Available online at www.sciencedirect.com

ScienceDirect

J. Parallel Distrib. Comput. 67 (2007) 1137–1154

Journal of Parallel and Distributed Computing

www.elsevier.com/locate/jpdc

Dynamic partner identification in mobile agent-based distributed job workflow execution

Yuhong Feng[a], Wentong Cai[a,*], Jiannong Cao[b]

[a]*School of Computer Engineering, Nanyang Technological University, Block N4, Nanyang Avenue, Singapore 639798*
[b]*Department of Computing, Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong*

Received 3 October 2005; received in revised form 29 August 2006
Available online 19 July 2007

Abstract

This paper is concerned with the design, implementation, and evaluation of algorithms for communication partner identification in mobile agent-based distributed job workflow execution. We first describe a framework for distributed job workflow execution over the Grid: the Mobile Code Collaboration Framework (MCCF). Based on the study of agent communications during a job workflow execution on MCCF, we identify the unnecessary agent communications that degrade the system performance. Then, we design a novel subjob grouping algorithm for preprocessing the job workflow's static specification in MCCF. The obtained information is used in both static and dynamic algorithms to identify partners for agent communication. The mobile agent dynamic location and communication based on this approach is expected to reduce the agent communication overhead by removing unnecessary communication partners during the dynamic job workflow execution. The proof of the dynamic algorithm's correctness and effectiveness are elaborated. Finally, the algorithms are evaluated through a comparison study using simulated job workflows executed on a prototype implementation of the MCCF on a LAN environment and an emulated WAN setup. The results show the scalability and efficiency of the algorithms as well as the advantages of the dynamic algorithm over the static one.
© 2007 Elsevier Inc. All rights reserved.

Keywords: Distributed job workflow execution; Mobile agent communication; Communication partner identification; Grid computing

1. Introduction

Data-intensive collaborative scientific computations, such as bioinformatics [18] and digital imaging survey (e.g., Sloan Digital Sky Survey (SDSS) [1]), have the following characteristics:

- They can be represented as Directed Acyclic Graph (DAG).[1] For example, the identification process of galaxy clusters within SDSS can be represented as a DAG [2].
- They often require diverse, high volume and distributed data sets. Hence, huge volume of data motion over the Internet makes it a bottleneck for such applications [3].

Job workflow is a natural technology for developing such computations. To keep data local as much as possible and to provide decentralized control of execution, distributed job workflow execution using mobile agent (MA) is getting increasing research attention. Many systems have been developed (e.g., those described in [9,10,14,23,25,26]). However, in these systems, an MA usually carries all the implementation codes and supporting packages during its migration. When the code size is increased, the overhead caused by code transfer is increased fast, thus prolonging the job workflow execution time [13]. This may compromise the benefits of code mobility.

The Mobile Code Collaboration Framework (MCCF) was developed to enable and enhance the productivity of large-scale data-intensive computation over the Grid [12] using MA technology. In order to reduce the overhead caused by code transfer during the distributed job workflow execution, Lightweight Mobile Agent (LMA) [5] and Code-on-Demand (CoD) [15] techniques are adopted in the development of MCCF. Similar to the aforementioned distributed workflow systems using MAs, there is no centralized workflow engine in MCCF.

* Corresponding author. Fax: +65 67926559.
E-mail address: aswtcai@ntu.edu.sg (W. Cai).

[1] There exist scientific applications in which some subjobs need to be executed repeatedly. This kind of applications cannot be represented as DAG [24]. In this paper, we are interested only in applications that can be represented as DAGs, whose nodes denote subjobs and edges denote their data dependency.

0743-7315/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2007.06.012

Both execution results of subjobs and the job workflow control are directly transferred amongst distributed resources. During a job workflow execution, dynamic LMA replication, parallel LMA migration and execution are employed to support concurrent execution of multiple data independent subjobs so as to improve system performance. This in turn gives rise to the requirement of communication between agents.

Two agents communicating with each other are called *communication partners* or *partners* in short. For message passing based agent communication, the first step is to identify corresponding partners, the second step is to locate (or discover) them, and the third step is to route the message to them.

Agent discovery methods in the existing message passing based MA communication can be classified into *discovery infrastructure-based* methods and *fully decentralized* methods. Discovery infrastructure-based methods include *home-based* [27], *forwarding-based* [7] and *hierarchical-based* [30] methods. As pointed out in [19], all discovery infrastructure-based methods rely on a static and small-scaled distributed lookup server for MA discovery, which does not scale well for large-scale MA systems.

Fully decentralized methods include *flooding-based* [17], *distributed hash table* (DHT) based [19], and *contact list*-based [28] methods. Flooding-based methods generate excessive flooding messages which will degrade the overall system's performance considerably and cause a large amount of traffic over the Internet. The DHT-based mechanism takes advantage of the DHT overlay, but the maintenance of the DHT overlay is costly. In addition, the number of hops required for sending a message is $\log n$, which complicates the reliable message delivery besides introducing additional delay in MA migration.

A contact list-based mechanism for MA discovery is adopted in MCCF for its simplicity. For contact list-based MA location and communication, each MA maintains a list of all its partners' locations. Before an MA is migrated or discarded, it will notify its partners so that they can update their contact lists accordingly.

Previous work [7,17,19,27,28,30] on MA communication normally assumes that the partner is already known and focuses mainly on dynamic MA location/discovery and message routing. However, with the dynamic MA replication during the distributed job workflow execution in MCCF, this assumption does not hold any more. Thus, partner identification is an important issue in MA communication in MCCF. Particular, for contact list-based method, it is to identify partners to be included in the contact list.

This paper makes three important contributions in the field of partner identification for agent communications during distributed job workflow execution: (i) Via studying agent communications and dynamic characteristics of a job workflow execution over MCCF, we identify the unnecessary agent communication partners. (ii) We propose a subjob grouping algorithm for preprocessing the job workflow's static specification. The obtained information is then used in both the static and dynamic algorithms for communication partners identification. MA dynamic location and communication based on this approach is expected to reduce the agent communication overhead. In addition, the proof of the correctness and effectiveness of the dynamic algorithm are elaborated. (iii) We provide a performance comparison study of the simulated job workflows on the actual implementation of the MCCF and the algorithms.

This paper is organized as follows: the MCCF overview and further explanation of the problem will be given in Section 2. The subjob grouping algorithm together with the static and dynamic partner identification algorithms based on it will be given and analyzed in Section 3. An experimental study of our contact list-based method will be described in Section 4. Finally, Section 5 concludes the paper and outlines our future work.

2. MCCF

A job workflow can be graphically represented as a DAG, denoted as $\mathcal{G} = (J, E)$, where $\mathcal{J}$ is the set of vertices denoting subjobs, i.e., $\mathcal{J} = \{J_0, J_1, \ldots, J_{n-1}\}$, where n is the number of subjobs. $\mathcal{E}$ is the set of directed edges between subjobs. There is a directed edge from J_i to J_j (that is, $(J_i, J_j) \in \mathcal{E}$), if J_j requires J_i's execution results as input. In this case, J_i is called J_j's predecessor, and J_j is called J_i's successor. *Data dependency* (denoted as "$<$") between two subjobs exists if there is a path (a concatenation of directed edges) between the subjobs.

Let $\mathcal{S}(J_i)$ denote the successor set of subjob J_i. $\forall J_{j_1}, J_{j_2} \in \mathcal{S}(J_i)$, J_{j_2} is called J_i's *indirect successor* if $J_{j_1} < J_{j_2}$, otherwise, it is called J_i's *immediate successor*. Similarly, we can define *indirect predecessor* and *immediate predecessor*.

A job workflow application is modelled as a partially ordered set $\mathcal{W}_x = (\mathcal{J}, <)$, where x is the job workflow id. If $J_i < J_j$, then J_i is called J_j's ancestor, and J_j is named as J_i's offspring. The set of ancestors of J_i is denoted as $\mathcal{A}(J_i)$, and the set of offsprings of J_i is denoted as $\mathcal{O}(J_i)$.

It is assumed that a DAG representing a job workflow always has a unique staring node, J_0, and a unique end node, J_{n-1}, and $\forall J_i \in \mathcal{J}$, $0 < i < (n-1)$, $J_0 < J_i < J_{n-1}$, $\mathcal{A}(J_i) \neq \emptyset$ and $\mathcal{O}(J_i) \neq \emptyset$ must be true. An example job workflow $\mathcal{W}_a$ is illustrated in Fig. 1(a).

2.1. Grid resources

In MCCF, executable codes are provided as dynamic services, i.e., executable codes can be downloaded and instantiated on arbitrary computational resources [20]. Assuming that executable codes are kept in code repositories to provide dynamic services, Grid resources then include data repositories, computational resources, code repositories, and network resources.

For a certain data set required by a subjob, there may exist several data repositories having the data set, which are represented as a set $\mathcal{D} = \{d_0, \ldots, d_{d-1}\}$. Similarly, for a certain code required by a subjob, multiple code repositories may have the code, which are represented as a set $\mathcal{C} = \{c_0, \ldots, c_{c-1}\}$. For a certain subjob to be executed, there could be multiple computational resources satisfying the subjob's computation