Astronomical Data Analysis Software and Systems XIII
ASP Conference Series, Vol. 314, 2004
F. Ochsenbein, M. Allen, and D. Egret, eds.

A Prototype toward Japanese Virtual Observatory (JVO)

Masatoshi OHISHI, Yoshihiko MIZUMOTO, Naoki YASUDA[1], Yuji SHIRASAKI, Masahiro TANAKA, Satoshi HONDA

National Astronomical Observatory of Japan, 2-21-1, Osawa, Mitaka, Tokyo 181-8588, Japan

Yoshifumi MASUNAGA[2]

Ochanomizu University, 2-1-1, Otsuka, Bunkyo, Tokyo 112-8610, Japan

Ken MIURA[3], Hirokuni MONZEN, Kenji KAWARAI, Yasuhide ISHIHARA, Yasushi YAMAGUCHI and Hiroshi YANAKA

Fujitsu Ltd., 1-9-3, Nakase, Mihama, Chiba 261-8588, Japan

Abstract. We developed the first prototype toward a Japanese Virtual Observatory (JVO) by using the Globus Tool Kit 2 (GTK2). We found that the system worked as we had expected, including the functionality of JVO Query Language. However, it took a very long to initiate each Grid process. Thus we replaced the GTK2 with a tool distributed by NSF Middleware Initiative, and shortened the polling interval of the job-manager from 30 seconds to 3 seconds. As a result, the serious problem was partially resolved, and the elapsed time for a query reduced to about half of the previous one.

1. Introduction

The National Astronomical Observatory of Japan (NAOJ) operates the Subaru telescope in Hawaii and large radio telescopes in Nobeyama. All the observed data are digitally archived and are accessible via internet. The radio telescopes of Nobeyama produce about 1 TByte per year, and the Subaru telescope outputs about 20 TBytes per year. Because astronomical objects radiate electromagnetic waves in a wide frequency range, it has been recognized that multi-wavelength analyses are essential to understand the physical and chemical behavior of galaxies, stars, planets and so on.

JVO is designed to provide seamless access to federated databases and data analyses systems for astronomers by utilizing the state-of-the-art GRID tech-

[1]Institute for Cosmic Ray Research, University of Tokyo

[2]National Astronomical Observatory of Japan

[3]National Institute of Informatics

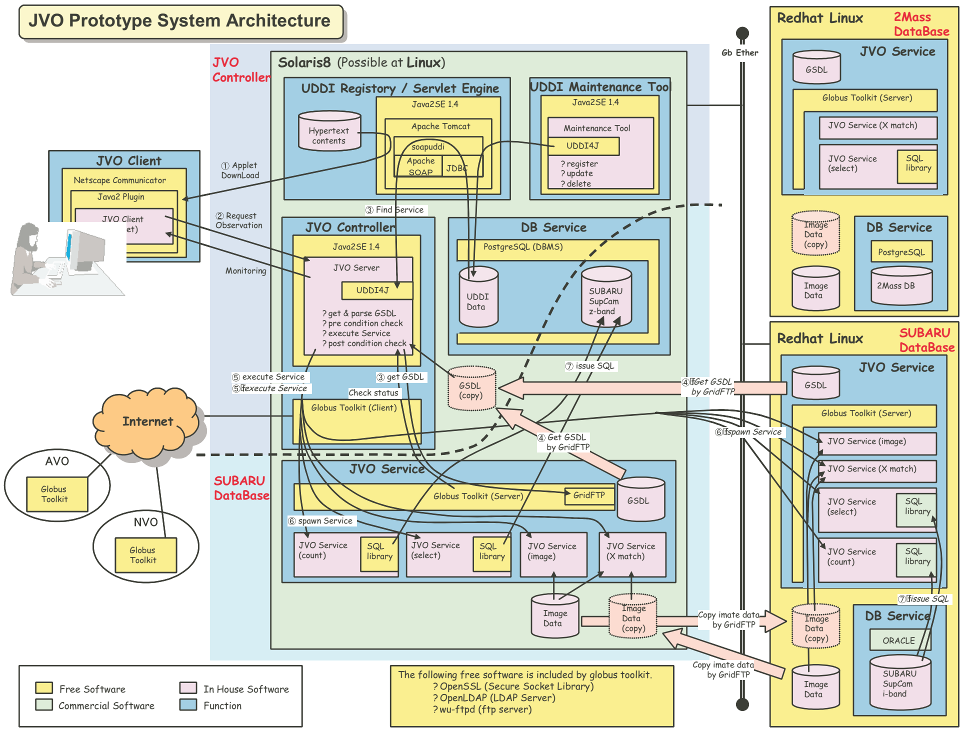

Figure 1. Architecture of JVO prototype. Note that the prototype has not been connected to other VOs yet.

nology through the 10 Gbps SuperSINET (`http://www.sinet.ad.jp/english/`) which was installed in 2002. The basic concept and a new query language to access to the distributed databases, JVO Query Language, are already described by Mizumoto et al. (2003).

This paper describes in detail the implementation and assessment of the first prototype toward JVO.

2. Implementation of the JVO Prototype

We implemented the first prototype in a closed subnet in NAOJ. The architecture of the JVO prototype is shown in Figure 1. We adopted the Globus Toolkit 2 for the prototype. However we also take into account the Web service concept which is included in the OGSA.

Here we describe how the prototype works. First of all, researchers provide the JVO with simple instructions, described by using JVOQL, how they want to perform their "Virtual Observation" through the JVO portal. The JVO portal interprets them and generates a "work-flow" by consulting the UDDI servers to find where available JVO services are registered.

Based on the work-flow, built-in or user-defined services are called sequentially by the JVO controller. Prior to command execution the JVO controller issues a "pre-condition check" to make dynamic assignment of distributed resources according to their availabilities. When one step of the work flow is finished, the result is examined by "post-condition check" to determine whether the step finished successfully or not. If the step finished successfully, the JVO controller generates the next step(s) of the work flow and executes them. If the step finished unsuccessfully, the JVO controller searches for an alternative server which provides the same service, and executes the same step on that server, if available. Successful execution results of the work flow are transferred from remote servers to the JVO controller through GridFTP, and are presented to the researchers by the JVO client.

It is a very important service in the JVO to cross-match (X-match) query results from multiple wavelength data. Each query is sent from the JVO controller to an appropriate database server. Then the smallest query result is GridFTPed from the server to another server with the next smallest result. The recipient server is asked to run its X-match engine, and the result is further GridFTPed to a server with the third smallest query result. The final result is GridFTPed to the JVO controller.

3. Assessment of the Prototype

We used several JVOQLs to assess this prototype. Table 1 contains each step of the work flow and elapsed time for each step.

Table 1. A sample of work flow and elapsed time

Step #	Host	Command	Elapsed Time
0	mizu-g	JVOQLparser.sh	1' 12"
0.0	mizu-g	jvo-query.sh	1' 15"
0.1	minazuki-g	jvo-query.sh	1' 09"
0.2	mizu-g	Scheduler.sh	1' 14"
0.2.0	mizu-g	jvo-query.sh	1' 15"
0.2.1	minazuki-g	post-xmatch.sh	1' 33"
0.2.2	mizu-g	jvo-query.sh	1' 21"
0.2.3	minazuki-g	jvo-query.sh	2' 26"

Table 1 contains several commands described as shell scripts: JVOQLparser.sh reads input JVOQL script and parses into individual queries in SQL; jvo-query.sh issues individual queries to database servers, counts up database records hit, and cuts images out from image databases; Scheduler.sh collects count results and determines the order to request database servers query results and image data; and post-xmatch.sh kicks off the cross-match engine. These commands were submitted by using the GRAM service of the Globus Tool Kit 2. As we expected, all steps were generated automatically, and we could get results successfully.

We examined the robustness of our prototype by forcing the system to issue a command, at step 0.2.2, which would fail at one server but succeed at another server. At first the issued command to the "wrong" server failed, but then the system reissued the same command to the "right" server through dynamic generation of the work flow.

However we found the elapsed times were too long for all steps. We knew that an elapsed time for each command was less than a few seconds when it was issued in a non-globus environment. It should be noted that the final step, 0.2.3, corresponds to cutting out images and needs a very long CPU time even in a non-globus environment. Such very long elapsed times seemed to be due to the authentication process and the "globus-job-run" command of the Globus Tool Kit 2. It is well known that the authentication process takes nearly 10 seconds and the "globus-job-run" command takes long during its initial hand-shaking procedure before issuing a "real command". Since JVO is a pseudo-real-time system, it was crucial to shorten such large overhead in each process.

4. Improvement of the Prototype

We introduced the NSF Middleware Initiative (NMI)[1] to accelerate the slow authentication process in the Globus Tool Kit 2, because NMI provides a binary module for the authentication. Then we analyzed the source code of the "globus-job-run", and found that the polling interval was fixed to 30 seconds. Therefore we modified the polling interval to 3 seconds, and recompiled the tool kit. As the result the elapsed times in Table 1 were shortened by more than a factor of 2. For example the elapsed time for step 0 became 20 - 25 seconds. Although we succeeded to accelerate all steps in our prototype, the elapsed times are much longer compared with those for cases in non-globus environment. Thus it is necessary to investigate further, for example, the source code of the tool kit to make our system to run much faster.

5. Summary

We constructed the first version of the JVO prototype based on GTK2, and confirmed that the JVOQL has sufficient functionality to access federated databases. We found that the prototype worked as we had expected, however, it took a very long to initiate each Grid process. Thus we replaced the GTK2 with a tool distributed by NSF Middleware Initiative, and shortened the polling interval of the job-manager from 30 seconds to 3 seconds. As a result, the serious problem was partially resolved, and the elapsed time for a query became less than half compared with the previous one.

Acknowledgments. This research was supported by Grant-in-aid "Information Science" carried out by the MEXT (14019092 and 15017289).

References

Mizumoto, Y., et al. 2003, in ASP Conf. Ser., Vol. 295, ADASS XII, ed. H. E. Payne, R. I. Jedrzejewski, & R. N. Hook (San Francisco: ASP), 96.

---

[1] `http://www.nsf-middleware.org/`