SOVIA: A User-level Sockets Layer Over Virtual Interface Architecture

Jin-Soo Kim
jinsoo@computer.org

Computer System Research Department
Electronics and Telecommunications Research Institute (ETRI)
Korea
Outline

- Introduction
- Performance Issues
- Compatibility Issues
- Results on Real Applications
- Concluding Remarks
Introduction
Virtual Interface Architecture

Overheads in traditional communication architecture
- Protocol overhead in TCP/IP
- Data copying
- Context switching

Virtual Interface Architecture
- Led by Compaq, Intel, and Microsoft
- A standard software interface for user-level access to a network hardware
VI Architecture Overview

<table>
<thead>
<tr>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS Communication Interface</td>
</tr>
<tr>
<td>VI User Agent</td>
</tr>
</tbody>
</table>

**VI Consumer**
- User mode: Open/Connect/Register Memory

**VI Provider**
- Kernel mode: Send/Receive/RDMA

**VI Kernel Agent**

**VI Network Adapter**
## VI Provider Library (VIPL)

<table>
<thead>
<tr>
<th>Hardware Connection</th>
<th>Memory Protection and Registration</th>
<th>Data Transfer</th>
</tr>
</thead>
<tbody>
<tr>
<td>VipOpenNic()</td>
<td>VipCreatePtag()</td>
<td>VipPostSend()</td>
</tr>
<tr>
<td>VipCloseNic()</td>
<td>VipDestroyPtag()</td>
<td>VipSendDone()</td>
</tr>
<tr>
<td></td>
<td>VipRegisterMem()</td>
<td>VipSendWait()</td>
</tr>
<tr>
<td></td>
<td>VipDeregisterMem()</td>
<td>VipSendNotify()</td>
</tr>
<tr>
<td>Endpoint Creation and Destruction</td>
<td></td>
<td>VipDisconnect()</td>
</tr>
<tr>
<td>VipCreateVi()</td>
<td>Querying</td>
<td></td>
</tr>
<tr>
<td>VipDestoryVi()</td>
<td>VipQueryNic()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VipQueryVi()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VipSetViAttributes()</td>
<td></td>
</tr>
<tr>
<td>Connection Management</td>
<td>VipSetMemAttributes()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VipQueryMem()</td>
<td>Completion Queues</td>
</tr>
<tr>
<td></td>
<td>VipQuerySystemManagementInfo()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Error</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VipErrorCallback()</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>VipCreateCQ()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VipDestroyCQ()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VipResizeCQ()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VipCQDone()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VipCQWait()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VipCQNotify()</td>
</tr>
</tbody>
</table>
Motivation

Problem in VIPL
- “VIA is considered by many systems designers to be too low a level for application programming.”
  -- TFCC Cluster Computing White Paper

Requirements for a new layer
- High Performance
- Portability

SOVIA (Sockets Over VIA)
Berkeley Sockets API

Traditional architecture

- **Pros**
  - Full compatibility

- **Cons**
  - TCP/IP overhead
  - Context switching
  - Data copy

```
application

Library

kernel

Sockets
TCP / UDP
IP
NIC Driver

NIC
```
Sockets over VIA (1)

Using an IP-to-VI Layer

- Giganet’s LANE (LAN Emulation) Driver

Pros
- Full compatibility

Cons
- TCP/IP overhead
- Context switching
- Data copy
- Emulating Connectionless IP over connection-oriented VIA

VI-NIC

VI Kernel Agent

IP-to-VI Layer

IP

TCP / UDP

Sockets

Library

application
Sockets over VIA (2)

Using a Sockets-to-VI Layer

• VIsocket on Solaris

Pros

- No TCP/IP overhead
- Good compatibility

Cons

- Context switching
- Data copy

Diagram:

- Application
  - Library
  - Sockets
  - Sockets-to-VI Layer
  - VI Kernel Agent
  - VI-NIC
Sockets over VIA (3)

Using a User-level Sockets Layer

- MS WinSock Direct Path, SOVIA

**Pros**
- TCP/IP overhead
- Context switching
- Data copy

**Cons**
- Less compatibility
Goals of SOVIA

- Provide a simple, yet versatile communication service
- Accelerate the existing Sockets-based applications with a reasonable effort
  - Parallel and cluster file systems
  - User-level software DSMs, ...
- Target for another upper layers
  - RPC (Remote Procedure Calls)
  - MPI (Message Passing Interface), ...

The Third IEEE International Conference on Cluster Computing, Newport Beach, CA, USA, October 8 - 11, 2001. -- Jin-Soo Kim (jinsoo@computer.org)
Performance Issues

- Minimizing latency
- Maximizing bandwidth
- Evaluation
Synchronization Protocol (1)

VIA’s pre-posting constraint

- The receiver should pre-post a descriptor before the sender initiates a data transfer.
- A high-level synchronization protocol needs to be implemented between the sender and receiver.
Synchronization Protocol (2)

Satisfying pre-posting constraint

- Guarantee that at least one descriptor is available on the RQ for each send.

Three-way Handshaking

Two-way Handshaking
Message Handling Strategies

Incoming message handling

• The completed descriptors should be extracted from a queue manually.
• Incoming messages are delivered asynchronously.
• When and by whom?
Multi-threaded Handling

- High thread synchronization cost (\( \approx 20 \text{ us} \)) offsets the low-latency benefit of VIA.
- Does not achieve a true user-level data transfer.
Single-threaded Handling

• The application thread handles incoming messages when it calls communication-involved functions.

• Can overlap the communication with the computation if multiple descriptors are pre-posted.
Conditional Registration

Unit cost for memory registration/deregistration and memcpy()

- Registration/Deregistration Cost (M-VIA)
- Registration/Deregistration Cost (cLAN)
- memcpy(), source cached
- memcpy(), source uncached

Message size (Bytes)

Time (usec)

0 5 10 15 20

1 8 16 32 64 128 256 512 1K 2K 4K 8K 16K 32K 64K
Performance Issues

- Minimizing latency
- Maximizing bandwidth
- Evaluation
Flow Control

A sliding window protocol

- Receiver pre-posts $w$ descriptors
- `send()` decreases $w$ by one
- Sender is blocked if $w = 0$
- Window size $w$ is increased by ACK
Delayed ACKs and Piggybacking

- Delay up to $t$ ACKs
- Piggyback ACKs to DATA

Before

```
send();
recv();
send();
send();
send();
send();
send();
send();
send();
send();

. . . . .
```

DATA
ACK

After ($t = 8$)

```
send();
recv();
send();
send();
send();
send();
send();
send();
send();
send();
send();

. . . . .
```

```
recv();
send();
recv();
recv();
recv();
recv();
recv();
recv();
recv();
recv();
recv();
recv();
recv();
recv();
recv();
recv();
recv();
recv();
recv();
```

DATA (+0)

DATA (+1)

DATA (+0)

DATA (+0)

DATA (+0)

ACK (+8)
Combining Small Messages

- TCP uses Nagle algorithm
- Append outgoing data into a buffer (< 2KB), and start a timer
- The buffer is flushed either
  - when the timer expires
  - when there is no enough room
  - when the size is > 2KB
  - when the application calls recv() or close()
- Dynamically turned off for latency-sensitive applications.
Performance Issues

- Minimizing latency
- Maximizing bandwidth
- Evaluation
Evaluation Platform

Linux servers

- Intel L440GX+ motherboard
- Intel Pentium III-500MHz
- 256MB main memory
- Linux kernel 2.2.16

VIA implementations

- cLAN v1.1.1 on Giganet cLAN1000
Latency (cLAN)

Latency (Giganet cLAN1000)

TCP: 54.9 s
SOVIA: 10.5 s
VIA: 8.3 s
Bandwidth (cLAN)

TCP: 452 Mbps
SOVIA: 814 Mbps
VIA: 815 Mbps
Design Choices

Minimizing latency
• Two-way handshaking
• Single-threaded implementation
• Conditional memory registration

Maximizing bandwidth
• A sliding window protocol
• Delayed acknowledgments and piggybacking
• Combining small messages
Compatibility Issues

- Connection management
- Enhancing portability
**Connection Management**

Problems in the single-threaded implementation

---

Connection Management

Client

- close()
- FIN
- FINACK
- ?
- ?

Server

- connect()
- FIN
- FINACK
- close()
- ?

No chance to handle incoming messages after last close()

---

Client

- connect() returns successfully, once the server issues listen()

Server

- listen()
- other codes
- accept()

VipConnectWait() can’t be called in listen(), as it would block the thread.
Establishing a Connection

Client

- close thread
- main thread
- socket()
- connect()
- send()
- signal
- VI connection established
- WAKEUP

Server

- connection thread
- main thread
- close thread
- listen()
- accept()
- recv()
- send()
- signal
- WAKEUP

Client Server
• The close thread is not activated if there is an open connection.
• Hence, the presence of the close/connection thread does not affect the application’s performance.
Compatibility Issues

- Connection management
- Enhancing portability
Issues in Porting Applications

- **Sockets-specific interface**
  - `socket()`, `connect()`, `accept()`, ...

- **File system interface**
  - `read()`, `write()`, `close()`, ...

- **Standard I/O library**

```c
int s;
FILE *fp;
... s = socket (AF_INET, SOCK_STREAM, 0);
connect (s, (struct sockaddr *) &server, sizeof(server));
fp = fdopen (s, “w”);
fprintf (fp, “Hello, world...
”);
```
Enhancing Portability

- sov_socket(), sov_read(), sov_write(), sov_close(), ...
- socket(), read(), write(), close(), ...
- fprintf(), fgetc(), fputc(), ...

**compatibility layer**

**user**
- SOVIA
- EVIPL (extended VIPL)
  - M-VIA’s VIPL
  - cLAN’s VIPL
  - M-VIA’s Kernel Agent
  - cLAN’s Kernel Agent
  - Intel eepro100
  - cLAN1000

**kernel**
- libc
- system call interface
- kernel
Other Compatibility Issues

**Fork()**
- OK if the child process is not involved in any communication – Had to fix the VIPL due to the copy-on-write problem.
- Sockets can not be shared with child processes.

**Exec()**
- Exec() will destroy any user-level data.

**Normal termination**
- Register a cleanup function using atfinalize().

**Abnormal termination**
- Catch the abnormal termination in signal handlers and call a cleanup function.
Results on Real Applications
## FTP Performance

### Linux NetKit 0.16
- `linux-ftpdp-0.16 & netkit-ftp-0.16`

<table>
<thead>
<tr>
<th></th>
<th>File 1</th>
<th>File 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File size (bytes)</strong></td>
<td>19,090,223</td>
<td>145,864,380</td>
</tr>
<tr>
<td><strong>TCP/IP on Fast Ethernet</strong></td>
<td>90Mbps (1.63 sec)</td>
<td>90Mbps (12.7 sec)</td>
</tr>
<tr>
<td><strong>TCP/IP on cLAN</strong></td>
<td>262Mbps (0.59 sec)</td>
<td>254Mbps (4.61 sec)</td>
</tr>
<tr>
<td><strong>SOVIA on cLAN</strong></td>
<td>573Mbps (0.27 sec)</td>
<td>532Mbps (2.20 sec)</td>
</tr>
<tr>
<td><strong>Local copy (in ramdisks)</strong></td>
<td>611Mbps (0.25 sec)</td>
<td>538Mbps (2.17 sec)</td>
</tr>
</tbody>
</table>

The Third IEEE International Conference on Cluster Computing, Newport Beach, CA, USA, October 8 - 11, 2001. -- Jin-Soo Kim (jinsoo@computer.org)
RPC Performance

sunrpc in glib-2.1.3

call r()

argument

return

TCP (FE): 200 \( ? \) s
TCP (cLAN): 149 \( ? \) s
SOVIA (cLAN): 35 \( ? \) s
Concluding Remarks

The SOVIA layer provides
- High performance from VIA
- Portability based on Sockets API

On-going & Future Work
- Accelerate other applications
  - Parallel file systems (e.g. PVFS)
  - Network Block Device (e.g. GNBD)
  - User-level software DSMs
  - Message-passing library, etc.
- Compare with kernel-level SOVIA