# On-Demand Service in Grid: Architecture, Design and Implementation

ZhenChun Huang[†] , Chuan He[†] , Lei Gu[†] , JianFeng Wu[‡]
*[†] Dept. of Computer Science and Engineering, Tsinghua Univ., Beijing 100084, China*
*[‡] Shanghai Stock Exchange, Shanghai, China*
*huangzc@tsinghua.edu.cn,{hechuan97, jackflit98}@mails.tsinghua.edu.cn, jfwu@sse.com.cn*

## Abstract

*Service Oriented Architecture becomes the mainstream of grid research today. On-demand service, which can provides better flexibility for grid users, is purposed in this paper. According to the research on different implementation policies, on-demand service architectures for virtual machine based on-demand service and service specification reference based on-demand service is proposed. Furthermore, it is designed and implemented based on Java virtual machine and Groovy script language. Performance testing to the implementation shows that on-demand service can provide better flexibility for grid by low cost, and can speedup mass data processing services usually.*

***Keywords:*** *Grid, On-Demand Service, Java Virtual Machine, Groovy Script Language*

## 1. On-Demand Service in Grid and its implementa-tion strategies

Service Oriented Architecture becomes the mainstream in Grid research since the OGSA [1] and WSRF [2] are published. In order to make it more flexible and extensible, grid requires more extensibility, flexibility and performance than web service. For instance, service in web service framework is deployed statically and can't be invoked until it is deployed. But, in order to make it more flexible and extensible, grid requires dynamic deployment, which can generate and deploy grid service to response the requirement of users. For example, grid system is often used to process mass data that stores in different service node from the node that provides the processing service. We have to transfer the mass data from the storage node to the processing node so that it can be processed. If the processing service can be deployed on the node near the storage node, the analyzing result that is much less will be transferred instead of mass data, and the service cost will be down.

In order to provide more flexibility, a dynamic deployment technique in grid named "On-Demand Service" is proposed in this paper. It can provide dynamic deployment capability in grid and fulfill the requirement of flexibility. Furthermore, it is designed, implemented and tested based on Axis/Tomcat in this paper.

Literally, on-demand service can be explained as "provide right service for user according to his requirement". It can be implemented by different strategies in several levels. The simplest one is parameterized custom service (PCS), which is still a kind of static service. It is pre-customized for different requirements of users by parameters. It can be regarded as a kind of aboriginal on-demand service.

In order to provide sufficient flexibility for users, on-demand service must be deployed dynamically: user makes a request of service features such as interface description, function specification, etc., system generate service code from the request and deploy the code as a service into service container dynamically for user's invoking.

According to the difference of service code and its generation, implementation strategies of on-demand service can be separated into several types. The first one is homogeneous platform based on-demand service (HP-ODS). In homogeneous platform environment, binary codes for service implementation can be moved among nodes and run on any node simply. But it's too difficult to make all nodes homogeneous in grid.

In order to make the dynamic on-demand service practicable, source-code compatible platform based on-demand service (SCCP-ODS) and virtual machine based on-demand service (VM-ODS) are proposed. They are based on a common high-level programming language such as C language or a well-defined virtual machine, and describe service implementation by high-level programming language or virtual machine pseudo code. Unlike HP-ODS, their restrictions of run time environment is more reachable in a grid system.

Under the support of a series of service specification reference, a kind of more flexible and powerful on-demand service named "Service Specification Reference based On-Demand Service" (SSR-ODS) is purposed. By SSR-ODS, a user provides the service specification, which includes the service interface description, service function description and service implementation description, first. With the help of these specifications, service provider can generate service code and deploy them dynamically. Then, the user can invoke the deployed service. It's independent of programming language, platform architecture, or virtual machine. Compared with other strategies, SSR-ODS is much more flexible.

In summary, features of on-demand service implementation strategies are listed below: (Figure 1)

| | **Service specification language** | **Deploy-ment** | **Limitation** |
|---|---|---|---|
| **PCS** | Binary code | Static | Pre-deployed |
| **HP-ODS** | Binary code | Dynamic | Homogeneous platform |
| **SCCP-ODS** | Source code of high level language | Dynamic | Source code compatible |
| **VM-ODS** | Pseudo code of virtual machine | Dynamic | Virtual machine |
| **SSR-ODS** | Service specification language | Dynamic | Support service specification reference |

| | **Executing performance** | **Deploy-ment cost** | **Flexi-bility** | **Feasibility** |
|---|---|---|---|---|
| **PCS** | High | Very low | Very bad | Good |
| **HP-ODS** | High | Low | Bad | Bad |
| **SCCP-ODS** | High | Middle | A bit bad | Average |
| **VM-ODS** | Middle | Low | Average | Good |
| **SSR-ODS** | From low to high | High | Good | Good |

Figure 1. Strategies of on-demand service implementations

Because of it is impossible to keep all nodes homogeneous in a grid, HP-ODS is unpractical. SCCP-ODS is restricted by the difficult of confirming the source code compatibility of platforms. It is practicability is not good. VM-ODS builds on a universal virtual machine, which is easy to be installed and configured usually. The differences of hardware and software such as architecture, operating system, programming language are masked by the virtual machine. It's possible to support VM-ODS by universal virtual machine in a grid system. SSR-ODS is also feasible because of it only requires nodes in grid to support a universal service specification reference.

It is shown in figure 1 that on-demand service can be implemented feasibly by VM-ODS and SSR-ODS strategies. In this paper, VM-ODS and SSR-ODS will be researched, designed and implemented based on Java virtual machine and Groovy script language. Tests on the implementations show that they can provide all functions of on-demand service without high costs.

## 2. Architecture of on-demand service implementation

Architecture of on-demand service implementation can be shown as figure 2.

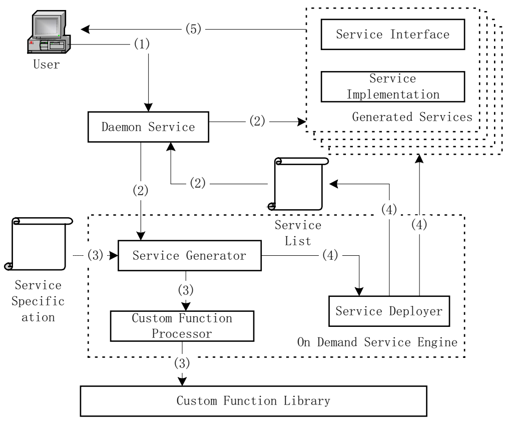


Figure 2. Architecture

The on-demand service architecture is separated into four parts: Daemon Service, On-Demand Service Engine, Custom Function Library and Generated Services. Daemon service is the only entrance of on-demand service node, it is always running and waiting for user's on-demand service requirement which is pre-formatted. (Step 1) As soon as daemon service receives the requirement, it analyzes the requirement, searches whether there is a generated service in system can fill the requirement. If there is none, daemon service invokes the Service Generator to generate the service automatically. (Step 2) Service generator is the kernel of on-demand service implementation. It generates the service codes from service specification given by user automatically, and calls the Custom Function Processor to access the Custom Function Library for custom function requirements. (Step 3) The generated service codes will be transferred to Service Deployer, so that they can be deployed as a service dynamically. (Step 4) The user can invoke the on-demand service generated when the service access point is returned. (Step 5)

Implementations of SSR-ODS and VM-ODS are on the same principle by and large. They are different at how to describe the service and how to generate service codes from service specification. In VM-ODS, because service is described by virtual machine pseudo code, the Service Generator and Custom Function Processor can be implemented simply. But service in SSR-ODS is described by service specification reference, and must be translated into the binary codes, pseudo codes or executable script that can execute on service node. The Custom Function Processor must process the custom function according to the service node environment, too. The generation and deployment of service may relate to code generation automation, just-in-time compile and link, function code automatic download, and so on. It is more complex than VM-ODS.

## 3. Design and Implementation

Java language is one of the most popular programming languages today, especially in network development area. It is simple, safe, robust, platform-independent and object-oriented. Java Virtual Machine (JVM) and Java Runtime Environment (JRE) based on JVM provide a well-defined run-time environment for executing Java pseudo code (Java byte code) on the target node. Based on JVM and JRE, a VM-ODS is designed and implemented.

In order to describe a service, the interface and implementation should be both described. The interface description is separated into grammar description and semantic description. In grid, WSDL [3] describes interface grammar; UDDI [4], WSIL [5] or GRSL [6] can play the role of interface semantic description.

Implementation description describes how to implement a service in logic and arithmetic, it depends on an arithmetic description language or script language. A lot of languages are available for description. For example, familiar script languages such as JavaScript [7], Python [8], and function languages such as OZ language [9], J language [10], which are not so familiar, are all able to play the role of service implementation description language. In this paper, a java based script language named Groovy [11] is selected to describe the service implementation. Groovy is powerful, readable and high-performance on Java platform, and can make it easier to build up our SSR-ODS.

A Java virtual machine based VM-ODS and a Groovy script language based SSR-ODS are designed and implemented on Tomcat 4.1.30 and Axis 1.2 RC2. They share the same Daemon Service and Service Deployer, and deploy services by Axis dynamically. In Groovy script language based SSR-ODS, service implementation specifications are compiled just in time by Groovy script language compiler.

In order to avoid the duplicated deployment of service, a deployed service list with key information of all the deployed services is saved in Daemon Service. When a service is requested, Daemon Service searches in the list first. Only when the service not exists, daemon service starts on-demand service engine, tries to find implementation of the required service, generates service code by service generator, deploys service by service deployer, and returns the generated service interface as response. At the same time, the dynamically deployed service will be stored in the service list so that it can be used future.

In JVM based VM-ODS, service generator find the service implementation which is made up by some Java archive files by given service request. Service generator will get these Java archive files and add them into class path so that they can be deployed as a service. Furthermore, in Groovy script language based SSR-ODS, service implementation is made up by Groovy scripts and script plug-ins. They are downloaded for service generation by service generator and custom function processor when required.

## 4. Test and Performance

In order to prove the validity, practicability and feasibility of the implemented JVM based VM-ODS and Groovy script language based SSR-ODS, some tests are performed. The following are results.

**1). Cost of on-demand service**

The cost of on-demand service is defined as the spent time from the user's request to service's deployment. It depends on the performance of platform, the amount of service specification data and custom functions data, bandwidth of network, etc. We test the JVM based VM-ODS and Groovy script language based SSR-ODS in different network environment with bandwidth 128kbps, 512kbps, 1.5Mbps and 10Mbps. The result is shown as figure 3 and 4.

Assumes that the amount of service specification and custom functions is 10kbytes, 100kbytes and

1Mbytes, we test the cost of JVM based VM-ODS. And assumes that the amount of service specification and custom functions is 2kbytes, 20kbytes and 200kbytes, we test the cost of Groovy script language based SSR-ODS. The test result shows that the cost of on-demand service is less than 10 seconds when the amount of service specification data is not very large. It is acceptable for its advantage of flexibility. When the amount of service specification data is very large and the bandwidth of network is very narrow, the cost may be high. But it is still acceptable for its benefit.

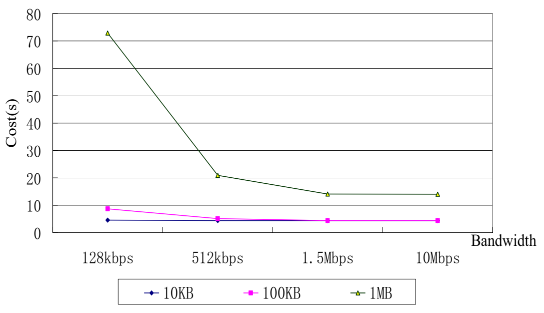


Figure 3. Cost of JVM based VM-ODS

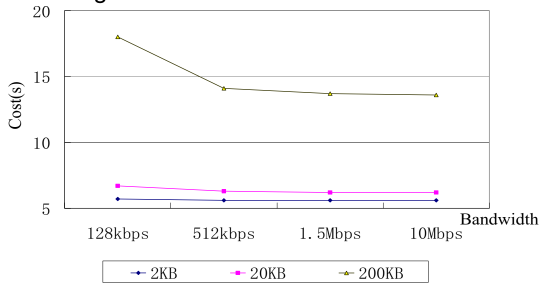


Figure 4. Cost of Groovy based SSR-ODS

Furthermore, comparing between figure 3 and figure 4 shows that cost of Groovy script language based SSR-ODS is higher than cost of JVM based VM-ODS when the amount of service specification data is close. The reason is that Groovy script language based SSR-ODS needs compile the script to Java byte code first when generates the service. It makes the cost higher. But, besides the flexibility advantage of script language, Groovy script language often needs less data than Java byte code for describing the service. So, the Groovy script language based SSR-ODS often outperforms JVM based VM-ODS, especially when the network is not satisfied.

**2). Comparing with the conventional service**

In order to find out advantage of on-demand service, test of comparing JVM based VM-ODS with conventional service is performed. The test compares the data processing service time of JVM based VM-ODS and the conventional service on different network environment with different data amount. By the conventional service, the data is downloaded to the node of service and processed. By the JVM based VM-ODS, the on-demand service is deployed on the near node from data dynamically and process the data nearly. The service time of conventional service is:

$$T_{classic} = T_{download} + T_{process} + T_{overhead} = A_{data} / R_{network} + T_{process} + T_{overhead} \qquad (1)$$

Here, $T_{download}$ is the time for downloading data, in relation to the amount of data $A_{data}$ and the bandwidth of network $R_{network}$; $T_{process}$ is the time for data processing; $T_{overhead}$ is the cost of service, includes the cost of data transmission, XML parsing, etc.

The service time of on-demand service is:

$$T_{ondemand} = T_{process} + T'_{overhead} = T_{process} + T_{overhead} + T_{ondemand} = T_{process} + T_{overhead} + A_{code} / R_{network} + T_{g\&d} \qquad (2)$$

Here, $T_{process}$ is the time for data processing; $T_{overhead}$ is the cost of service, includes the conventional service cost $T_{overhead}$ and the on-demand service cost $T_{ondemand}$. The on-demand service cost can be divided into two parts: service specification download cost $A_{code} / R_{network}$ and service generation/deployment cost $T_{g\&d}$. Comparing with the cost of conventional service:

$$T_{ondemand} - T_{classic} = A_{code} / R_{network} + T_{g\&d} - A_{data} / R_{network} = T_{g\&d} + (A_{code} - A_{data}) / R_{network} \qquad (3)$$

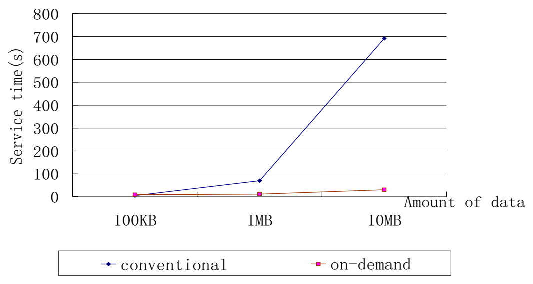
Formula 3 shows that the performance of on-demand service is higher than performance of conventional service when the amount of service specification data is much smaller than the amount of data processed (It's true today) or the network environment is not good so that the service spends a long time downloading data. In the test, the amount of service specification is about 18kbytes. Test is performed in network environment with bandwidth 128kbps, 512kbps, 1.5Mbps, and 10Mbps; the amount of processed data is 100kbytes, 1Mbytes, 10Mbytes, or 100Mbytes. The result is shown as figure 5.

The result shows that performance of on-demand service is better than conventional service usually, except the amount of data to be processed is very small. The advantage is more obvious when the network bandwidth is narrow. The amount of data is larger and the network bandwidth is narrower, the performance of on-demand service is more advantage.

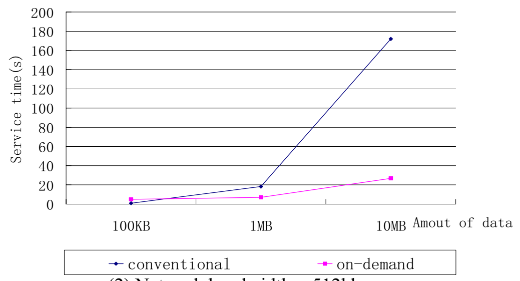
## 5. Conclusion

The grid's final object of providing on-demand computing capability for users on all levels requires more extensibility, more flexibility and higher performance. In grid with service-oriented architecture, the flexibility of service is an important part of the flexibility of grid. In order to provide a more flexible base for grid, on-demand service is purposed in this paper. It tries to offer fit services for user's request. According to the difference of service specification and deployment, on-demand service can be implemented by several strategies, including parameterized custom service (PCS), homogeneous platform based on-demand service (HP-ODS), source-code compatible platform based on-demand service (SCCP-ODS), virtual machine based on-demand service (VM-ODS), and service specification reference based on-demand service (SSR-ODS). In these strategies, the VM-ODS and SSR-ODS are better in flexibility and feasibility.

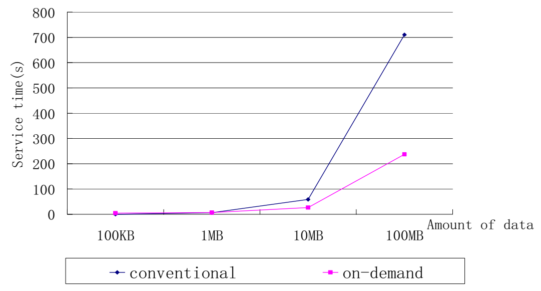
On the base of the analyzation, architecture of on-demand service is purposed. Modules named Daemon Service, Service Generator and Service Deployer cooperate with the service specification to providing on-demand service for users. The on-demand service is designed and implemented on Apache Axis and Tomcat by Java VM and Groovy script language. Tests on on-demand service implemented show that on-demand service can not only provide a flexible service framework without high cost, but also improve the service performance for mass data processing by reduce the data transmission on network. The on-demand service is useful and necessary for the grid's final object of providing on-demand computing capability for users on all levels.



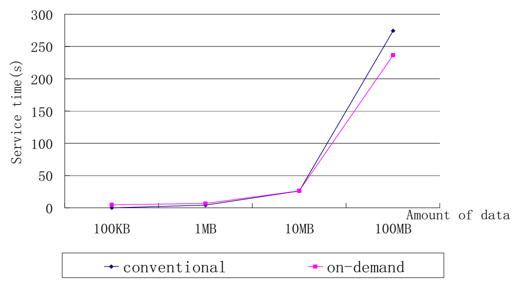
(1) Network bandwidth = 128kbps



(2) Network bandwidth = 512kbps



(3) Network bandwidth = 1.5Mbps



(4) Network bandwidth = 10Mbps

Figure 5. Service time of conventional service and on-demand service

## References

[1] I Foster, C. Kesselman, J. M. Nick, and S. Tuecke: Grid Services for Distributed System Integration. IEEE Computer, 35(6):37.46, 2002.
[2] Globus/IBM Whitepaper, *Modeling Stateful Resources with Web services*
[3] W3C: *Web Services Description Language (WSDL) 1.1*, http://www.w3.org/TR/wsdl
[4] OASIS: *UDDI Specification TC*, http://www.oasis-open.org/committees/tc_home.php?wg_ abbrev=uddi-spec
[5] The W3C Web Services Architecture working group: *Web Services Architecture*, public draft, August 2003. http://www.w3.org/TR/2003/WD-ws-arch-20030808/
[6] ZhenChun Huang, LeiGu, Bin Du and Chuan He: *Grid Resource Specification Language based on XML and its usage in Resource Registry Meta-Service*, 2004 IEEE International Conference on Services Computing, Sep.15-18, ShangHai, China, pp. 467-470
[7] Netscape Inc., *"JavaScript Guide"*, http://wp.netscape.com/eng/mozilla/3.0/handbook/javascript/
[8] M. Lutz, *Programming Python*, O'Reilly, ISBN 1-56592-197-6, 1996.
[9] Gert Smolka, *The Oz Programming Model*, Computer Science Today, Jan van Leeuwen, editor, Lecture Notes in Computer Science, Volume 1000, pp. 324-343, Springer-Verlag, Berlin, 1995.
[10] Jsoftware Inc., *Homepage of J Language*, http://www.jsoftware.com/
[11] Codehaus open-source project repository, *Groovy: a new agile dynamic language*, http://groovy.codehaus.org/