A Client/Broker/Server Substrate with 50 μ s Round-Trip Overhead

Olivier Richard and Franck Cappello

LRI, Université Paris-Sud, 91405
Orsay, France
(fci)@lri.fr.

Abstract. This paper describes an environment (API and runtime) for fast remote executions in the context of NOWs using high performance networks and low cost multiprocessors. This environment is based on the usual client/broker/server architecture. It is designed and optimized for the features of local area networks. The main result of the early performance measurements is a $50\mu s$ remote execution overhead.

1 Introduction

A significant advance in networks of workstations are the availability of high performance network hardwares (ATM, Myrinet, SCI and Ethernet 1 Gbits/s) and protocols (VIA). An other advance is the progress of software technology toward the use of components. Distributed environments like NetSovle [1] and Ninf [2] already allow remote execution through the use of software components. These *Metacomputing* tools are mostly dedicated to MAN and WAN. In this note, we present a low level software mechanism (a substrate) designed for fast remote executions inside a LAN and based on high performance networks.

2 Software Architecture

The controlled remote execution substrate is based on the classical Client-Broker-Server (CBS) architecture (Figure 1). The broker provides a strong frontier between users and servers. It has the responsibility to furnish users with a unique view of servers. Users can not directly access servers. The substrate had been designed to fulfill a very low round-trip latency for remote executions. Figure 1 presents the critical path for remote executions.

Three significant points have influenced the substrate design towards a CBS dedicated to LAN. First, high performance LANs (Myrinet, SCI, Ethernet 1 Gigabits) allow to use high performance protocols (BIP, PM, Unet, Active messages, Fast Messages, VIA) because the network hardware have several key properties (almost error free, packets stay ordered, homogeneous physical support). Second, the broker and server architectures are optimized for performance. In particular objects management, heterogeneity and security are not fundamental

P. Amestoy et al. (Eds.): Euro-Par'99, LNCS 1685, pp. 790–794, 1999.

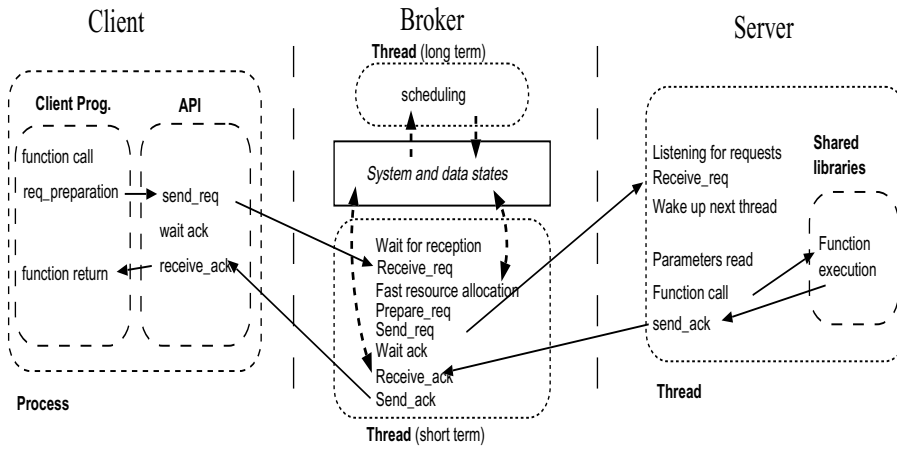

Fig. 1. Critical Path for Remote Executions.

issues for our substrate. The substrate follows a multi-thread oriented design. Servers execute only library function calls and users cannot directly launch processes on the server. Since server libraries are suppose to be fault free, there is no particular memory protection mechanism on the server. Third, the substrate uses a global naming system for the data sets on clients, broker and servers to limit the data transfers (between these entities) to the ones required by the program semantic. Function results and parameters are passed by references to avoid multiple exchange transfers between client and server when consecutive functions presents data-flow dependencies. Four client operations allow to manage data on the server side: `create`, `kill`, `put` and `get`. First operations create (destroy) a variable and return references that may be used to store and read data sets by client remote function calls. Last operations write (read) values to existing variables on the server side using references. When consecutive remote functions present dependencies, values are simply communicated between functions using references.

Clients interact with the substrate using a low level API. Users make requests to the substrate annotating their programs with directives. Directive annotations concerns the substrate operations and users functions to execute remotely. A preprocessor translates user directives in API function calls. The following program shows how annotations are used in a C program to request a remote execution of a matrix product.

```
float a[100];
//ovm create A,100  /* create a 100 entries array in the server side */
//ovm put A,a       /* send the value of the a array to A array */
//ovm req(bserv,dgemm,N,A,B,C) /* Requests remote execution of dgemm */
```

The client first requests to create an array A on the server side. A is a reference to a contiguous memory region (100 words) on the selected server. Then the value of the client local array a is transfered to the remote array A (*put*

directive). Matrix element distribution (column first or row first) in memory regions is not relevant for the substrate which conserves the words order during the transfer. Finally, the client requests the remote execution of the `dgemm` function.

3 Performance Evaluations

The platform contains a Myrinet network. 2*400 Mhz Pentium II biprocessor nodes and 2 300 Mhz Pentium II biprocessor nodes are used for the experiments. The software environment includes Linux 2.0.36, BIP 0.95c [3] and Linux Pthread library. BIP raw performances on Myrinet is a latency of 5 us and a bandwidth of 1 Gbit/s. Every tests use 2*300 Mhz PII and 1*400 Mhz PII. Client and server use 300 Mhz nodes and the broker runs on a 400 Mhz node.

We measure the bandwidth and the latency for data transfers between a client and a server through the broker. The protocols used for the `Put` and `Get` transfers are implemented on top of BIP low level communication operations. `Put` operations use two protocols according to the message size. The threshold for protocol change is 240 Bytes. Figure 2 presents the communication bandwidths for both transfer types (`Put` and `Get`) and the communication bandwidth obtained with the BIP library between two nodes.

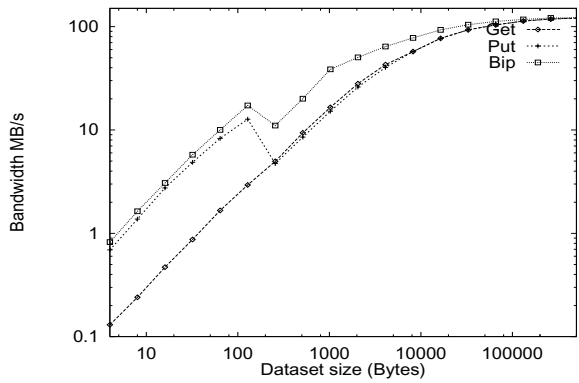

Fig. 2. Communication Bandwidth for BIP Library, `Put` and `Get` Operations

`Put` and `Get` operations reach the bandwidth of BIP for large data transfers ($> 1MB$). Client `Get` operations require $32\mu s$ to get small data sets from servers. Client `Put` operations have a $6\mu s$ inter-sending delay.

Client request latencies are shown in table 1.

Performance results do not consider the servers management algorithm executed by the broker. The client part of remote execution is about 40 times higher than a client local void function call.

Figure 3 presents the execution time seen by the client program for the matrix product routine (`dgemm`) of the BLAS library. We compare the local uniproces-

sor, the remote uniprocessor and the remote biprocessor execution times. These timings do not take into account the matrix transfers.

asynchronous remote void execution + `get`	50 μs
synchronous remote void execution	32 μs
client part of remote execution	7 μs
client local void function call	0,16 μs

Table 1. Execution Delays for Main Remote Operations of the CBS Substrate

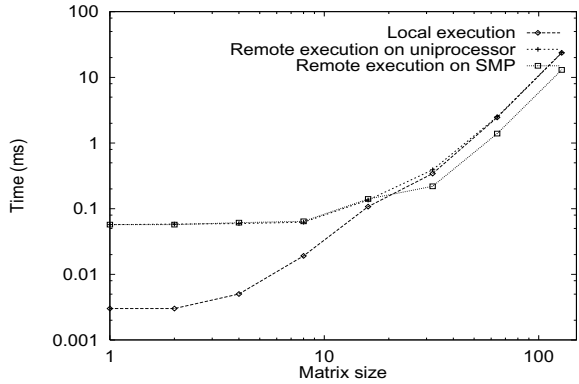

Fig. 3. Execution Times of the Matrix Products for Several Configurations.

Remote uniprocessor execution time equals the local uniprocessor execution time from 32 * 32 matrix. For higher matrix size, the parallel biprocessor implementation can be used efficiently.

Table 2 presents the speed-up of several implementations of the CG NAS NPB 2.3 serial benchmark (class W). The `conj_grad` function is executed remotely. We measure sequential as well as multi-threaded (2 threads) versions.

Executions	Remote uniprocessor	Local biprocessor	Remote biprocessor
Speed-up	0.96	1.34	1.32

Table 2. Speed-up from Local Uniprocessor of Several Remote Executions for the CG Class W Serial Benchmark

Remote executions of serial and multi-threaded versions reach respectively 96% and 98% the performance of local executions. The data transfers surrounding the remote execution of the `conj_grad` function are main limiting factors.

4 Conclusion

The CBS substrate has been designed in the perspective of very low remote execution overhead inside a LAN. Results section shows that remote execution overhead is lower than $50us$ and very short remote executions ($400us$) may be effective if they need few data transfers. By lowering the granularity of relevant remote executions, the CBS substrate enlarges the number of potential functions that are worthwhile to execute remotely and the applications domain.

References

[1] Henri Casanova et Al. NetSolve: A network-enabled server for solving computational science problems. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(3):212–223, Fall 1997.

[2] A. Takefusa et Al. Multi-client LAN/WAN performance analysis of Ninf: A high performance global computing system. *SC'97; High Performance Networking and Computing: Proceedings of the Conference*, November 15–21, 1997.

[3] L. Prylli et Al. Bip: a new protocol designed for high performance networking on myrinet. *Workshop on Personal Computers based Networks Of Workstations*, 1998.