RSR-CGSF: A Robust Semantic Resource Based Cooperative Grid Service Framework

Ling Chen
Shenzhen Institute of Advanced Technology (SIAT),
Chinese Academy of Sciences (CAS)
Shenzhen, China

Sheng Di
Department of Computer Science,
The University of Hong Kong
Hong Kong, China

Abstract—Efficient cooperative management on semantic interrelated grid resources is a challenging problem. This paper presents a cooperative grid service framework which mainly has two advantages. First, we design an ontology based algorithm to maintain the consistency in cooperative work with low transmission overhead. Second, we propose a resource recovery mechanism to ensure system robustness. Through our analyses we prove that our design has high robustness; through our experiments we validate that the framework is efficient.

Keywords: Grid Service; Robustness; Semantic Relationship; Cooperation

I. INTRODUCTION

Compared to the traditional web service, Grid service, armed with Web Service Resource Framework (WSRF) concept [1], is increasingly paid attentions, in that it can support the utilization of resources with Open Grid Service Architecture (OGSA). Hence, the applications based on Grid framework are able to provide more powerful functions in uniform ways, such as operating database or maintaining user sessions. However, the resource used in traditional Grid framework lacks the semantics supported and the corresponding controlling mechanism, thus causing relatively low usability and high cooperative (or collaborative) synchronization overhead.

In this paper, we propose a cooperative Grid service framework based on semantic resource (RSR-CGSF). Compared to the traditional Grid service framework, RSR-CGSF has at least two advantages, semantic based cooperative management and robustness.

Semantic based cooperative management: Due to the high possibility that the semantic grid resources may be cooperatively managed (e.g. modified) by various users meanwhile, one of the biggest challenges lies in how to design an efficient consistency mechanism. The traditional solutions to solve such problem usually focus on frequently multicasting the operational changes to all cooperative users for simplicity, causing high network traffic. Comparatively, our method leverages the semantic relationships among different resources to minimize the transmission overhead by focusing on the minimum partition of the relative semantic resource set. Since ontology [2], a core concept in semantic web [3], is a uniform tool to reveal semantic relationships among objects, we propose an ontology based mechanism, not only ensuring the correctness of concurrent modification on resources but reducing transmission cost.

Robustness: Any resources may probably be removed by any user’s inattentive behaviors or lost by failure of server machines. Thereby, RSR-CGSF also supports at least two replicas of each resource loosely-coupled stored on a cluster server and one copy on the client side to maintain a high reliability. Based on our analysis, the safe degree of resources can easily reach 99.9% in case of user’s operational mistake probability being 10% or server’s failure probability being 10%.

Besides, our RSR-CGSF also integrates many classical system design principles, such as rule of clarity/simplicity, rule of modularity, and so on. RSR-CGSF is also flexible and easy to use in that it provides three interfaces: web service interface, client launch interface, and web browser interface.

The rest of the paper is organized as follows. In Section II, we present our design architecture and describe the above two advantages in details. In Section III, we describe our implementation. We evaluate and analyze the performance of our system in Section IV. Some related works are presented in Section V. Finally, we conclude and present the future work in Section VI.

II. ROBUST SEMANTIC RESOURCE BASED COOPERATIVE GRID SERVICE FRAMEWORK

The architecture of our Grid service framework is shown in Fig. 1. There are 4 layers from top to bottom, Representation Layer, Engine Layer, Controller Layer and Single System Image (SSI) Resource Layer. Representation Layer provides several web service (WS) interfaces that enable any user or application submit SOAP [4] requests. WSRF SOAP Engine in Engine Layer is responsible for receiving these requests and forwarding them to SOAP Decoder/XML Parser, so that the controller layer composed of Account Controller, Consistency Controller, and Operation Controller, can continue to process the requests. Account Controller is used to register and check user’s identity. Operation Controller together with Consistency Controller is used to execute users’ requests according to semantic based consistency mechanism. All users’ data are kept in SSI Resource Layer, which is on the basis of distributed file system, such as Google File System [5] or Hadoop Distributed File System (HDFS) [6], in that the data in files are relatively easy to understand and operate for “Everything is file in UNIX/LINUX” [7]. There are two types for each resource, operational resource (such as R.i shown in Fig. 1) and its
B. Consistency Mechanism

Before describing our consistency algorithm, we dictate a scenario as follows: whenever a user wants to add a new section to a document commonly shared in a Virtual Organization (VO), he/she has to demonstrate the relation between it and other parts in the document. In this example, each section is a resource instance and they have unidirectional/bidirectional relationships.

The relationships among resources conforms to the following relation-laws:

- Symmetry: A B-DEP B ⇔ B B-DEP A
- Transitivity: A B-DEP B, B B-DEP C ⇒ A B-DEP C
- Distributivity: A B-DEP B, A U-DEP C ⇒ B U-DEP C

A B-DEP B, C U-DEP A ⇒ C U-DEP B

Notice that there is no transitivity rule on U-DEP in the relation-laws. That is, A U-DEP B, B U-DEP C cannot get A U-DEP C. This is because the U-DEP’s transitivity will get the network traffic overhead non-linear, which goes against our expectation of lowering the transmission cost.

Assume all the resources are organized in a set namely W. In terms of Symmetry and Transitivity law, the resources which are B-DEP relation to each other will construct a subset of T, namely B-DEP set Q. It is obvious that the B-DEP sets will construct a partition of the whole T, each of which belongs to an excluded VO or a group. Based on the Distributivity rule, for any B-DEP set, there must be another two sets, namely PU-DEP set and RU-DEP set. The former contains the resources that unidirectional dependent (U-DEP) upon the resources from the B-DEP set. The latter contains the resources upon which the resources from B-DEP set are unidirectional dependent (U-DEP). Fig. 3 shows the example of the B-DEP set Q and its two relative U-DEP sets according to Fig. 2.

**Algorithm 1: Consistency Algorithm**

**Input:** ontology, target resource instance A, set S of other resource instances

**Output:** The operation of writing or reading resources

1. Perform relation-laws on the inputted ontology;
2. Construct set P in which any resource is B-DEP relation to each other;
3. IF (there does not exist R locked by others, where R ∈ P) THEN
4. Lock A;
5. WHILE (resource R in S) DO
6. IF (A U-DEP R) THEN
   Subscribe R; /*Any changes to R will be notified to A’s user*/
7. END IF
8. IF (A B-DEP R) THEN
9. Lock R; /*exclusive lock*/
10. END IF
11. END WHILE
12. NEW THREAD T IS CREATED:
    /*Periodically perform the following check in the spawn thread T*/
13. WHILE (resource R in S) DO
14. IF (R U-DEP A) THEN
15. Notify the modification of A to R.
16. END IF
17. END WHILE
18. END THREAD T
19. Release the lock on A & all the locks on R in S by the algorithm.
20. END IF
Line 1 is used to generate minimum complete restrictions on the semantic related resources. Line 5–11 is used to perform either subscription or exclusive lock on semantic related resources. Line 12–18 is a new thread spawned to periodically (in real time) notify other users based on their subscriptions on resource A. At the end, all the locks should be released. Line 1 consumes at most 3n steps, because all the relative resources can be split to three categories, PU-DEP set, B-DEP set, and RU-DEP set. Likewise, line 2, line 5–11 and line 12–18 respectively costs up to n steps. Thus, the time complexity of this algorithm is $O(n)$, where $n$ is the number of resources in the input set $S$. Besides, we prove that there is no deadlock problem in our design based on the THEOREM 1.

**THEOREM 1**: The deadlock will never occur in the semantic resource management if and only if each user locks their resources based on relation-laws and Algorithm 1.

**Proof**: Based on the deadlock’s necessary conditions proved by Coffman [8], the deadlock occurs when all the following 4 conditions are met: Mutual exclusion condition, Hold and wait condition, No preemption condition, and Circular wait condition. Notice that the first three conditions are always met for $B$-DEP relationship because $B$-DEP indicates that the relative resources are supposed to be exclusively locked in the mean time. Accordingly, the deadlock exists if the fourth Coffman’s condition follows: There exists a circle in the $B$-DEP set partitioned from the whole resource set such that $A_i$ $B$-DEP $A_j$, ..., $A_m$ $B$-DEP $A_i$, where $A_i$ is resource and $m \geq2$. It is clear that the unique possibility of leading to deadlock must originate from this circle. Assume a deadlock exists; there must be a circle in which at least two resources are locked by different users, which contradicts to Line 3 of Algorithm 1. ■

**C. Robustness Mechanism**

The replica mechanism of resources (or files) is crucial to any Grid system. In our design on the server side, every user is just allowed to modify or delete the operational resource (i.e. $R_{i,j}$ shown in Fig. 1), while the hided corresponding shadow resource (i.e. $R_{i,j}$ shown in Fig. 1), will never be changed directly. In fact, for saving the cost of accessing resources, the direct changes to the status or content of resources are always maintained in the memory and/or file disk of the client-side. The operational resources on server end will never be changed unless the client sends a SOAP request to command so. Further more, the shadow resource will only be updated by the system periodically or by user’s explicit commands.

We argue that two replicas for each resource (or file) stored on server side is quite enough in most cases. Suppose the probability of a user making a mistake by deleting/modifying resources on client-side, server’s operational resource and shadow resource are respectively $P(X_d)$, $P(X_r)$, and $P(X_s)$, then the combinational probability $P(Dirty)$ of generating a dirty resource which cannot be restored is shown in Equation (1).

That is, the safety of user’s operation is $1 - P(Dirty)$. For instance, by considering $P(X_d) = P(X_r) = P(X_s) = 10\%$ which is a fairly bad case, $P(safety\ of\ user’s\ operation) = 1 - 0.11^2 = 99.9\%$, which is quite high for the reliability of user’s daily work.

$$P(Dirty) = P(X_d) \cdot P(X_r) \cdot P(X_s)$$

In addition, as for the node failure event, suppose the error probability of the memory/disk on the client and that on any server node are $P(F_d)$ and $P(F_r)$ respectively. Then, the situation of losing the right status of resources only occurs when the client and both of the two server disks containing the two replicas fail in the mean time. That is, the failure probability $P(Failure)$ is shown in Equation (2) and the safety of user’s resource is $1 - P(Failure)$. For instance, by considering $P(F_d) = 10\%$ which is a bad enough case, $P(F_r) = 10\%$, $P(safety\ of\ user’s\ resource) = 1 - 0.11^2 = 99.9\%$.

$$P(Failure) = P(F_r) \cdot P^2(F_d)$$

**D. Other Classical System Design Principles**

We follow many classical system design principles [7] in RSRCGSF, as following (but not limited):

**The Rule of Modularity/Clarity/Simplicity**: “Clarity/ simplicity with clean interfaces are better than cleverness”. RSRCGSF is organized in four distinct layers and each of them is implemented with clean interfaces. Moreover, the users/clients are located on the edge of the Internet and are able to access the services through the readable SOAP messages.

**The Rule of Security**: Compared to the way of adding extra complex Public Key Infrastructure (PKI) layer, we use the Account Controller in Controller Layer to ensure the access security for the lightweight purpose. In our policy, any grid user should be consistent with the server’s system account image, such as Network Information System (NIS) or Linux Virtual Server (LVS), so to ensure PKI similar security.

**The Rule of Extensibility**: “Design for the future”. The representation layer composed by web services is a powerful interface for any clients or other systems/applications to easily inter-operate.

**III. IMPLEMENTATION**

Based on our framework, we implemented a prototype. In this implementation, we construct Grid services by Apache Muse open source project and build users’ resource space by Hadoop Distributed File System (HDFS) [6].

As mentioned in Section II, the crucial contribution is our designed semantic based consistency mechanism. Fig. 4 shows the resource-changing message in SOAP format. If a resource being modified by one user is unidirectional dependent on another resource being modified by another user named Tony, the former user’s client will periodically request the modifications of Tony every 5 seconds in our implementation.

**IV. EXPERIMENTAL RESULTS**

Since the scalability of the system mainly depends on the performance of the container or SOAP engine, which is beyond the scope of this paper, we mainly test the response times of various remote operations through SOAP over Internet. In our experiment, the communication is across the Internet. Fig. 5 shows the response times of different operations generated by client side. From this figure, we can observe that Register operation costs least because it transmits the least amount of information, while Login operation usually costs distinctly
higher due to its relatively complex confirmation procedure on the server side. The recoverOperation spends much less than either QueryRecord or UpdateRecord (160K in Fig. 5), in that it just needs to dump a replica in terms of an existing resource or restore a resource from its replica on servers. In general, the amount of the places changed by cooperative users on a resource within a short period (5 seconds in our implementation) is very limited (e.g. just a record or a sentence or a metadata), thus the consistency transmission overhead mainly relies on the network latency, close to about 0.05 second of “changing metadata” operation in this experiment.

V. RELATED WORKS

One related work is workflow [9]. However, the execution of workflow always depends on the time sequence based relations without semantic consistency support and recovery mechanism. Some other related works [9, 10] focus on cooperative editing. Unlike our work, they neither support flexible grid resources nor semantic scenarios. To the best of our knowledge, compared to the existing Grid researches [11], our work is also innovative by using semantic consistency management on grid resources.

VI. CONCLUSION AND FUTURE WORK

In this paper, we present a robust Grid service based framework which supports cooperative work, abbreviated as RSR-CGSF. The major advantages of this framework are its inherent flexibility and reliability due to the adopted Grid service supported and semantic based resource organization. Based on RSR-CGSF, we implemented a prototype to validate that our consistency mechanism owns high efficiency and robustness. For the future work, we plan to manage widely distributed resources from different clusters over Internet.

ACKNOWLEDGMENT

This paper is supported by The National High Technology Research and Development Program of China under grant 2007AA120502 and 2006AA01A114.

REFERENCES