rERA: An Optimization Algorithm of Task Dependency Graph for Scheduling

Zhuo CHENG Yasuo TAN Yuto LIM

School of Information Science, Japan Advanced Institute of Science and Technology (JAIST)

1 Introduction

Currently, almost all the practical systems are multi-task systems, such as chemical and nuclear plant control, telecommunications, and multimedia systems. In such systems, different tasks work together to achieve desired functions. In order to guarantee the correctness of the functions, the tasks are required to be completed in specific orders. Any violation of such orders will lead to the systems in unpredictable states, which may cause disasters.

To characterize such orders, task dependency graph is an expressive form. In order to generate a scheduling sequence that is consistent with the orders characterized by the task dependency graph, scheduling algorithms need to traverse the graphs. Therefore, the complexity (size of node and edge) of the graphs can obviously affect the efficiency of the scheduling algorithms. Unfortunately, task dependency graph directed obtained by some technical, such as causality interfaces [1], usually contains some redundant edges which can increase the complexity of the graph.

In this paper, based on the analysis of task dependency, we propose an algorithm to remove the redundant edges of the task dependency graphs. The effectiveness of the algorithm is illustrated through a simple example. By this way, the complexity of the task dependency graph is reduced, which can improve the efficiency of scheduling algorithms.

2 Task Dependency Graph

Definition (task dependency graph) *A task dependency graph is a directed acyclic graph.* $G = (V, E, v_0, v_e)$*, where* V *is task (node) set,* $E \subseteq V \times V$ *is dependency relation (edge) set, with* $(v_i, v_j) \in E$*,* $v_i \neq v_j$*, where* $v_i, v_j \in V$*.* $v_0 \in V$ *is the start task, and* $v_e \in V$ *is the end task.*

An edge (v_i, v_j) in the task dependency graph means task v_j can start to execute only after task v_i has been completed. We use $v_i \prec v_j$ to illustrate this dependency relation. The dependency relation is transitive. That is, $v_i \prec v_j, v_j \prec v_k \implies v_i \prec v_k$. The left side of Fig. 1 shows a task dependency graph with six tasks and eight dependency relations.

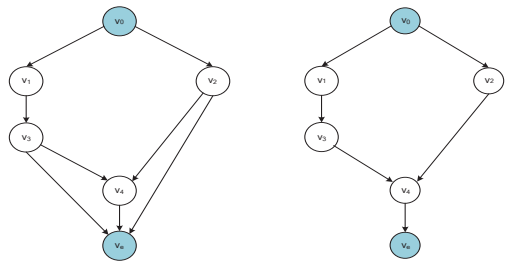

Fig. 1 Task Dependency Graph

Algorithm 1 redundant Edge Removal Algorithm (rERA)

Input: task dependency graph $G = (V, E, v_0, v_e)$
Output: task dependency graph $G' = (V, E', v_0, v_e)$ without redundant edges
1: $E' := E$
2: **for all** node $v_i \in V$ **do**
3: compute $D(v_i)$, the set of descendants nodes of task v_i
4: compute $C(v_i)$, the set of child nodes of task v_i
5: **end for**
6: **for all** node $v_i \in V$ **do**
7: **for all** $v_j \in C(v_i), v_k \in D(v_i)$, and $v_j \in D(v_k)$ **do**
8: remove the edge (v_i, v_j)
9: update $C(v_i)$
10: $E' := E' \backslash (v_i, v_j)$
11: $C(v_i) := C(v_i) \backslash v_j$
12: **end for**
13: **end for**
14: **return** the optimized graph $G' = (V, E', v_0, v_e)$

3 Optimization Algorithm

Task dependency graph is to characterize task execution orders. However, some edges in the graph are not necessary, which can increase the complexity of the graph. For example, in the left figure of Fig. 1, edge (v_3, v_4) and (v_4, v_e) indicate the relation $v_3 \prec v_4$ and $v_4 \prec v_e$. Based on the transitivity of dependency relation, we can get $v_3 \prec v_e$, which means edge (v_3, v_e) in the graph is not necessary as it indicates the dependency relation $v_3 \prec v_e$ that has already been indicated by other edges. To remove such redundant edges, we propose *redundant Edge Removal Algorithm (rERA)* for task dependency graph described in Alg. 1.

Through using rERA, we can get the optimized task dependency graph as shown in the right side of Fig. 1. Compared with the left side of Fig. 1, we can see that, two redundant edges (v_3, v_e) and (v_2, v_e) are removed. This has reduced the complexity of the task dependency graph, which can increase the efficiency of scheduling algorithms.

References

[1] E.A. Lee, H. Zheng, and Y. Zhou "Causality Interfaces and Compositional Causality Analysis," *Foundations of Interface Technologies*, pp. 1–16, 2008.