

SANKHYA Varadhi 1.1 – Performance Report

SANKHYA Varadhi is an Object Middleware Product from Sankhya Technologies conforming to Object Management Group’s CORBA 2.2 (Minimum CORBA) Specification and can be used for Minimum CORBA based distributed systems development. With its small footprint, high performance and easy configurability, Varadhi is ideally suited for systems where size and performance are critical.

A simple and complete Varadhi client/server for the x86 target takes up just about 100KB of space (code size + data size).

Performance tests have been carried out on Varadhi to identify or measure the following:

- **Turnaround time** for the first and subsequent method invocations on the same or remote hosts (Solaris* <-> Solaris, Solaris <-> Linux, Linux** <-> Linux) on 10 Mbps LAN.
- **Memory footprint** of the following components (in bytes) – server, client, stub and skeleton for different test cases on Solaris and Linux hosts.

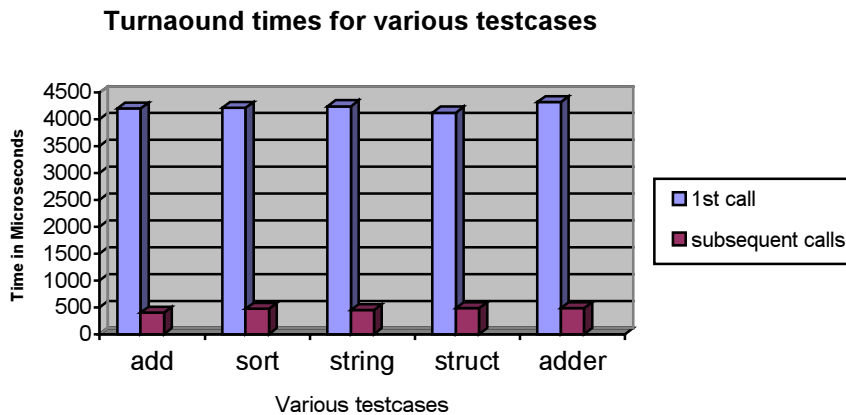
1. Turnaround time for first and subsequent method invocations on same/different hosts

The analysis of turnaround times for first and subsequent method invocations is significant since the first message/call establishes a new connection whereas the second and subsequent calls just reuse the cached connection.

Graphical illustrations of this difference in various cases are given below.

1.1 Server and Client running on same host

1.1.1 Solaris <-> Solaris

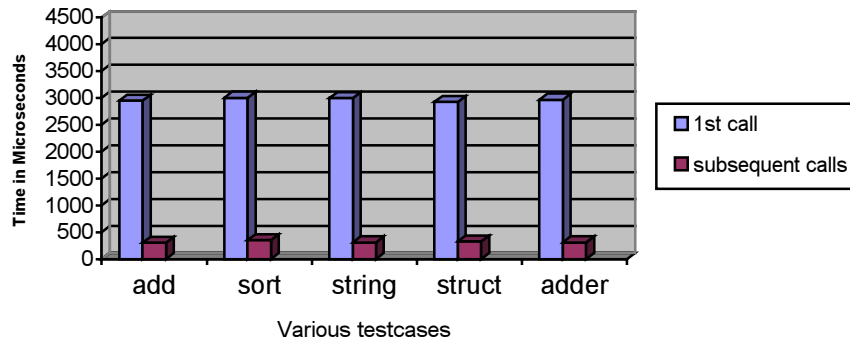


* In all cases, Solaris refers to Solaris 2.7 version of the operating system.

** In all cases, Linux refers to RedHat Linux 6.2 version of the operating system.

1.1.2 Linux <-> Linux

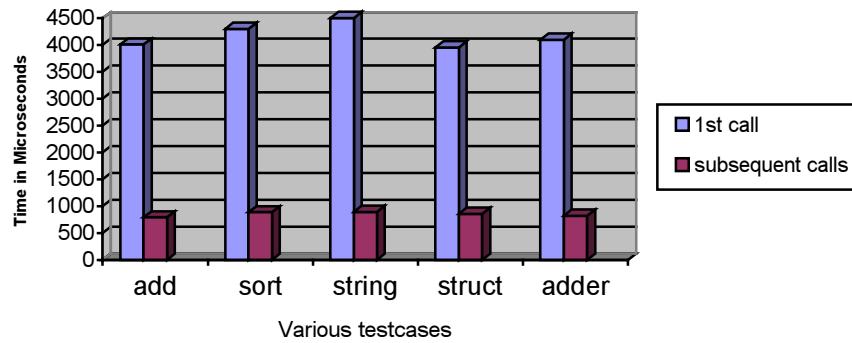
Turnaround times for various testcases



1.2 Server and Client running on different hosts

1.2.1 Linux <-> Solaris

Turnaround times for various testcases



2. Memory Footprint Tables

The following tables show the sizes (in bytes) of each of the components – Server, Client, Stub and Skeleton, broken in terms of Text, Data and BSS sections and also the total memory footprint of each component for certain important test cases.

2.1 Host OS: Solaris 2.7, Compiler: Varadhi IDL Compiler, idlc

- a) Testcase : **empty_interface**
Description: Empty Operation

Component Name	Text	Data	BSS	Total*
Server	184532	31516	2496	218544
Client	183076	30692	2448	216216
Stub	7773	1056	8	8837
Skeleton	2195	296	0	2491

- b) Testcase : **string**
Description: To perform simple string operations

Component Name	Text	Data	BSS	Total*
Server	186636	31708	2496	220840
Client	187092	31132	2440	220664
Stub	12329	1544	8	13881
Skeleton	4749	388	0	5137

- c) Testcase : **adder**
Description: Addition of two numbers

Component Name	Text	Data	BSS	Total*
Server	184148	31404	2496	218048
Client	183740	30844	2448	217032
Stub	9418	1288	8	10714
Skeleton	3104	324	0	3428

- d) Testcase : **coloc**
Description: Varadhi server and client objects reside in the same process

Component Name	Text	Data	BSS	Total*
Server	205540	35388	2520	243448
Client	192124	32196	2528	226848
Stub	18041	2284	16	20341
Skeleton	11340	1240	0	12580

* Text, Data, BSS and Total give memory footprint information for each component in terms of bytes

2.2 Host OS: RedHat Linux 6.2, Compiler: Varadhi IDL Compiler, idlc

- a) Testcase : **empty_interface**
Description: Empty Operation

Component Name	Text	Data	BSS	Total*
Server	81513	34696	2340	118549
Client	81072	33860	2308	117240
Stub	1375	1024	8	2407
Skeleton	276	280	0	556

- b) Testcase : **string**
Description: To perform simple string operations

Component Name	Text	Data	BSS	Total*
Server	82841	35032	2340	120213
Client	83808	34336	2308	120452
Stub	2458	1552	8	4018
Skeleton	1250	428	0	1678

- c) Testcase : **adder**
Description: Addition of two numbers

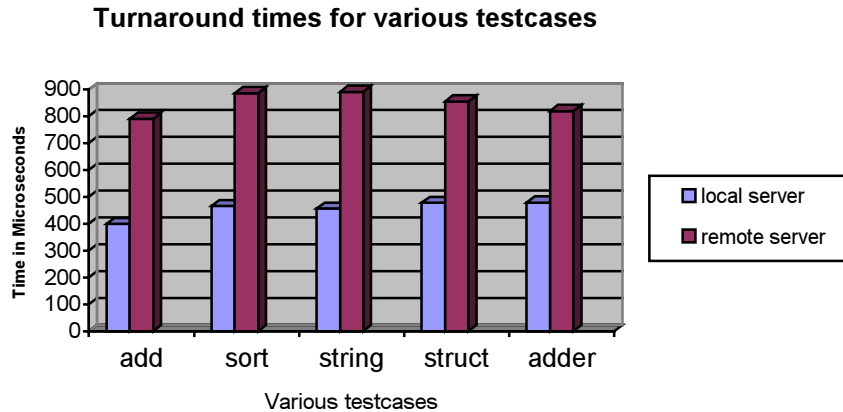
Component Name	Text	Data	BSS	Total*
Server	81369	34572	2340	118281
Client	81440	33928	2308	117676
Stub	1664	1256	8	2928
Skeleton	484	316	0	800

- d) Testcase : **coloc**
Description: Varadhi server and client objects reside in the same process

Component Name	Text	Data	BSS	Total*
Server	88201	38320	2340	128861
Client	85292	35328	2372	122992
Stub	3624	2280	16	5920
Skeleton	2348	1240	0	3588

* Text, Data, BSS and Total give memory footprint information for each component in terms of bytes

3. Turnaround time for Method Invocations – Local versus Remote host servers



Summary:

The performance results highlight the Varadhi features – **Small footprint and Portability**. Portability is achieved by abstraction of the OS layer. The protocol layer is also abstracted so that environment specific inter-operable protocols (ESIOPs) can be supported. Varadhi can be configured to use static memory allocation for various ORB objects. This enables the generation of an ORB image that has a statically determined footprint, essential for embedded systems.

Request Free SANKHYA Varadhi 30-Day Evaluation:
<http://www.sankhya.com/info/products/varadhi/download.html>

Annexure – A

IDL testcases used in measuring Varadhi performance

The interfaces or IDL files (empty_interface, string, adder, coloc) used in measuring Varadhi performance are given below:

(a) **empty_interface (file: empty.idl)**

```
// "Empty" interface
```

```
interface Empty
{
};
```

(b) **string (file: string.idl)**

```
interface String
{
void set(in string x);
string get();
string copy(in string x, out string y);
void swap(inout string x, inout string y);
void concat(in string x, in string y, in string z, inout string result);
};
```

(c) **adder (file: adder.idl)**

```
// "Adder" interface
```

```
interface Adder
{
long add(in long x, in long y);
};
~
```

(d) **coloc (file: coloc.idl)**

```
// "Adder" interface
```

```
interface Adder
{
long add(in long x, in long y);
};
```

```
// "Multiplier" interface
```

```
interface Multiplier
{
unsigned long multiply(in unsigned long x, in unsigned long y, in Adder adder);
Adder get_adder();
};
```