A Decentralized Quickest Response Algorithm for Grid Service Discovery

Jianjun Hu
Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences
P.O.Box 8718, Beijing 100080, P.R.China
+8610-62630989
hujj@otcaix.iscas.ac.cn

Heqing Guan
Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences
P.O.Box 8718, Beijing 100080, P.R.China
+8610-62630989
ghq@otcaix.iscas.ac.cn

Hua Zhong
Technology Center of Software Engineering,Institute of Software, Chinese Academy of Sciences
P.O.Box 8718, Beijing 100080, P.R.China
+8610-62630989
zhongh@otcaix.iscas.ac.cn

ABSTRACT
Computational grid is an emerging computing infrastructure that enables effective access to high performance computing resources. Service discovery is one of the most important research issues of grid computing. However, traditional service discovery algorithms are not suitable for grid environment due to the facts of inherited decentralization and loose coupling of grid application itself. To address this problem, we in the paper analyze the limitation of those traditional algorithms, and present a novel grid oriented service discovery algorithm based on quickest response approach, namely DSDA, which is essentially decentralized. DSDA also takes network latency and processing capability of service nodes into account. Our later experiment justified DSDA's effectiveness in the grid environment.

Categories and Subject Descriptors
D.2.m [Software Engineering]: General.

General Terms
Algorithms

Keywords
Grid computing, service discovery, decentralization, network latency

1. INTRODUCTION
The emergence of global computational grids brings changes to the traditional paradigm of distributed computing. It is now possible that one computation task is executed by a collection of different computing and communication services or resources, virtually brought together by the grids [1]. Grid applications are distributed, autonomous, dynamic and loop-coupled, and numerous homogeneous or heterogeneous nodes work cooperatively.

Infoscale 2007, June 6–8, 2007, Suzhou, China.

These characteristics bring challenges to traditional service discovery algorithms [2]. First, traditional algorithm needs to select an arbitration node that is responsible for request assignment. If the arbitration node is malfunctioned, other nodes could not work well. Taking UDDI (Universal Description, Discovery and Integration) as an example, tremendous service provider information is stored on centralized UBR (UDDI Business Registry) servers potentially, which makes servers become the bottleneck of the system. Additionally, the information stored in UBR servers is static. To keep the coherence and validity of the UBR servers' information is difficult. Second, traditional algorithms focus only on static functional specification, and seldom consider run-time information of service provider. In grid environment, nodes connected via wide area network, the network latency cannot be neglected, and the processing capacity of each service node differs much. So, runtime service discovery should take these factors into consideration.

The desirable features of grid service discovery should be both scalability and efficiency [10]. To achieve scalability, service discovery should not rely on only a few centralized nodes, which could be potential performance and security bottleneck. To achieve efficiency, qualified service providers should be located rapidly without too much network and processing overhead. In this paper, on the basis of quickest response algorithm, we present an enhanced decentralized service discovery algorithm for global grid environment, namely DSDA, which is both scalable and efficient.

The remainder of this paper is organized as following, section 2 describes the system model and service model; section 3 presents DSDA algorithm, and also the validation proof; section 4 gives performance tests and results; related researches are discussed in section 5, and we conclude the paper in the last section.

2. Model Description and Formulation

2.1 System Model
Suppose the number of nodes in the system model is n ($n>=2$). These nodes are connected by network in some kind of topology, and different node has different processing capability. Because there may be some joining nodes, crashing nodes and dropping nodes that change the topology of the system, we simply suppose the topology of the grid system does not change during one time of load distribution procedure, i.e., the amount of nodes in the

system model does not change. So the system can be represented by an undirected graph $G= (V, E)$, where set V denotes heterogeneous nodes and set E denotes communication links of nodes.

As we can see, $|V|=n$. We mark the nodes in V as V_1, V_2, …, V_n. So set $V=\{V_i \mid 1 \le i \le n\}$. The heterogeneity of V_i is depicted by a five-tuple $(i, C_i, L_i, Service_i, Table_i)$, which is defined as follows:

- i represents the unique serial number of the node, and $1 \le j \le n$.
- C_i represents the processing capability of node V_i, and it's a positive number. C_i is determined by the hardware architecture of V_i, such as CPU and memory. We define a standard value, and each C_i is a relative value to the standard value.
- L_i represents the load status (e.g. CPU & memory utilization rate) of V_i, and its range is [0, 1).
- $Service_i$ is the table of every running service in V_i.
- $Table_i$ is a forward list, and is a set of record R. $R=(ID_S, V, t)$, where ID_S is the ID of service to be invoked, V is the node to forward the request of invoking ID_S, and t is the latest update time of record R.
- Set E denotes communication links of nodes, i.e., $E= \{D_{i,j} \mid 1 \le j \le n, 1 \le j \le n\}$. $D_{i,j}$ represents the communication latency time between V_i and V_j. We simply suppose $D_{i,j}= D_{j,i}$.

2.2 Service Model

Each service is depicted by a triple (ID_S, T_S, M), which is defined as follows:

- ID_S is the unique ID of service, and in our case is the name of service.
- T_S is the time cost of the service that running in a node with one unit processing capability, and it is determined before the service is deployed.
- M represents the weighted factor of the system, and it is the most important factor which is using to calculate the latency time of each node.

3. DSDA Algorithm and Analysis

3.1 Strategy of DSDA

In decentralized service discovery, any grid node can accept and forward service request, there is no proprietary service broker. The basic idea of quickest response approach for service discovery is when any node in G that receives a request for invoking service S, to find the most suitable node and forward the request to this node. Unostentatiously, in order to get the best result, the node V_i which receives the invoking request has to forward the request to other nodes in G. Other nodes which have deployed the service S, run S and return the result to V_i. If the first response is from V_j, then V_j will be the best node for running this request. Obviously, this approach is costly.

An improved approach is, when each request arrives, every node doesn't run service S but uses a simulation of running service S. Suppose the request is forwarded by node V_i, and the time of forwarding request, running service and returning the result to V_i by V_j is $T(i, j)$. The time of running service S on a node of one unit processing capability is represented by T_S. Then the time of running S on node V_j is $T_{Sj} = \dfrac{T_S}{C_j \times (1 - L_j)}$. And $T(i, j) = 2 \times D_{i,j} + T_{Sj}$

If node V_j has deployed S, then it waits T_{Sj} and return a response to V_i when receiving request from V_i. The time T_{Sj} is regarded as a simulation of running S on node V_j. Node V_i will assume the first responded node to be the most suitable one and forward the request to this node.

It's easy to see the improvement of this approach, for it uses a simulation of running services and the result of this approach is almost the same as the former approach. However, it also has some limitations, i.e., when node V_j receives requests, it has to wait a interval of T_{Sj} and then returns the response to V_i. This interval is so long that it is almost the same as running the service.

A further improvement could be, when node V_j receives the request, it doesn't wait for T_{Sj}, but multiply T_{Sj} with a constant M, i.e., $T'_{Sj} = M \times T_{Sj}$. And $T'(i, j) = 2 \times D_{i,j} + T'_{Sj}$.

We can decrease the value of M, and let T'_{Sj} not affect the result. That is, the value of M satisfies the following condition:

Condition 1: If $T'(i,k)$ =min{ $T'(i,1)$, $T'(i,2)$, … , $T'(i,n)$ }, then $T(i, k)$=min{$T(i, 1)$, $T(i, 2)$, …, $T(i, n)$}.

When node V_j receives request from node V_i, then wait for a interval of T'_{Sj} and return a response to V_i. Node V_i will forward this request to the first responded node. Because the condition 1 is satisfied, $T'(i, k)$ is the minimum value, so $T(i, k)$ is also the minimum. That is, no matter node V_j waits for T_{Sj} or T'_{Sj}, it will get the same result.

In this approach, the interval that nodes should wait has been decreased, and the algorithm DSDA is based on this approach.

3.2 DSDA Algorithm

In a grid $G= (V, E)$, node V_i receives the request of invoking service S at the moment of t_{Begin}. The following algorithm outputs the response node V_{result}.

For node V_i:

(1) Searches service table $Service_i$. If service S isn't deployed, then jumps to (3), otherwise searches corresponding record $R_i=(ID_S, V, t_i)$ of S. Compares t_i with t_{Begin}, if R_i is expired, then jumps to (2), otherwise $V_{result}=V$ and algorithm ends.

(2) Sends $Request= (i, ID_S, t_{Begin})$ to other nodes, and calculate $T_i = \dfrac{M \times T_S}{C_i \times (1 - L_i)}$ at the same time. Suppose the first response is from node V_j at the time t_{i1}, then $V_{result}=V_j$, and update the record $R_i= (ID_S, j, t_{i1})$. If no response received after T_i, then $V_{result}=V_i$. Suppose the time of V_i is t_{i2}, then update record $R_i= (ID_S, i, t_{i2})$, and algorithm ends.

(3) Sends $Request= (i, ID_S, t_{Begin})$ to other nodes. Suppose the first response is from node V_j at the time t_{i3}, then $V_{result}=V_j$,

and update the record R_i= (ID_S, j, t_{i3}). Algorithm ends.

For other node V_j in G:

(1) Search service table $Service_j$ when receiving request from V_i. If service S isn't deployed, then algorithm ends.

(2) Calculates $T_j = \frac{M \times T_S}{C_j \times (1 - L_j)}$ and sends *Response= (i, ID_S, j, t_{Begin})* to V_i after T_j. Algorithm ends.

3.3 Validity of DSDA

The objective of DSDA is to take the processing capability, load and network latency of each node in account, and provide the fastest response of each request. So the criterion for evaluating the validity of DSDA should be, whether the output of DSDA is the best choice for processing the specific request.

We propose the following theorems in DSDA.

Theorem 1: There is a non-negative M which satisfies Condition 1.

Proof: First, the time that node V_j waits can't be negative, i.e., $\frac{M \times T_S}{C_j \times (1 - L_j)} \geq 0$ .And T_S, C_j and 1-L_j are all positive values, so $M \geq 0$.

Second, when M=1, $T(i, j) = T'(i, j)$. Condition 1 is obviously to be satisfied. And when M>1, $T'(i, j) > T(i, j)$. That is, the time node V_j waits is longer than running the service, the algorithm is useless when M>1. So the value of M should be in the area of [0, 1].

Now define $\alpha_j = 2 \times D_{i,j}$, $\beta_j = \frac{T_S}{C_j \times (1 - L_j)}$. So $T'(i, j)$ can be a linear equation of M. $T'(i, j) = \alpha_j + \beta_j \times M$. So Theorem 1 can be represented as:

$\exists P \in [0, 1)$, When M=P and $T'(i, j)$ has the minimum value, then when M=1, $T'(i, j)$ still has the minimum value. $T'(i, j)$ is represented as the line in Figure 1. Because of the continuity of the line, there is a point P_1. When the value of M is in the area (P_1, 1), Condition 1 is satisfied.

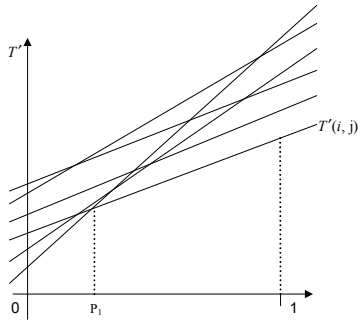

Figure 1 *T' (i, j)*

So, there is a non-negative M which satisfies Condition 1.

Theorem 2: The output of DSDA is the best choice for processing the specific request, if proper value of M is set.

Proof: Suppose the first response that node V_i receives is from node V_j. That is, $T'(i, j)$ =min{ $T'(i, 1)$, $T'(i, 2)$, … , $T'(i, n)$ }.

According to theorem 1, there's a proper value of M that satisfied Condition 1, and $T(i, j)$=min{$T(i, 1)$, $T(i, 2)$, …, $T(i, n)$}.

So, V_j is the best choice for processing the specific request.

There's no cycle in DSDA, so the complexity of DSDA is $O(1)$.

In DSDA, the value of M affects the result and performance. When the value is relatively small, DSDA stresses on the network latency, while the value is big, stresses on load. But obviously, the value should be as small as possible at the precondition of satisfying Condition 1.

4. Experiments and Analysis

We have made three groups of tests aiming at the scalability and performance of DSDA, which are presented in detail as follows.

The hardware environment of the test is Dell PowerEdge2600, with two Intel Xeon 3.06G Hz CPUs, 2G memory and 36G SCSI hard driver. The software environment is Globus Toolkit3.2 [3] and RedHat Linux9.0. We use threads to simulate the nodes in grid environment, and each thread represents for one node.

In the experiment, we use the CPU frequency and memory volume to assess the processing capability of each node, use CPU usage and memory usage to judge the load. We compare DSDA with QRA (quickest response algorithm) [4]. QRA is very similar to DSDA in that, the controller in QRA records the network latency to other nodes, and forwards the request to the fastest responded node. However, differs from DSDA, QRA still need a monitoring node to forward the requests.

4.1 Scalability evaluation

The service invoking requests arrive at a random order, and satisfy following conditions:

- The requests are independent of each other.
- To a Δt which is small enough, the possibility of there being one request arrives in the area of [t, t+ Δt], is independent to t and in the direct ratio to Δt .
- To a Δt which is small enough, the possibility of there being more than one request arrives in the area of [t, t+ Δt] is very small.

With these conditions, we can describe the arriving of the requests by using Poisson distribution. So, in the interval of [t_0, t_0+t), the possibility of there being n requests is $P_t(n) = \frac{(\lambda t)^n}{n!} e^{-\lambda t} (n = 0, 1, 2, \text{L})$.

This experiment is to evaluate the scalability of DSDA. The network latency between each nodes variates from 50ms to 1s. And the execution time of 80% services ranges between 200ms and 2s. In

this experiment, we set λ to be a constant and adjust the number of the nodes to evaluate the scalability. The experiment data is shown in Figure 2, where *X* axis represents for the number of nodes, and *Y* axis represents for the average complete time.

As shown in Figure 2, the average complete time keeps almost steady when the number of nodes increases within 500. And as the number of nodes are over 500, the average complete time increases at a very low speed. So DSDA works well in scalability.

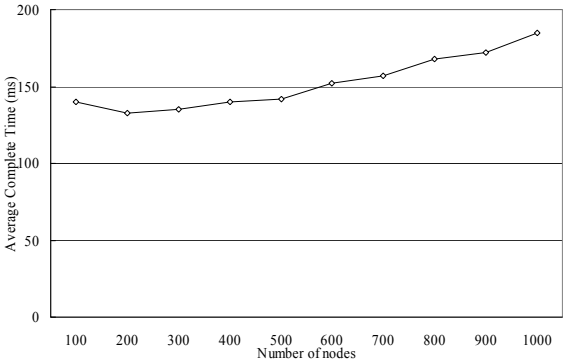

Figure 2 Scalability of DSDA

4.2 Performance evaluation

The environment is almost the same as experiment 1, except that the number of nodes is fixed to 100. We compare the performance of DSDA and QRA in different working loads and show the result in Figure 3. In this figure, the *X* axis represents for the average number of requests that arrive in one second, to evaluate the working load.

QRA is based on network latency and doesn't take working load into consideration. When the requests arrives slowly, i.e., working load is not heavy, QRA gains a better performance because of its simpler processing procedure. However, when the load becomes heavy, DSDA assigns working load better and gains a better performance.

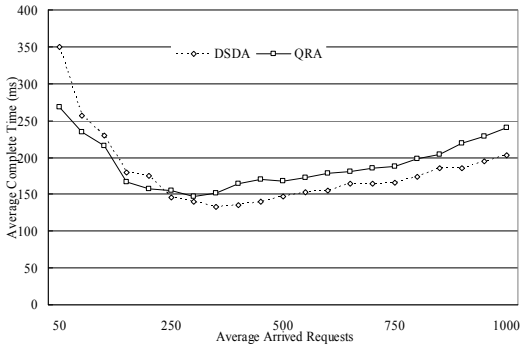

Figure 3 Performance of DSDA

4.3 Load balancing evaluation

We also fixed the number of node to 100, and run the same task to compare the load balancing performance of the two algorithms. We pick 20 time point randomly and get the load L_i of that moment in node V_1 to V_{100}. The value of L_i is represented in number, e.g., when the load of V_5 is 50%, then L_5=50.

So the average value of working load is:

$$\overline{L}_t = \frac{\sum_{i=1}^{100} L_i}{100}$$

And the standard deviation is:

$$\sigma_t = \sqrt{\frac{\sum_{i=1}^{100} \left(L_i - \overline{L}_t\right)^2}{100}}$$

We calculate the σ_1 to σ_{20} of these two algorithms, and show them in Figure 4. As we can see from the definition of the standard deviation, the bigger value of σ means the working load of each node differs more. So DSDA gets a better assigning load balancing performance than QRA.

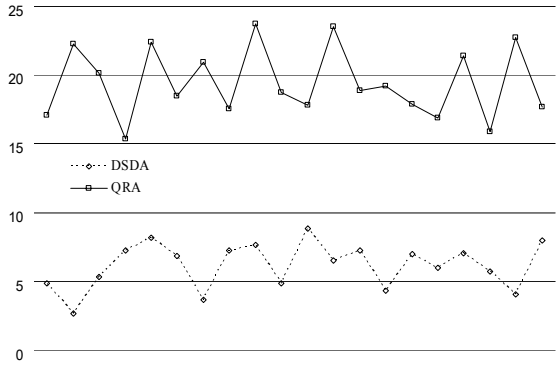

Figure 4 Standard deviation

5. Related works

There are many researches concerning service discovery in grid environment. Initial researches mainly focus on service functionality matching. The key technology is building specification model to describe service functionality. The approach evolves from syntactic-based interface description (for example WSDL) to semantic-based description (for example, Semantic web and ontology) [13], and in many service ontology, service providers' QoS are included. However, these approaches do not consider the runtime information of service environment, the discovery mechanism using static QoS attributes of service provider specified in service deployment time can not adapt to the variation of runtime environment.

With the development of global large-scale computing grid, decentralized service discovery mechanism has been intensively investigated. Presently there are two kinds of solution: P2P-based approach and agent-based approach.

In P2P-based service discovery models, certain data structure such as DHT (distributed hash table) [11] or DPT (dynamic prefix tree) [12] were used to efficiently manage service information and forward service request. In fact, these approaches are not completely decentralized, because they require DHT nodes to

route service requests. As in our approach, there is no proprietary router, any node can accept and forward request, so it is thoroughly decentralized.

The SLP (service location protocol) [5] of IETF defines an agent-based service discovery framework. However, SLP is designed for service discovery in limited area, such as local area network, so it is not very suitable for wide area network environment. Although some extensions to SLP have been proposed aiming to the WAN environment [6], it is still found unsuitable for global grid environment. Zegura and Stemm introduce anycast of applications into service discovery [7, 8], and compare the QoS of each service provider, but their approach does not consider runtime information of grid environment. Cao J. et al also propose an agent-based and performance-driven resource scheduling framework [9]. This framework uses agent to represent the resource grid. Agents communicate and cooperate with each other, and use service discovery mechanism for resource sharing. Comparing with ours, this approach is tight coupled with applications and will increase the additional development.

6. Conclusion

Traditional service discovery and resource sharing algorithms in grid computing are lack of loose-coupling and distribution, and do not take the network latency and working load into consideration. We proposed a grid service discovery algorithm based on decentralized quickest response approach, it has an efficient scaling mechanism for load balancing among requests. Our simulated experiment justified algorithm's effectiveness in the grid environment.

The deficiency of the algorithm lies in how to determine and change the value of weight, so one of our future works will emphasize on the adaptive change of the weight and the method the weight to be propagated in grid environment. Another future work will be applying this algorithm in a real grid infrastructure.

7. REFERENCES

[1] Foster, I., and Kesselman, C. The Grid: Blueprint for a New Computing Infrastructure, Morgan-Kaufmann Publishers, San Francisco, CA, 1998.

[2] Tanenbaum, C., and Steen, M. Distributed Systems-Principles and Paradigms, Prentice Hall, 2002.

[3] Globus. Available at http://www-unix.globus.org

[4] Bourke, T. Server Load Balancing, O'Reilly, 2001.

[5] Guttman, E., Perkins, C., Veizades, J., and Day, M. Service Location Protocol, Version 2. IETF RFC-2165, 1998.

[6] Steven, E., Czerwinski, S., Zhao, B., Hodest, T., Joseph, A., and Katz, R. An Architecture for a Secure Service Discovery Service. Proceedings of ACM MobiCom'99. Seattle, WA, 1999.

[7] Zegura, E., Ammar, M., Fei, Z., and Bhattacharjee, S. Application-Level Anycasting: A Server Selection Architecture and Use in a Replicated Web Service. IEEE/ACM Transactions on Networking, 8(4)455-466, 2000.

[8] Stemm, M., Seshan, S., and Katz, R. A Network Measurement Architecture for Adaptive Applications. Proceedings of IEEE Infocom 2000. Tel Aviv, Israel. 2000.

[9] Cao, J., Daniel, P., Stephen, A., Subhash, S., and Graham, R. Agent-Based Grid Load Balancing Using Performance-Driven Task Scheduling. Proceedings of 17th IEEE International Parallel & Distributed Processing Symposium (IPDPS 2003). Nice, France, 2003.

[10] Zhu, C., Liu, L., Zhang, W., Xu, Z.N., and Yang, D.S. Using Service Taxonomy to Facilitate Efficient Decentralized Grid Service Discovery. Proceeding of 3rd International Conference on Grid and Cooperative Computing (GCC 2004). Wuhan, China, 2004.

[11] Xia, Q., Yang, R., Wang, W., and Yang, D. Fully Decentralized DHT based Approach to Grid Service Discovery using Overlay Networks. Proceedings of the Fifth international Conference on Computer and information Technology (CIT 2005). Washington, DC, 2005.

[12] Caron, E., Desprez, F., and Tedeschi, C., A Dynamic Prefix Tree for the Service Discovery Within Large Scale Grids. Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing (P2P2006), Cambridge, UK, 2006.

[13] Zhu, C., Liu, Z., Zhang, W., Xiao, and W., Huang, J. An Efficient Decentralized Grid Service Discovery Approach based on Service Ontology. Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence (WI'04), 2004.