i is activated, the backup process will be carried out accordingly. However, the retrieving process will only be triggered in case of failure of t_i 's descendants.

The risk evaluation model in Section 4.2 does not consider the use of data backup service, therefore, the failure leads to a complete re-execution and the cost of failure needs to be calculated from the start node each time. In comparison, with the backup service, the retry of the workflow can retrieve

Figure 5: Example for retry sphere

the saved intermediate results and continue thereafter. To clarify affected tasks for re-execution, the concept of retry sphere is introduced.

Definition 8 (Retry sphere). Retry sphere, denoted as RS_j , is a set of tasks which need to be re-executed in case of retrying a failed task t_j . Let $saved(t_x)$ stand for that the result of t_x has been saved with backup service DBS_x . Formally, $RS_j = \{t_j\} \cup \{t_i | \exists \pi_{i \dots j} [\neg saved(t_i) \wedge [\forall t_k (t_i \rightarrow^* t_k) \wedge (t_k \rightarrow^* t_j) \wedge \neg saved(t_k)]]\}$.

Definition 8 says that if a task t_j fails, retry will involve itself intuitively. Moreover, for a given path from task t_i to t_j , if there is no intermediate task on the path between t_i and t_j is saved, t_i has to be re-executed to regenerate related dataset.

Figure 5 shows an example for retry sphere RS_7 . In the example, there are two paths to t_7 , $t_2 \rightarrow t_3 \rightarrow t_4 \rightarrow t_6 \rightarrow t_7$ and $t_2 \rightarrow t_5 \rightarrow t_6 \rightarrow t_7$ respectively. According to Definition 8, the retry sphere for the current running task t_7 , marked inside the dotted line, can be specified.

All tasks in the retry sphere are required for re-execution since their outputs are not stored. However, tasks outside the sphere have no impact on the retry of the failed task since their outputs can be retrieved from the storage service directly without re-execution.

The retry sphere can be computed recursively according to the precedent relationship between tasks as proved in Theorem 1. For a task t_j , let $Parents(t_j) = \{t_i | t_i \rightarrow t_j\}$ be the set consisting of parent nodes of t_j .

Theorem 1. *The retry sphere of task t_j is*

$$RS_j = \{t_j\} \cup \bigcup_{t_i \in Parents(t_j) \wedge \neg saved(t_i)} RS_i \quad (1)$$

Proof. If $Parents(t_j) = \emptyset$, according to Formula 1, $RS_j = \{t_j\}$, which satisfy the definition of retry sphere.

Assume that the formula is true for the case that $Parents(t_j) = t_1, t_2, \dots, t_n$ and the result of $t_i (1 \leq i \leq n)$ is unsaved, then

$$RS_j = \{t_j\} \cup \bigcup_{1 \leq i \leq n} RS_i$$

In case that $Parents(t_j) = t_1, t_2, \dots, t_n, t_{n+1}$ and the result of $t_i (1 \leq i \leq n)$ is unsaved, if the result of t_{n+1} is saved, the added edge $t_{n+1} \rightarrow t_j$ has no impact on the computation of RS_j . That is,

$$RS_j = \{t_j\} \cup \bigcup_{1 \leq i \leq n} RS_i$$

and Formula 1 remains true.

If the result of t_{n+1} is unsaved, t_{n+1} needs to be included into RS_j . According to Definition 8, all tasks in RS_{n+1} should also be re-executed. That is,

$$RS_j = \{t_j\} \cup RS_{n+1} \cup \bigcup_{1 \leq i \leq n} RS_i = \{t_j\} \cup \bigcup_{1 \leq i \leq n+1} RS_i$$

where Formula 1 also remains true.

That is, Formula 1 remains true for the case $n + 1$. Hereby Theorem 1 is proved. $\square$

Although backup services can avoid the re-execution of the corresponding tasks, they come with extra overhead of data storage and network transfer. In some cases, the overhead may outweigh the re-execution cost, especially when each service has a high probability of success. That is, if each service is likely to succeed, the saving of intermediate result is not likely to be used, which becomes a waste. Therefore, we apply the concept of cost of failure to find the optimal decision on whether to perform backup for the current running task or not.

Theorem 2. Let t_i be the completed task and t_j be its child ($t_i \rightarrow t_j$). If the output of t_i is not saved, the cost of failure for the retry of t_j is $cost_f^{ReG(i \rightarrow j)}$. $ReG(i \rightarrow j)$, consisting of RS_i and t_j , is the subgraph of the original workflow G for retrying t_j .

Proof. After the completion of t_i , since DBS_i is not performed, all tasks inside its retry sphere RS_i need to be re-executed if t_j fails. Therefore, these tasks are part of the re-execution subflow $ReG(i \rightarrow j)$.

If t_j fails, itself will be retried. Therefore, t_j is also included in the re-execution subflow $ReG(i \rightarrow j)$.

For a task t_x that $t_x \neq t_j \wedge t_x \notin RS_i$, it is not involved in the re-execution of t_j because the regeneration of t_i does not need it.

In summary, the cost of failure for retrying t_j is that of the subgraph including RS_i and t_j . $\square$

Theorem 2 shows that after the completion of the current task t_i , if the output of t_i is not saved, the failure of its child will lead to the re-execution of all tasks inside the retry sphere of t_i as well as itself. It should be notable that $ReG(i \rightarrow j) \subseteq RS_j$ since t_j may have more than one parent.

As shown in Figure 5, if the output of t_7 is not saved, the retry of t_8 contains a task set of $\{t_2, t_4, t_5, t_6, t_7, t_8\}$. A dummy task can be inserted as a splitting point for t_2 and t_4 to convert the subgraph to be well-structured DAG. Therefore, the cost of failure in this situation comes from the cost of failure for the subgraph contains the retry set. The detailed computation process for the cost of failure of a graph can be referred to [4].

Theorem 3. *Let t_i be the completed task and t_j be its child. If the output of t_i is saved, the cost of failure for the retry of t_j is*

$$cost_f^{DBS_i} = cost_i^{backup} + (1 - p_j)(cost_i^{retrieve} + cost_j)$$

.

Proof. After the completion of t_i , since DBS_i is performed, all tasks before t_i are no longer needed to re-execute. However, the overhead of running service DBS_i , which is $cost_i^{backup}$, is required no matter t_j is retried or not. Moreover, the probability that t_j fails is $1 - p_j$. After its failure, the cost for retrieving saved data is $cost_i^{retrieve}$ and the cost for re-executing t_j is $cost_j$. Therefore, the cost of failure for retrying t_j equals to $cost_i^{backup} + (1 - p_j)(cost_i^{retrieve} + cost_j)$. $\square$

According to Theorem 3, if DBS_i is performed, all tasks prior to t_i are no longer needed and the risk are diverted into DBS_i . As shown in Figure 5, if the output of t_7 is saved, the cost of failure to retry t_8 is $cost_7^{backup} + (1 -$

$p_8)(cost_8 + cost_7^{retrieve})$, which is highly dependent on the volume of output by t_7 .

With Theorem 2 and 3, Algorithm 1 can be applied to decide whether to perform backup service after the completion of the current task.

The main idea of the algorithm is to compute and compare the cost of failure for doing or doing not save the intermediate dataset for future recovery, and then make the optimized decision. The function $GET_RS(G, t_i)$ applies breadth-first search to traverse the graph to find the retry sphere or subgraph. It is rooted on Theorem 1.

5.2. Risk-aware service selection

If services are pre-assigned for each task, Algorithm 1 can be applied to find the set of tasks that need backup service. However, in service-oriented environment, there are generally several services with different processing capacities eligible to complete each task in the workflow graph G . The selection of services and the application of backup service have interrelated affects. On one side, selection of different services results in different backup strategy because of variance in the cost and reliability. On the other side, the application of backup service reduces retrying cost, thus allowing to select services somewhat unreliable but cheap.

The selection of different services results in different backup strategy, as well as different execution cost and reliability. Therefore, the selection of a set of services to satisfy the user's constraint of budget B from business logic, along with the backup strategy provided by Algorithm 1 is a multi-objective optimization problem.

The use of backup services in fact partitions the workflow graph into independent subgraphs with respect to task re-execution. Therefore, the cost of retrying is computed based on graph partitions. With Algorithm 1, a set of backup services are located, denoted as TS_{dbs} . Thereafter, the original workflow graph are partitioned into two parts:

- TS_{dbs} , the set of tasks accompanied by data back services. The cost of failure for these nodes are computed by the formula in Theorem 3.
- $subG_s$, the set of subgraphs containing task nodes not in TS_{dbs} . That is, $t_i \in subG_s \Leftrightarrow t_i \in G \wedge t_i \notin TS_{dbs}$.

The topology of elements in $subG_s$ is decided by the location of tasks in TS_{dbs} . By removing nodes, which are in TS_{dbs} , from G , the rest of the

Algorithm 1 Intermediate Dataset Backup Strategy

Require: G - DAG definition

Require: t_i - Current completed task

▷ Make optimal decision on performing backup or not

```
1: function DECISION_BACKUP( $G, t_i$ )
2:    $do\_backup = \text{false}$ 
3:    $\triangleright$  get the cost of failure in case of backup
   compute  $cost_f^{DBS} = cost_i^{backup} + (1 - p_j)(cost_j + cost_i^{retrieve})$ 
4:    $\triangleright$  Construct the subflow for possible retry
    $ReG = \text{GET\_RS}(G, t_i)$ 
5:   treat all children of  $t_i$  as parallel and substitute them as  $t_j$ 
6:   add  $t_j$  into  $ReG$ 
7:    $\triangleright$  add dummy task  $t_d$  into  $ReG$  to make it DAG
   for all  $t_x \in ReG$  do
8:     if  $t_x$  has no parent then
9:       add edge  $t_d \rightarrow t_x$ 
10:    end if
11:  end for
12:   $\triangleright$  get the cost of failure if no backup performs
   compute  $cost_f^{ReG}$ 
13:  if  $cost_f^{ReG} \geq cost_f^{DBS}$  then
14:     $do\_backup = \text{true}$ 
15:  end if
16:  return  $do\_backup$ 
17: end function

18: function GET_RS( $G, t_i$ )
19:    $RS = UnVisited = \{t_i\}$ 
20:    $\triangleright$  Find the retry sphere of  $t_i$ 
    $\triangleright$  Graph search
   for all  $t_j \in UnVisited$  do
21:     remove  $t_j$  from  $UnVisited$ 
22:     get  $Parents = \{t_x | t_x \rightarrow t_j\}$ 
23:     for all  $t_k \in Parents$  do
24:       if  $saved(t_k) == \text{false}$  then
25:         put  $t_k$  into  $RS$  and  $UnVisited$ 
26:       end if
27:     end for
28:   end for
29:   return  $RS$ 
30: end function
```

(a) Subgraphs are not connected

(b) Subgraphs are connected

Figure 6: Graph partition by removing tasks with data backup services

original graph may be divided into several disconnected components. That is, $|subG_s| > 1$. Figure 6a shows a workflow graph which is divided into two disconnected subgraphs. In contrast, the rest nodes may be connected as one graph as shown in Figure 6b. Subgraphs in $subG_s$ may not be complete since the removal of data backup nodes. In this case, the broken point can be linked to the ending node to make it a complete graph for the computation of cost of failure.

Let $\langle cost_i, p_i \rangle$ be the execution cost and probability to succeed for the case of selecting service s_i to execute task t_i . Multi-objective optimization parameters can be defined as follows to judge the suitability of the service selection.

$$\begin{aligned}
Cost(G) &= \sum_{t_i \in G} cost_i + \sum_{t_j \in TS_{dbs}} cost_j^{backup} \\
P(Suc(G)) &= \prod_{t_i \in G} p_i \\
cost_f^G &= \sum_{g \in subG_s} cost_f^g + \sum_{t_j \in TS_{dbs}} cost_f^{DBS_j}
\end{aligned}$$

Maximize $P(Suc(G))$ and minimize $cost_f^G$ subject to:

$$Cost(G) < B$$

$Cost(G)$ is the total cost of the overall schedule, which consists of the execution cost and data backup cost, and must be lower than the budget. $P(Suc(G))$ indicates the probability that the solution will finally succeed. It is a product of p_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 service's probability of success is independent. $cost_f^G$ is the cost of failure for the workflow. The value represents the risk of a schedule with respect to the possible loss.

Therefore, a cost-effective and robust scheduler should try to maximize $P(Suc(G))$ and minimize $cost_f^G$ while satisfying the constraint of budget. It should be noted that the proposed method aims to find a robust and reliable schedule within the budget, but not to merely find a solution with minimum cost. Whenever users wish to save more on the cost, they can input a lower B and relaunch the scheduling tool.

To find an optimized schedule for this multi-objective scheduling problem, we adopt genetic algorithm (GA) 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. For scheduling purpose, we treat each task in the workflow as a gene, and the vector of scheduled tasks as a chromosome. Task dependencies are not encoded into the chromosome. Instead, they are modeled into the computation of the cost of failure, which are in turn 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 function, denoted as $F(G)$, to encourage the higher robustness of the schedule, and the penalty function, denoted as $P(G)$, to enforce the constraints of budget B . In our definition, the higher the value of Ψ , the better the solution.

$$\begin{aligned}
F_{cost}(G) &= 1 - Cost(G)/B \\
F_{risk}(G) &= cost_f^G / Cost(G) \\
F(G) &= (1 - F_{risk}(G)) \cdot P(Suc(G)) \cdot F_{cost}(G) \\
P(G) &= \begin{cases} F_{cost}(G) & \text{if } Cost(G) \geq B \\ 0 & \text{otherwise} \end{cases} \\
\Psi &= F(G) + P(G)
\end{aligned}$$

In $F(G)$, the multiplication of $P(Suc(G))$ and $F_{cost}(G)$ indicates that the saving on execution cost is only valuable if the schedule succeeds. The result is further multiplied with $(1 - F_{risk}(G))$ to ensure the portion of saving on execution time or cost is in the safety zone. $P(G)$ is designed to enforce the bias against ineligible schedules which exceed the budget. After the chromosome and the evaluation function are defined, standard GA operators can be applied to find the optimized schedule.

6. Experiments and evaluations

In this section, experiments were designed to evaluate the impact of using intermediate data backup strategy or not on workflow scheduling with respect to cost-effectiveness and reliability. The environment for the experiments was created by adopting the DAG dataset of the Resource-Constrained Project Scheduling Problem [15].

6.1. Experiment design

In simulation, each task was supported with a set of different services with various cost and success probability levels (mean values). Assume the size of data for task t_i to process is VOL_i , which was randomly selected between 100 and 2000 (GB). To make the simulation more realistic, we assume that the cost of services for t_i is in direct proportion to its VOL_i . That is,

$mean_cost(t_i) = \alpha_i \cdot VOL_i$, where α_i is a random number between $[0.1, 0.5]$. Moreover, the mean probability to succeed was set to 1.0.

The exact value of cost and probability for each service were obtained by randomly fluctuating about 20% around the cost and probability level of its corresponding task. To make the simulated environment more practical, the fluctuation followed the observation that in commerce the success probability of a service is in direct proportion to the cost.

In addition, the size of output data for t_i is set to $vol_i = \beta_i \cdot VOL_i$ where output portion β_i is a random number between $[0, 1]$. To investigate the impact of output volume on system performance, we classify the experiments into four categories with different β_i groups, including $[0, 0.1]$, $[0, 0.2]$, $[0, 0.5]$, and $[0, 0.8]$. Referring to Amazon and Google, the prices for data storage is set as \$0.03/GB per month, and outbound network usage is \$0.10/GB.

To simulate different levels of user constraints, the budget B was evaluated at a wide range of possible values between the maximum and minimum allowed cost for the workflow. k was used for the constraint levels of B , which ranged from 0 to 1. The higher the value, the tighter the constraints.

$$B = maxCost - k(maxCost - minCost)$$

In the experiments, $maxCost$ and $minCost$ were obtained by always selecting the service with the most or least processing cost for each task respectively.

The implementation of the GA was based on [16]. The population for each generation was 50 and the total generation was 2000 for each experiment. All other parameters, such as crossover and mutation probability, were the system default.

The result was evaluated with two parameters. One was $P(Suc(G))$ for the probability to succeed, and the other was the risk ratio $RiskRatio = cost_f^G / Cost(G)$ for the percentage of cost of failure in the total cost. $cost_f^G$ for both service selection strategies, with and without backup services, are computed and compared. The higher, the better for $P(Suc(G))$, but the lower, the better for $RiskRatio$.

A summary of the parameters used in the experiments is listed in Table 1.

6.2. Results and analysis

The experiments were carried out by increasing k from 0 to 1 by 0.1, representing constraint of B from the relaxed to the tight.

Table 1: Key parameters in the experiments

Type	Name	Meaning	Value
Simulation Parameters	VOL_i	the size of data for task t_i to process	[100, 2000]
	α_i	the ratio between the size of data and its processing cost for t_i	[0, 0.5]
	β_i	the ratio between the output and input size of data for t_i	[0, 1]
	k	the constraint levels for the budget B	[0, 1]
Evaluation Parameters	$P(Suc(G))$	the probability for the schedule to succeed	[0, 1]
	$RiskRatio$	the ratio between the cost of failure and the total cost	[0, 1]

Experiments were conducted on 20 randomly chosen DAGs with 32 nodes. Figure 7a and Figure 7b were two example graphs.

Figure 8 shows the results about the correlation between B , $P(Suc(G))$, and $RiskRatio$ of schedules in these experiments. In fact, although the DAG definitions and resource settings were different in these experiments, they yielded similar results.

The results show that $P(Suc(G))$ and $RiskRatio$ get worse as the constraint levels become tighter for both strategies. However, service selection with backup services performs better, with respect to the probability of overall success and possible failure loss, especially when the output portion is small. The improvement relies on the running of backup service when it outperforms the re-execution overhead. As shown in Algorithm 1, when the output portion increases, the cost of failure for applying backup service becomes larger, making it less likely to happen.

7. Conclusion and future Work

In this paper, we extend the risk evaluation model [4] with the consideration of data backup service to balance the trade-off between overhead of backup and re-execution after exceptions. The proposed method computes and compares the potential loss with and without data backup to optimally

Figure 7: Random DAGs with 32 nodes

decide the occasion to perform backup service after the completion of a task. Thereafter, we design the utility function with the model and apply a genetic algorithm to find the optimized schedule, thereby maximizing the robustness of the schedule while minimizing the possible cost of failure.

The experiments show that application of backup service can generally improve system robustness and reliability without exceeding the customer constraints, such as cost and reliability requirements. Specifically, the proposed method performs better when the data generated by each task is not large.

The method proposed in this paper has not taken the needs of data privacy and anonymity [17, 18] by intermediate datasets into consideration. Therefore, in the future, we plan to add and integrate some privacy preserving mechanism to ensure the security for datasets stored and transferred. Moreover, incremental processing of high-volume datasets in cloud is a common scenario in data-intensive applications [19, 20]. We will extend our model to make it suitable for this kind of processing model. We also intend to evaluate and study the impact of various cost models from cloud providers on the application of the proposed method. Finally, we plan to take more QoS parameters, such as deadline and responsiveness, into the service selection process.

(a) Output portion < 0.1

(b) Output portion < 0.2

(c) Output portion < 0.5

(d) Output portion < 0.8

Figure 8: Cost of failure at variant output portion and constraint levels

Acknowledgement

The research work reported in this paper is supported by National Science Foundation of China under Grant No. 61100172, No. 61272512, and No. 61300177.

A preliminary version of this paper appeared in 2012 International Conference on Cloud and Green Computing.

References

- [1] F. Magoules, J. Pan, F. Teng, Cloud Computing: Data-Intensive Computing and Scheduling, 1st Edition, Chapman & Hall/CRC, 2012.
- [2] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berri-man, J. Good, On the use of cloud computing for scientific workflows, in: ESCIENCE '08: Proceedings of the 2008 Fourth IEEE International Conference on eScience, IEEE Computer Society, Washington, DC, USA, 2008, pp. 640–645.
- [3] S. Pettifer, J. Ison, M. Kalas, D. Thorne, P. McDermott, I. Jonassen, A. Liaquat, J. M. Fernandez, J. M. Rodriguez, I. Partners, D. G. Pisano, C. Blanchet, M. Uludag, P. Rice, E. Bartaseviciute, K. Rapacki, M. Hekkelman, O. Sand, H. Stockinger, A. B. Clegg, E. Bongcam-Rudloff, J. Salzemann, V. Breton, T. K. Attwood, G. Cameron, G. Vriend, The embrace web service collection, Nucleic Acids Research.
- [4] M. Wang, K. Ramamohanarao, J. Chen, Dependency-based risk evaluation for robust workflow scheduling, in: 2012 Workshop on Large Scale Distributed Service-oriented Systems (In conjunction with 26th IEEE International Parallel & Distributed Processing Symposium), 2012.
- [5] M. Rahman, R. Ranjan, R. Buyya, Reputation-based dependable scheduling of workflow applications in peer-to-peer grids, Comput. Netw. 54 (2010) 3341–3359.
- [6] M. Wang, K. Ramamohanarao, J. Chen, Trust-based robust scheduling and runtime adaptation of scientific workflow, Concurrency and Computation: Practice and Experience 21 (16) (2009) 1982–1998.

- [7] R. Duan, R. Prodan, T. Fahringer, DEE: A distributed fault tolerant workflow enactment engine for grid computing, Vol. 3726 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2005, pp. 704–716.
- [8] D. Yuan, Y. Yang, X. Liu, G. Zhang, J. Chen, A data dependency based strategy for intermediate data storage in scientific cloud workflow systems, *Concurrency and Computation: Practice and Experience* 24 (9) (2012) 956–976.
- [9] E. Deelman, D. Gannon, M. Shields, I. Taylor, Workflows and e-science: An overview of workflow system features and capabilities, *Future Generation Computer Systems* 25 (5) (2009) 528 – 540.
- [10] L. Qi, W. Lin, W. Dou, J. Jiang, J. Chen, A QoS-aware exception handling method in scientific workflow execution, *Concurrency and Computation: Practice and Experience* 23 (16) (2011) 1951–1968.
- [11] T. Dalman, M. Weitzel, W. Wiechert, B. Freisleben, K. Noh, An online provenance service for distributed metabolic flux analysis workflows, in: Ninth IEEE European Conference on Web Services (ECOWS), 2011, pp. 91 –98. doi:10.1109/ECOWS.2011.20.
- [12] M. Weißbach, W. Zimmermann, Termination analysis of business process workflows, in: Proceedings of the 5th International Workshop on Enhanced Web Service Technologies, ACM, New York, NY, USA, 2010, pp. 18–25. doi:10.1145/1883133.1883137.
- [13] J. Vanhatalo, H. Völzer, F. Leymann, S. Moser, Automatic workflow graph refactoring and completion, in: Proceedings of the 6th International Conference on Service-Oriented Computing, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 100–115.
- [14] J. Cardoso, A. Sheth, J. Miller, J. Arnold, K. Kochut, Quality of service for workflows and web service processes, *Web Semantics: Science, Services and Agents on the World Wide Web* 1 (3) (2004) 281 – 308. doi:10.1016/j.websem.2004.03.001.
- [15] Project Scheduling Problem Library - PSPLIB, accessed on 28-May-2014. [link].
URL <http://129.187.106.231/psplib/>

- [16] K. Meffert, N. Rotstan, C. Knowles, U. B. Sangiorgi, JGAP - Java Genetic Algorithms and Genetic Programming Package, accessed on 28-May-2014. [link].
URL <http://jgap.sourceforge.net/>
- [17] X. Zhang, C. Liu, S. Nepal, S. Pandey, J. Chen, A privacy leakage upper-bound constraint based approach for cost-effective privacy preserving of intermediate datasets in cloud, *IEEE Transactions on Parallel and Distributed Systems* 24 (6) (2013) 1192–1202. doi:10.1109/TPDS.2012.238.
- [18] X. Zhang, L. T. Yang, C. Liu, J. Chen, A scalable two-phase top-down specialization approach for data anonymization using mapreduce on cloud, *IEEE Transactions on Parallel and Distributed Systems* 25 (2) (2014) 363–373. doi:10.1109/TPDS.2013.48.
- [19] C. Olston, G. Chiou, L. Chitnis, F. Liu, Y. Han, M. Larsson, A. Neumann, V. B. Rao, V. Sankarasubramanian, S. Seth, C. Tian, T. ZiCorneil, X. Wang, Nova: continuous pig/hadoop workflows, in: *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, SIGMOD '11*, ACM, New York, NY, USA, 2011, pp. 1081–1090. doi:10.1145/1989323.1989439.
URL <http://doi.acm.org/10.1145/1989323.1989439>
- [20] X. Zhang, C. Liu, S. Nepal, J. Chen, An efficient quasi-identifier index based approach for privacy preservation over incremental data sets on cloud, *Journal of Computer and System Sciences* 79 (5) (2013) 542 – 555. doi:10.1016/j.jcss.2012.11.008.
URL <http://www.sciencedirect.com/science/article/pii/S0022000012001766>

Mingzhong Wang received his Ph.D. degree in computer science from the University of Melbourne in 2010. Since 2010, he has joined School of Computer Science, Beijing Institute of Technology. His research interests include parallel and distributed processing, massive data processing, and information security.

Liehuang Zhu is currently a professor at School of Computer Science, Beijing Institute of Technology. He is an expert in network security. His research interests include security protocol analysis and design, group key exchange protocol, wireless sensor network and cloud computing.

Zijian Zhang received his Ph.D. degree in computer science from Beijing Institute of Technology in 2012. He is currently a lecturer in School of Computer Science, Beijing Institute of Technology. His research interests include distributed systems, security protocol analysis, provable security, and group key exchange protocol.