XVI IMEKO World Congress
Measurement - Supports Science - Improves Technology - Protects Environment ... and Provides Employment - Now and in the Future
Vienna, AUSTRIA, 2000, September 25-28

PERFORMANCE ANALYSIS OF DISTRIBUTED MEASUREMENT SYSTEMS

J. Pekala, B. Kasprzak and J. Mroczka
Chair of Electronic and Photonic Metrology,
Department of Electronics, Wroclaw University of Technology
ul. B.Prusa 53/55, 50-317 Wroclaw, Poland

Abstract: Over the last years, standard network protocols and technologies have become very popular basis for distributed measurement system design. This paper describes a set of distributed measurement experiments designed with the use of selected network technologies. The performed test results give us an opportunity to compare these technologies and consider how particular technology affects the distributed experiment performance. We also discuss some important factors that influence performance. Finally, we give some advice for developers that we hope will help them to choose the best design methodology.

Keywords: distributed system, network technology, performance test

1 INTRODUCTION

As networks of computing resources and the usage of programmable instruments have become prevalent, the concept of distributing measurement tasks among multiple network nodes has become viable and desirable [1]. Over the past few years, several network technologies have evolved (DCE RPC, WWW/CGI, Client-Server paradigm, and the concept of middleware, Java, distributed objects to name only a few). Many of the above concepts have been adopted for distributed measurement systems designing [2-7]. Due to the great popularity of the Internet, and the extensive usage of the WWW in particular, the client-server architecture and WWW related technologies have gained widespread acceptance. On the other hand distributed objects architectures (Java RMI, DCOM, CORBA) offer a number of significant benefits and lead to drastic reduction of development effort. For example, an important benefit of using CORBA or RMI is the availability of Naming Service that can be used by client application to look up server application. Which network technology and design methodology should be chosen then? It usually depends on application specific requirements that often force developers to consider system performance.

It is obvious however, that performance of remotely controlled instrumentation is reduced by network-imposed overhead and depends on several factors e.g. network topology and traffic, number of concurrent tasks, instrument locking mechanism and design methodology used itself. Important question is what influence the particular network technology has on system performance. Another problem is what performance metric should be taken. Although some performance investigations have been reported [8,9] performance testing and analysis of existing and designed distributed measurement systems is still very challenging task.

2 SYSTEM ARCHITECTURE

We have assumed that all instrument resources are available by a unified network protocol. Different protocols can be taken into consideration both standardised and proprietary (e.g. Maritime Information Technology Standard). This article focuses on VXI-11 TCP/IP Instrument Protocol Specification that is authored by the VXIbus Consortium. This specification uses the ONC remote procedure calls (RPC) model and describes how instrumentation can be connected to industry standard networks. The RPC specification is among the most widely implemented in the industry, providing consistent behaviour across heterogeneous execution environment. Seventeen messages are defined in VXI-11 specification (e.g. device_read/_write and device_lock/_unlock). These messages are expected to be supported by devices that claim to be network instrument protocol compliant. The Instrument Protocol formulates a classical client/server architecture where the client identifies and communicates directly with the server. In our tests we have used different development environments that can be thought as implementation of RPC protocol (HP SICL, VISA, Linux RPC, Java RPC, HP VEE). If such approach lacks of some functionality it is possible to introduce the concept of ‘middleware’, that assumes a functional layer between the client and server. This layer may

provide additional services such as location of instruments resources and allow clients to interact with generic abstraction of a server rather than with specific host and/or process. Different protocols may be used for communication with the middleware layer like HTTP (CGI and Servlets) or RMI.

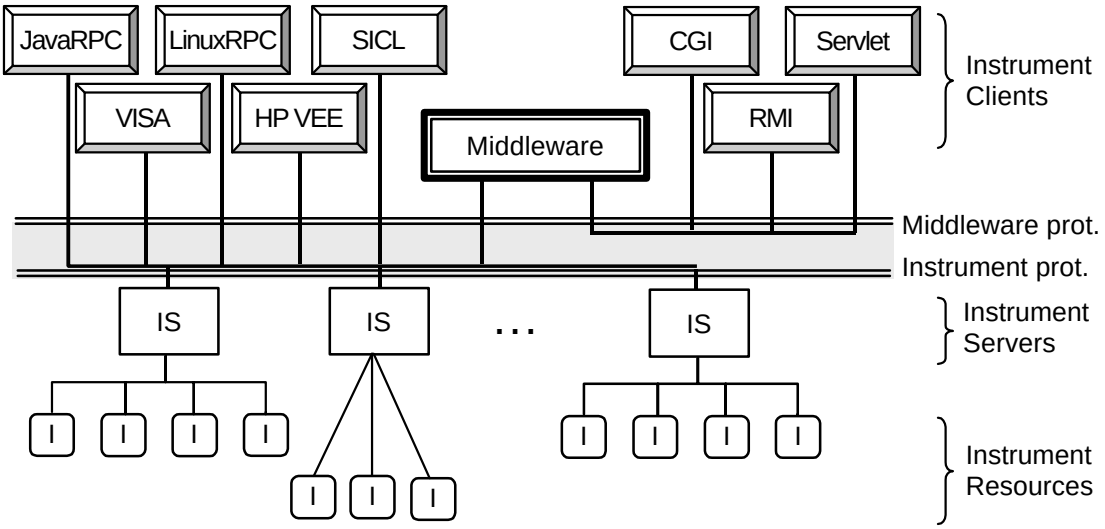

Figure 1. VXI-11 based distributed system architecture

3 OVERVIEW OF SELECTED TECHNOLOGIES

Various implementations of RPC protocols have been developed. For instrument client and CGI programs we applied standard C/RPC libraries from Linux RedHat 6.1 distribution.

VISA (Virtual Instrument Software Architecture) is a standard I/O library that provides an open multivendor foundation for instrumentation software. HP VISA library supports different kind of interfaces including the so-called VISA over LAN. Two LAN networking protocols are provided with the VISA software and one of it is TCP/IP Instrument Protocol.

HP SICL (Standard Instrument Control Library) is a very similar to VISA instrument communication library and uses the same concept and protocols to control instruments over LAN with slightly different addressing notations.

HP VEE (Hewlett-Packard Visual Engineering Environment) is visual programming language optimised for building test and measurement applications. Programs are constructed by connecting icons together on the screen that resembles a block diagram. HP VEE provides a big number of objects that allow controlling instruments, taking measurements, acquiring and processing data, displaying results and generating reports.

Thanks to its platform neutrality and object-oriented design Sun's Java has gained a great deal of acceptance. Two design techniques relying completely on Java we have applied for writing applications controlling remote instrumentation: a Java implementation of RPC protocol and RMI. The Java RPC we have used for writing both client programs that directly interact with instrument servers and servlets that communicate with instrument servers in response to HTTP client request. Moreover all HTTP client programs were written in Java language.

The Java Remote Method Invocation (RMI) system allows an object running in one Java Virtual Machine (VM) to invoke methods on an object running in another Java *VM*. RMI applications are often comprised of two separate programs: a server and a client. RMI provides the mechanism by which the server and the client communicate and pass information back and forth. Such an application is sometimes referred to as a distributed object application.

The CGI interface provides a simple, easy to use method of executing programs from within an HTTP server. This mechanism can be used to interface with services outside the WWW server. The CGI interface has many advantages, including portability between server software, and a large base of programs and development tools for its use. CGI has some drawbacks too. The CGI requires the server to execute a program, an operation that is very expensive to the server's resources. A second limitation of the CGI is its stateless nature. When a CGI program is accessed by a client, a new copy of CGI program is invoked. If a program must access an external resource (such as a communication link to Instrument Server), it must continually close and re-open that resource.

Servlets are modules that extend request/response-oriented servers. Servlets can be embedded in many different servers because the servlet API assumes nothing about the server's environment or protocol. Servlets have become most widely used within HTTP (WWW) servers. Although servlets use the same means of communication just as a CGI does, there are important differences as well. One major difference is that for each request to a servlet, a lightweight thread is spawned to handle that request as opposed to starting a new process. Another difference is that a servlet thread does not have to terminate once it has sent its response. It is even possible to specify that the server should be loaded during the server's start-up like a service. It makes servlets much more flexible because a servlet can make connections to instrument servers, create links to some instrument and then wait for client request that points at one of the published operations. Avoiding creating a new link to the particular instrument for every request gives an opportunity to design servlet interface at different level of abstraction. For example, one servlet may accept a set of requests that are similar to messages defined in VXI-11 specification, thus acting as a protocol converter. Another one can perform the whole experiment that may require a set of instrument resources and may last for a long time.

4 PERFORMANCE BENCHMARKS

Each instrument requires specified messages for taking measurements and acquiring data, which are specific type and length. A particular application controls a given number of instruments and requests instrument's operation at a certain rate. There are so many possibilities especially in distributed environments that instead of designing a number of complete experiments that it would be possible to use as a performance benchmarks we decided to prepare a couple of very short and simple programs. These programs consist mainly of diverse instrument operations that - we believe - are representative for a large number of applications.

To analyse performance, we need to define its meaning in distributed instrumentation context. We assumed very straightforward concept: as a performance, we measure an average time taken by client to complete a certain number of instrument operations. All tests were carried out only in LAN environment. We have used two variants of instrument server: HP E2050 LAN/HP-IB gateway and instrument server utility working on PC equipped with GPIB interface card. As a client station, we have used standard PC under Windows 98 and Linux RedHat 6.1 operating systems.

A simple DC measurement (HP 34401A multimeter) was executed several times in the first test. The messages in this test are rather short and typical for wide range of instruments, especially when they conform to the SCPI specification (e.g. SCPI query: "meas:volt:dc? 5.0,0.001"). Test results presented in figure 2 shows that in such conditions CGI and servlet implementations influence performance significantly.

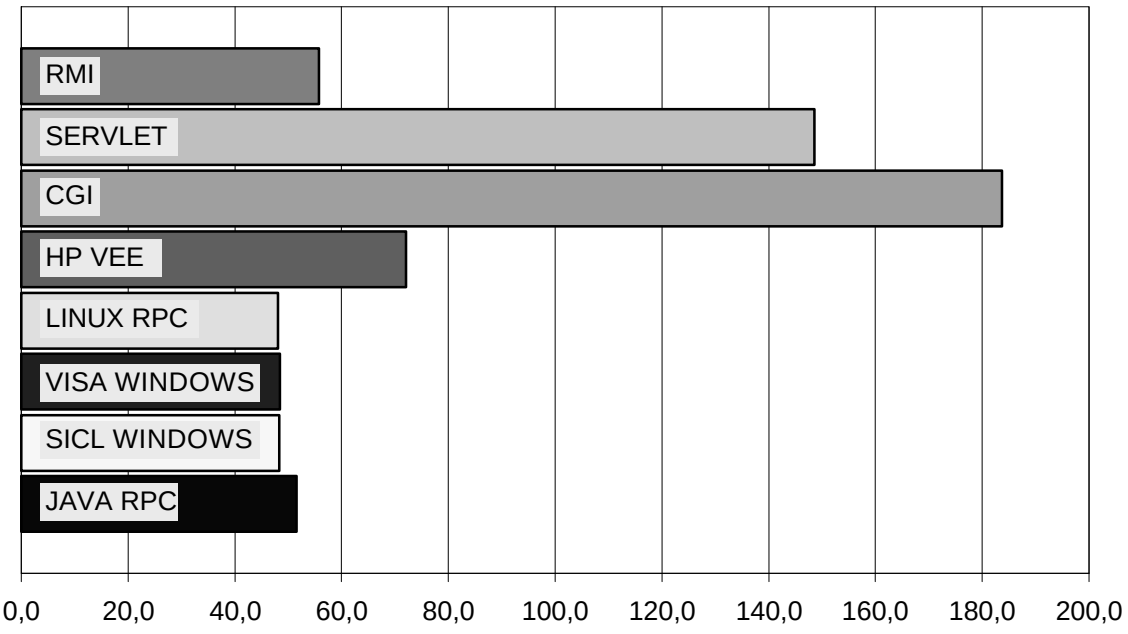

Figure 2. The first test execution time in ms.

The second test consists of operations that control HP 33120A Function/Arbitrary Waveform Generator. In contrast to the first test, this one, apart from messages strictly related to the instrument, also contains messages that open/close connection to the remote server, create/destroy link to signal generator and lock/unlock it. Another important difference is that one of the messages consists of more than 2000 bytes. Data blocks of such size are typical for instruments like spectrum analysers and digital oscilloscopes. The obtained results are presented in figure 3. If we compare them to the results obtained from the previous test we will notice far smaller differences. It is caused by fact that in this test much more instrument operations are executed. A rather unexpected result that we get for VISA implementation in this test is caused by the function, which opens the instrument session. If we plan a session that will last for a long period and contain many instrument operations, it is not a big problem. However, if we have an application, in which an instrument is used several times for a short period of time it may be a big disadvantage.

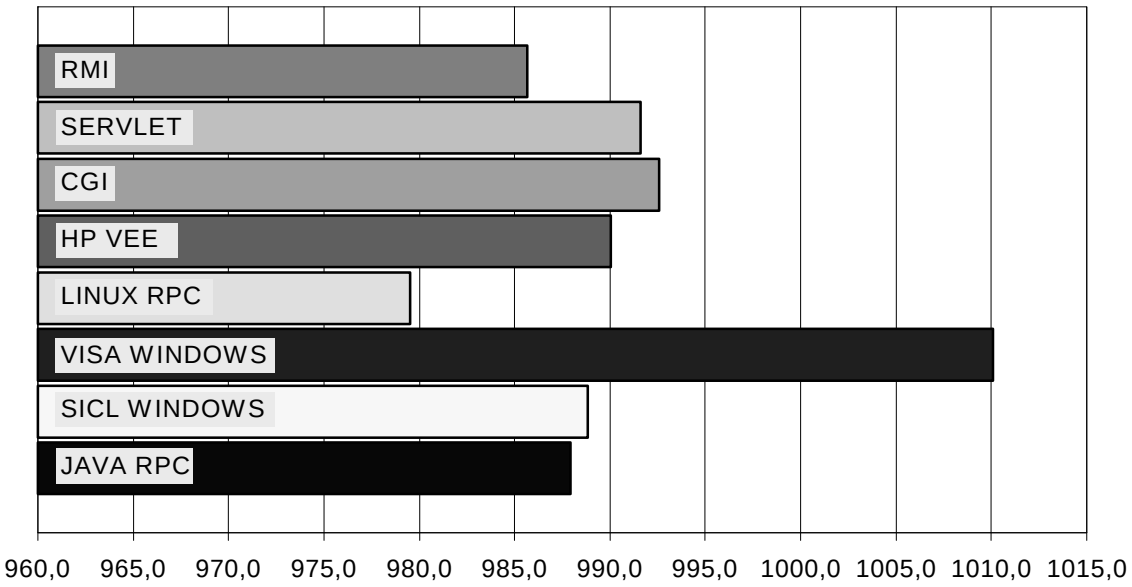

Figure 3. The second test execution time in ms.

In the third test two measuring instrument were employed (DC power supply and multimeter) to evaluate 200 points of U(I) characteristic of DUT. An interesting problem to solve when designing similar experiments using the middleware concept is what level of abstraction to chose for intermediate layer. As we have mentioned above there exists a broad range of options. Results obtained from the first and the second test let us draw a conclusion that there is a conflict between performance and flexibility.

If we decide for example that servlet or RMI server provides functions at a very low level, it will give us an opportunity to write different programs using the same servlet or RMI object. Such solution is flexible but the performance is deteriorated because every elementary instrument operation needs two network interactions: client-middleware and middleware-instrument server.

On the other hand if we choose that the intermediate layer offers only methods that perform a significant part or even the whole experiment the performance will be usually much better, because such approach requires only a few additional network interactions. However, the flexibility of such solution is rather poor. We have selected the compromise solution, in which it is possible to call the method that evaluates one point of characteristic. Appropriate software has been prepared and each version of this test was performed many times to evaluate accepted performance metric, which is shown in figure 4.

One general conclusion that can be drawn from the performed experiments is that in an Intranet environment all client/server architectures have similar and rather small influence on performance of distributed measurement experiment. Such conclusion in unlikely to be true in WAN environment. If from some reason, the middleware concept should be introduced the particular problem should be thoroughly investigated from the performance point of view. To overcome the problem caused by the conflict between flexibility and performance suitable level of abstraction while designing intermediate level API should be chosen.

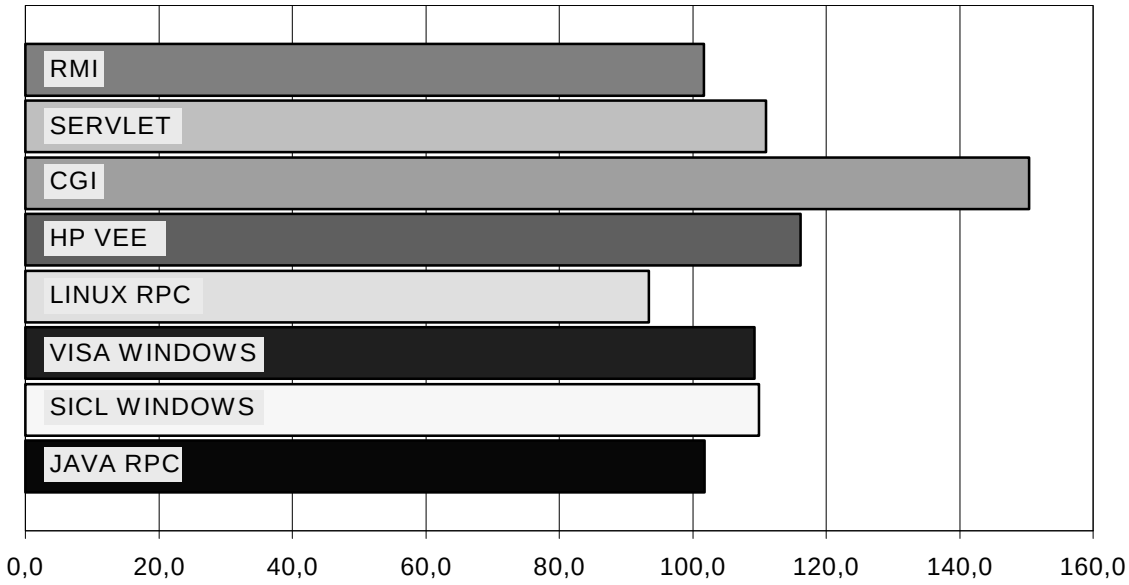

Figure 4. Total time in secons taken by experiment refered as the third test.

5 CONCLUSION

Of course, performance is not the only element considered while choosing network technology. We believe, however, that test results and suggestions we provided will help designers to make better design decision. Further research activity is being aimed at:

- Extending the range of instrumentation interfaces (VXI, instrument nets of fieldbus type),
- Extending the range of network technologies (CORBA, DCOM, JINI)
- Performance tests execution in WAN environment,
- Designing of our own implementation of VXI-11 compliant instrument server,
- Performance modelling.

REFERENCES

[1] N. Barnholt, Future test systems will use standard network protocols, Electronic Design, 10 (1994) 105.
[2] P. Arpaia, F. Cennamo, P. Daponte, M. Savastano, A distributed laboratory based on object-oriented measurement systems, Measurement Vol. 19 (3/4) (1996) 207-215.
[3] D. Grimaldi, L. Nigro, F. Pupo, Java-based distributed measurement systems, IEEE Transaction on Instrumentation and Measurement, Vol. 47 (1) 100-103.
[4] G. Fortino, D. Grimaldi, Multicast Control of Mobile Measurement Systems, IEEE Transaction on Instrumentation and Measurement, Vol.47 (5) 1149-1154.
[5] J. Pê kala, RPC/SCPI Measurement server, PAK 1997, (9) 274-276 (in Polish language).
[6] J. Pê kala, IEC-625 Bus server in LAN environment, PAK 1995, (5) 122-123 (in Polish language).
[7] J. Passquarette, Virtual instrumentation hits the Internet, National Instruments Instrumentation Newsletter, Vol. 8 (4) (1996/1997) 8.
[8] M. Bertocco, F. Ferraris, C. Offelli, M. Parvis, A client-server architecture for distributed measurement systems, IEEE Transaction on Instrumentation and Measurement, Vol. 47 (5) 1143-1148.
[9] B. Kasprzak, J. Pê kala, Performance of the IEC-625 Bus LAN server, PAK 1995, (6) 157-159 (in Polish language).

AUTHORS: Dr. J. PÊ KALA, Dr. B. KASPRZAK and Prof. J. MROCZKA, Chair of Electronic and Photonic Metrology, Wroclaw University of Technology, ul. B.Prusa 53/55, 50-317 Wroclaw, Poland
Phone Int ++48 71 3206282, Fax Int ++48 71 3214277, E-mail: pekala@kmeif.pwr.wroc.pl