Journal of Southeast University (English Edition) Vol. 24, No. 3, pp. 284 – 288 Sept. 2008 ISSN 1003—7985

Research on web service wrapping for command line programs

Ji Guang[1,2] Han Yanbo[1] Wang Jing[1] Chen Wanghu[1,2]

([1]Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China)

([2]Graduate School, Chinese Academy of Sciences, Beijing 100039, China)

Abstract: A web service wrapping approach for command line programs, which are commonly used in scientific computing, is proposed. First, software architecture for a basic web service wrapper implementation is given and the functions of the main components are explained. Then after a comprehensive analysis of data transmission and a job life cycle model, a novel proactive file transmission and job management mechanism is devised to enhance the software architecture, and the command line programs are wrapped into web services in such a way that they can efficiently transmit files, supply instant status feedback and automatically manage the jobs. Experiments show that the proposed approach achieves higher performance with less memory usage compared to the related work, and the usability is also improved. This work has already been put into use in a production system of scientific computing and the data processing efficiency of the system is greatly improved.

Key words: service-oriented computing; scientific computing; web service wrapping; web service wrapper

In recent years, service-oriented computing[1] has been gaining popularity in scientific computing fields such as bioinformatics, weather forecasting, and astronomy computing, etc. In this computing paradigm, computing power distributed across multiple nodes is presented as services, and can be composed and interacted with each other. For example, the VINCA4Science project[2] conducted at the Institute of Computing Technology of Chinese Academy of Sciences is such a kind of effort. But in common circumstances, the computing power is presented in the form of command line programs, such as BLAST, EMBOSS, and executable scripts, which are hard to incorporate in the service-oriented workflow software[3]. Therefore we need to wrap and transform them into web services, and this process is called service wrapping[4-6].

In the command line user interface, users operate by issuing commands and running the programs. But after the programs are transformed into web services, some issues arise:

1) Web services usually transmit data by SOAP messages, which are unfit to carry large data because the data should be Base64 encoded or escaped. However, data in scientific computing are often huge in amount. For example, the cholera genes stored in the ACE format are of megabytes, and the genes of more advanced species can be of gigabytes or even terabytes. Therefore, service wrapping should be able to take care of large data transmissions.

2) All the message exchange patterns provided by web services do not support continuous text streams, preventing users from monitoring the program status instantly. If the program status is not checked in time, the program may run several hours or days in vain, thus wasting valuable computing resources. Therefore, measures to instantly monitor the running status should be provided.

3) When a program runs, huge temporary data are generated. In the cholera gene assembling experiment, data are generated at a rate of 90 Mbit/s. If the program continues running, a workstation with a 120 GB capacity hard disk will crash in half an hour due to low disk space. Therefore cleaning up the temporary data and managing jobs should be considered.

At present there are already some web service wrapping tools[7-9], but they have not solved the above problems. The web services generated by them are often used to construct web-based user interfaces of the programs, instead of being integrated in a workflow environment.

This paper proposes a web service wrapper, namely Amber (a command-line-program web service wrapper). Amber wraps programs by a novel data transmission and job management approach, enabling fast data transmission and durable operation of the system, and has been applied in a production system together with the VINCA4Science workflow software suit.

1 Software Architecture

The software architecture is shown in Fig. 1. The code generator reads the command line descriptions, generates corresponding web services and deploys them on the web service proxy. The program managers in computing nodes communicate with the web services, and are responsible for running the command line programs.

The web service proxy enables the access from outer networks. It accepts computing requests, and dispatches the requests to the computing node, and returns the results when the job is finished. Therefore, the computing power of the computing nodes is exposed as web services to the outer networks.

The code generator generates web service interfaces for command line programs. The user writes command line descriptions for different programs and uses the code generator to generate different web service codes and deploys them on the web service proxy.

The program manager supports the communication between the web service proxy and computing nodes, and runs the command line programs. It is deployed at individual nodes, and provides an infrastructure on which the command line program can run.

Received 2008-04-15.

Biographies: Ji Guang (1984—), male, graduate; Han Yanbo (corresponding author), male, doctor, professor, yhan@ict.ac.cn.

Foundation items: The National Natural Science Foundation of China (No. 60573117), the National Basic Research Program of China (973 Program) (No. 2007CB310805).

Citation: Ji Guang, Han Yanbo, Wang Jing, et al. Research on web service wrapping for command line programs[J]. Journal of Southeast University (English Edition), 2008, 24(3): 284 – 288.

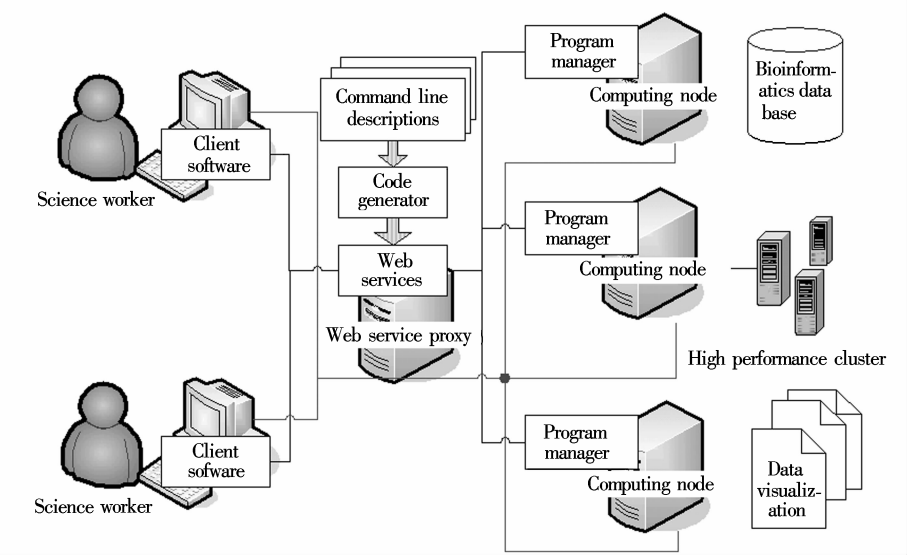

Fig. 1 Software architecture

In this software architecture, the selection of client software is flexible. It can be a common web service client, or a piece of workflow software supporting web services such as VINCA4Science.

2 Key Techniques

The above architecture is a basic framework of web service wrapping for command line programs. But as stated above, Amber should support fast data transmission between the web services; it should also transmit the program screen messages instantly, enabling science workers to monitor the status; it needs to provide job life cycle management, maintaining a stable and durable job running environment.

2.1 Proactive data transmitting

We abstract the program as a function:

$$y = f(x)$$

where x is the input data, and y is the output data. Given a computing process composed of programs f_0 and f_1, which are deployed in node 0 and node 1, respectively. The input of f_0 is x_0, and the output of f_0 is the input of f_1. So we have

$$x_1 = f_0(x_0),\ x_2 = f_1(x_1)$$

where x_2 is the ultimate result.

Web services bound with HTTP protocol follow the request-response pattern, in which the input data is transmitted to the web service provider by request, and the output data is transmitted to the web service requester by response. This pattern is illustrated in Fig. 2. The request sent to node 0 contains the input data x_0, and the response from node 0 contains x_1, which is the computing result of f_0. We use size(x) to denote the size of data x. It follows that the whole data size transmitted is

$$\text{size}(x_0) + 2\sum_{i=1}^{n}\text{size}(x_i) + \text{size}(x_{n+1})$$

where $x_1, \ldots, x_n$ are temporary results. Were the process to contain many computing programs, there would be significant overhead.

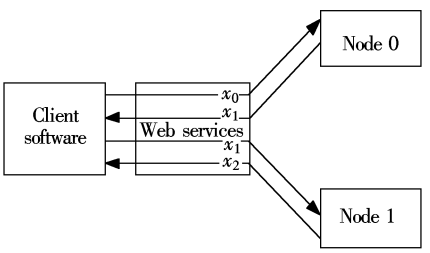

Fig. 2 Transmitting data by SOAP

If we make the data downloadable, and the request contains the data address rather than the data, the story will be different. The web service will download the data proactively, instead of receiving the data reactively. We call this the proactive transmission approach, which is illustrated in Fig. 3. The bold lines with arrows denote data transmission. In this approach, if there are $n+1$ computing programs $f_0, f_1, \ldots, f_n$, and each program's output is the next one's input, the total data amount transmitted between client software and web services is

$$\text{size}(x_0) + \text{size}(x_{n+1})$$

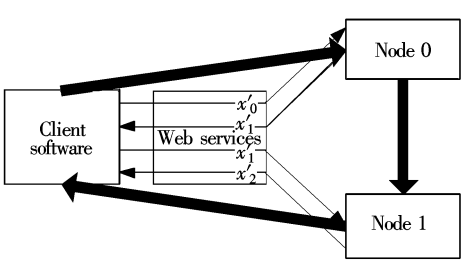

Fig. 3 Transmitting data by proactive approach

In other words, it is the summed size of initial data and ultimate result data, and is irrelevant to temporary results. This significantly improves data transmission efficiency.

In order to implement proactive data transmission, the web service should make the output data downloadable. On the client side, if a job needs input files, the client software should supply the file addresses. When the program manager receives the addresses, it will download the files to the job's

working directory. When the program terminates, the program manager generates the result files' download addresses, and sends them back to the client software by web services. When the client software receives the addresses, it can download the files or pass them on to the next job as input, thus avoiding transmitting the whole data to the client software.

2.2 Instant message transmission in the web framework

Instant message transmission means that as soon as the computing program generates the message, the message should be transmitted to the client software. But the request-response pattern of the web service makes it difficult: the message can only be transmitted after the client has requested it. Therefore, implementation by web services is not feasible.

But we can still implement that function in the web framework. We make the web service proxy provide an HTTP service, which communicates with computing nodes by internal protocol and receives the screen output messages instantly. When a client issues request for a particular job, the HTTP service connects to it; as soon as text messages arrive, the HTTP service transmits them to the client. Fig. 4 illustrates this approach.

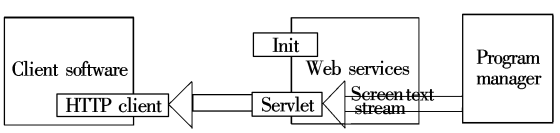

Fig. 4 Instant message delivery under the web framework

To implement this approach, there is a Servlet (which is a lightweight Java-implemented HTTP server) running on the web proxy. There is no data buffered in the Servlet. As soon as a new message arrives, it is forwarded to the client by the Servlet. When the client software gets the job ID (e.g., job1) by calling init, it can send the HTTP request to the Servlet (e.g., request for /job1). When the Servlet responds, it directs the screen output messages to the client software. Whenever there is text on the screen output stream, it will be sent to the client software. Therefore, the transmission is instant. This approach makes up for the disadvantage of web services and provides a solution under the web framework.

2.3 Job life cycle management

The job mentioned above is a scientific computing process performed on a computing node submitted by a science worker. In order to implement automatic cleanup of jobs, we analyze the life cycles of the jobs.

The events which appear in a job are shown in Tab. 1. There are dependency relationships between these events, and, therefore, we use a state machine to illustrate them in Fig. 5. The states are listed in Tab. 2.

The job will not be cleaned up until it reaches the Terminate state. If we permitted jobs not to reach the Terminate state, the files produced by the jobs would pile up in the computing nodes, which would fail in a short time due to low disk space. We cannot ensure that the science workers end their actions on the jobs with cleanup or abort. Therefore, we introduce the disposable state and timeout/hold action.

Tab. 1 Actions on a job's life

Name	Contents
Init	Setup working directory
Input	Download needed files
Start	Start computing program
Suspend	Suspend the program
Abort	Abort and clean the job
Exit	The program exits
Cleanup	Clean the working directory
Timeout	Timer times out
Hold	Reset the timer
Dispose	Forced cleanup

Tab. 2 Job states

State		Description
Initial state		The job does not exist
Live	Allocated	Working directory allocated
	Ready	Input files downloaded
	Running	Program running
Terminated		Program terminated
Disposable		Job may be disposed at any time
Final state		Job disposed

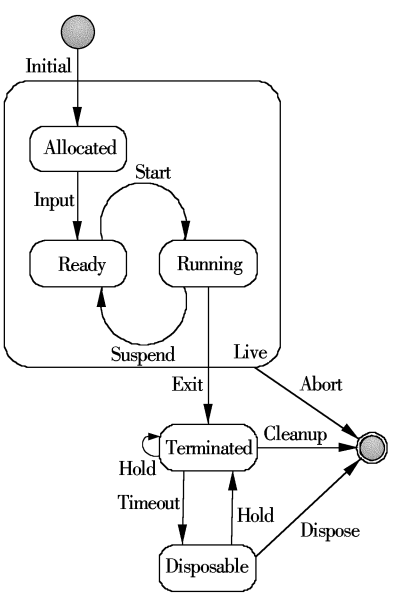

Fig. 5 A job's life cycle

If the job stays in the Terminate state for a given period, the timer triggers the Timeout action, and the job will be transferred to the disposable state. When the disk space is low, the dispose action will dispose of the job and make it reach the Terminate state. In this model, the job will eventually reach the Terminate state as long as there is a finite number of actions.

The client software calls a hold operation for every given period while downloading files in the Terminated state, thus avoiding the job to transfer to a disposable state. After the files have been downloaded, the client software calls a cleanup operation to clean the job up. Even if the cleanup operation is not called, the program manager will take action and make the job disposable and dispose of it eventually.

3 Use Cases

Amber has been deployed and used in a bioinformatics research institute. Fig. 6 shows the process diagram generated by Vinca4Science with the help of Amber. The rectangles in the figure are programs wrapped by Amber. These programs are distributed in four computing nodes: the node where the upper-left retrieve reference sequence program resides has a large gene sequence database, providing gene reference sequences; the node where the upper-right Retrieve Solexa reads program resides is connected to gene sequencing machines, providing the original data transferred from Solexa sequencing machines. These two programs provide data for the whole process. The lower-right Consed ace program resides on a machine with large memories, and visualizes the gene assembly results in ACE format for the science worker to facilitate manual adjustment. The other programs run on a high-performance cluster.

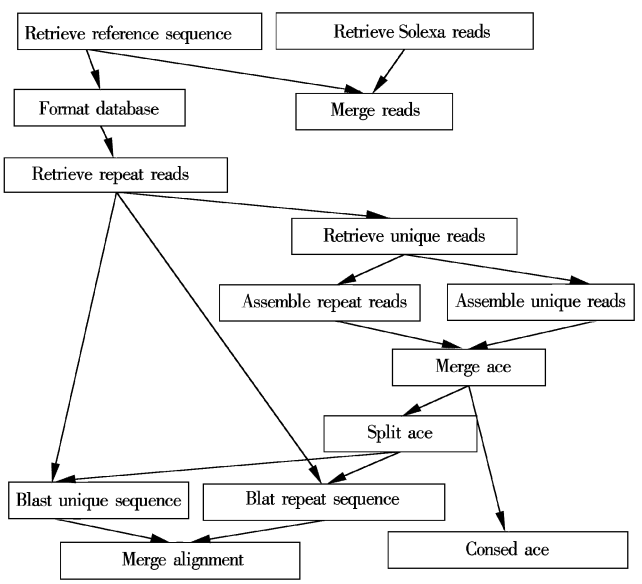

Fig. 6 Process diagram constructed by VINCA4Science

The black lines in the diagram denote the data file transmission path. Double click the format database and you will see the view shown in Fig. 7. The true value in the connection option on the left column shows that the input data are from the previous program's output data. The three rows in the right column show that the computing process generates three output files, available for subsequent programs.

Fig. 7 Process configuration

The process utilizes different programs from multiple nodes, and passes data among them. The data are transmitted automatically without human interference, and efficiency is improved.

4 Related Work and Comparisons

Soaplab[7] is a web service wrapper developed by European Bioinformatics Institute. Its users first describe the command line programs by the ACD language, and feed the descriptions to Soaplab to generate web services for the executable files. The host providing web services communicates with computing nodes in order to submit jobs and to query job status. Opal[8] is developed by San Diego Supercomputer Center. Its significant difference from Soaplab is that it does not generate web service codes from description files, but uses information from configuration files directly. In the meanwhile, it can conveniently integrate grid resources.

The generic factory service[9], developed at Indiana University, is based on Globus Toolkit. It requires that all data files be downloadable by Globus Toolkit and be marked with URLs. The web service is responsible for transmitting URLs instead of data files, thus improving efficiency. But its functioning depends on Globus, which is inflexible. Tab. 3 is a qualitative comparison among the related work and Amber.

Tab. 3 Comparison on related work

Feature	Soaplab	Opal	GFS	Amber
Data transmission efficiency	Low(by SOAP)	Low(by SOAP)	High(by Globus)	High
Instant message delivery	No	No	Weak	Yes
Job life cycle management	No	No	No	Yes

Soaplab and Opal transmit data by SOAP messages, and perform in a similar way. As mentioned above, this approach is low in efficiency and consumes many system resources. The latter's user manual explicitly requires the user to set the maximum heap size of the JVM to 1 GB. Experiments show that if the file size exceeds the maximum heap size, the JVM will crash. The GFS uses a StageIn function of GRAM to transmit files, and performs similarity to Amber. We perform a set of experiments to show the performance difference between these two approaches. The experiments transmit a set of binary files of 10 to 60 MB size, and measure the file transmission time and JVM heap size in use. The web server in the experiment has a Pentium M 740 CPU, 2 GB memory, and has Sun JVM version 6.0 and Tomcat 6.0 + Axis 1.4 installed. The client software is deployed on the same LAN, which connects the host via a 100 MB Ethernet switch. The results are shown in Fig. 8 and Fig. 9.

The experiments show that the proactive transmission approach has a significant advantage. The data transmission time is shorter because it avoids encoding the data by Base64 encoding. The heap in use is much smaller because it

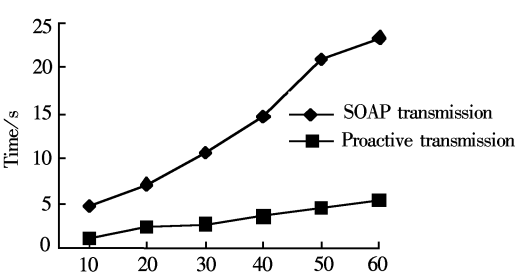

Fig. 8 Comparisons on data transmission time

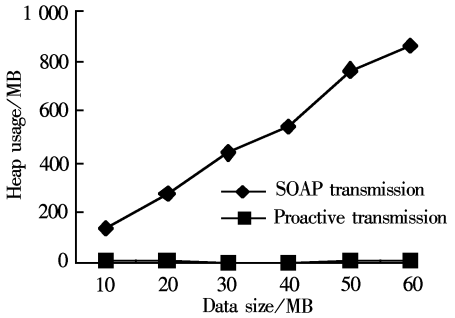

Fig. 9 Server side JVM heap size in use

writes the data directly to the disk instead of storing it in memory. The GFS relies on GRAM. Although it performs similarity to Amber, it requires that all files be local or accessible by GridFTP. However, Amber supports FTP, SFTP, HTTP and other common file transferring protocols, and it is more flexible.

It should be made clear that if the client is behind the firewall, there will be extra work to send out data. In this situation, the client software should forward the data to the outer network where it can be downloaded, and include the URL in the SOAP message. Besides, a proactive transmission approach requires that the client and computing node provide file downloading service. This may cause some management overhead, but it is worth that due to significant performance improvement.

Neither Soaplab nor Opal can provide instant message delivery. The GFS can query job status by a self-defined WS-messenger, but it cannot provide a continuous message text stream. The instant message delivery of Amber solves the problem, and all the related work does not provide automatic job life cycle management. Users should clean up expired jobs manually. Soaplab and Opal provide cleanup scripts (both available in http://soaplab.sourceforge.net/soaplab1/ToDo.html and http://nbcr.net/services/opal/), which need to be executed by the administrator. The web services generated by Amber can perform cleanup automatically.

5 Conclusion and Future Work

This paper introduces a web service wrapper for command line computing programs in a multiple node environment. It facilitates fast data transmission among computing nodes by an improvement on the file transmission approach, and enables the system to continue running without outer interference by job life cycle management.

We are now continuing to improve Amber, especially regarding security features. At present the web service proxy uses HTTPS to provide a simple authentication mechanism, and we are going to enhance it. We are collaborating with bioinformatics institutes closely, making the software adapt to different computing environments and become a useful software tool in science computing environments.

References

[1] Singh M P, Huhns M N. *Service-oriented computing: semantics, processes, agents* [M]. New Jersey: Wiley, 2005.

[2] Wang J, Han Y, Yan S, et al. VINCA4Science: a personal workflow system for e-science [C]//*Proceedings of International Conference on Internet Computing for Science and Engineering*. IEEE Computer Society Press, 2008: 444 - 451.

[3] Tiwari A, Sekhar A K T. Workflow based framework for life science informatics [J]. *Computational Biology and Chemistry*, 2007, **31**(5/6): 305 - 319.

[4] Canfora G, Fasolino A R, Frattolillo G, et al. A wrapping approach for migrating legacy system interactive functionalities to service oriented architectures [J]. *Journal of Systems and Software*, 2008, **81**(4): 463 - 480.

[5] Gannon D, Alameda J, Chipara O, et al. Building grid portal applications from a web service component architecture [J]. *Proceedings of the IEEE*, 2005, **93**(3): 551 - 563.

[6] Li M, Qi M. Leveraging legacy codes to distributed problem-solving environments: a web services approach [J]. *Software—Practice and Experience*, 2004, **34**(13): 1297 - 1309.

[7] Senger M, Rice P, Oinn T. Soaplab—a unified Sesame door to analysis tools [C]//*Proceedings of UK e-Science All Hands Meeting*. Nottingham: EPSRC, 2003: 509 - 514.

[8] Sriram K, Brent S, Karan B, et al. Opal: simple web services wrappers for scientific applications [C]//*Proceedings of IEEE International Conference on Web Services*. IEEE Computer Society Press, 2006: 823 - 832.

[9] Kandaswamy G, Fang L, Huang Y, et al. Building web services for scientific grid applications [J]. *IBM Journal of Research and Development*, 2006, **50**(2/3): 249 - 260.

命令行程序的 web 服务化问题研究

季 光[1,2] 韩燕波[1] 王 菁[1] 陈旺虎[1,2]

([1] 中国科学院计算技术研究所,北京 100190)

([2] 中国科学院研究生院,北京 100039)

摘要:提出了一种科学计算领域中常用的命令行程序 web 服务包装方法. 首先给出了实现一个基本的 web 服务包装器的软件架构,并对主要组件的功能进行了解释. 进而经过对数据传输和作业生命周期的深入分析,设计了一种新颖的文件传输和作业管理机制用以增强该软件架构,使得在命令行程序被包装成 web 服务的同时,保证了文件的高效传输,运行状态的即时反馈和作业的自动管理. 对比实验表明,该方法在使用了较少内存的前提下提高了程序运行性能,定性比较显示程序的易用性也有所提高. 本工作已经在实际的科学计算系统中得到了应用,极大地提高了系统的数据处理效率.

关键词:面向服务的计算;科学计算;web 服务化;web 服务包装器

中图分类号:TP311