The Impact on Latency and Bandwidth for a Distributed Shared Memory System Using a Gigabit Network Supporting the Virtual Interface Architecture

John Markus Bjørndalen[1], Otto J. Anshus[1], Brian Vinter[1,2], Tore Larsen[1]

Department of Computer Science [1]
University of Tromsø

Department of Mathematics and Computer Science [2]
University of Southern Denmark

Abstract

Previous studies have shown significant performance advantages in using Virtual Interface Architecture (VIA) instead of TCP/IP for handling network communication in the structured distributed shared memory system, PastSet. With the availability of network hardware that supports VIA, we wish to examine whether, and to what extend, an available hardware supported VIA implementation outperforms the software-only implementation for PastSet DSM. To do this, PastSet has been ported to two VIA implementations: M-VIA, which is a software implementation that we use on a 100 Mbit Fast Ethernet, and Giganet cLAN, which uses dedicated VIA hardware. The two implementations are tested, and performance results are compared with the reference TCP/IP implementation on the 100 Mbit Fast Ethernet.

For the experiment setups used, M-VIA latencies are between 1.1 and 2.6 times faster than corresponding latencies using TCP/IP.

For large packets, Giganet cLAN latencies are about 2.7 times faster than corresponding M-VIA latencies. However, for small packets, cLAN latencies are only about 1.04 times faster than corresponding M-VIA latencies, indicating that the current software design and implementation does not fully benefit from the improved performance of Giganet cLAN over Fast Ethernet. Further experiments demonstrate that significantly improved small-packet latencies on cLAN are possible, and may be achieved through a software redesign carefully considering the use of polling versus interrupts.

1 Introduction

The latency and bandwidth performance of a Distributed Shared Memory (DSM) system depends on the performance and interaction of the DSM and the underlying network subsystems. The key challenge [8] is to preserve the performance characteristics of the physical network (bandwidth, latency, QoS) while making effective use of host resources. Network bandwidths and latencies are constantly improving. Unfortunately, applications have not been able to take full advantage of these performance improvements due to the interactions of layers of user and kernel level software. A detailed breakdown of hardware and software costs of remote memory operations is discussed in [3]. The Virtual Interface Architecture (VIA) was developed to significantly reduce the software overhead between a high performance CPU/memory subsystem and a high performance network.

In this paper, we study the latency and bandwidth performance of PastSet DSM using either M-VIA[10], a software VIA implementation for Linux; Giganet cLAN[9], a VIA implementation with hardware VIA support; or TCP/IP. The paper briefly describes the functionality of PastSet, the organization of the implementation, the interactions between PastSet components, and the VIA implementations. Experiment configurations with micro-benchmarks and metrics are described before presenting and analyzing benchmark results.

2 Implementing the PastSet Server and Application Library

PastSet is a structured distributed shared memory system. PastSet memory objects are tuples. Operations exist to create tuples, copy tuples to DSM, and read tuples in DSM. The DSM is structured in that tuples are organized in disjoint elements, and that an ordering of tuples is maintained within each element. Operations exist to create elements and define ordering criteria for each element. A synchronization mechanism is included in the memory model, and a synchronization criterion may be set for each element. Operations are provided to set synchronization criteria.

All PastSet operations are blocking. The PastSet memory model complies with sequential consistency.

For this paper, the PastSet operation `move` is used in determining PastSet latencies. `Move` takes a tuple as parameter and copies the content of the tuple into a specified element in DSM, maintaining tuple order and synchronization criterion. The `move` operation blocks in the sense that it returns only after confirmation has been received from the PastSet server that the operation is completed.

The design, applicability, and performance of PastSet DSM is demonstrated in [1] and [12].

The synchronous nature of PastSet operations implies that each operation request requires a reply message with the result of the operation before the client may continue execution; consequently, two messages are required for each remote operation.

The version of the PastSet server used for the experiments reported on in this paper creates a new thread for each new client connection. Each thread is exclusively responsible for servicing it's associated connection. The threads loop, reading requests, performing operations on behalf of the client and returning results to the caller.

This is a simple approach, with low overhead for a small number of connections. However, the single-thread-per-connection approach is not well suited for multi-threaded clients where several client threads may need to share the same connection. We have developed alternatives to using a single thread per connection, but we will not report on these in this paper.

2.1 TCP/IP implementation

When a PastSet operation requests non-local data, the operation and its parameters are sent via a TCP/IP connection to the remote PastSet server, and the caller is blocked awaiting the reply from the PastSet server.

All connections disable the Nagle algorithm to ensure that even small data packets are sent immediately.

2.2 M-VIA implementation

The PastSet server and the application library were implemented using the M-VIA 1.0 [10] implementation of the VIA API. By using the message passing model of VIA, we got a simple port from the TCP/IP implementation. The alternative, using remote DMA, is complicated by the way PastSet operations can manipulate and address PastSet distributed shared memory.

The PastSet server and the application library use blocking calls to M-VIA in order to reduce the processor usage. M-VIA first check to see if the data already has arrived. If not, a block is done.

The 100Mbit network interface cards (NICs) we used do not support the "doorbell" mechanism of the VIA. Instead, this is done in software in M-VIA, making traps to the Linux kernel necessary.

The tuples that are used by the micro benchmarks we use are allocated in parts of the memory that are registered with the M-VIA NICs in order to reduce copying on send and receive. However, M-VIA first copies the data from the NIC to kernel level memory, and then from kernel memory to the user level application memory.

2.3 cLAN implementation

With hardware support, VIA is intended to enable applications to send and receive packets over a Virtual Interface without trapping to the operating system kernel. The kernel is basically only involved in setting up and tearing down connections, and in other book-keeping tasks. In particular, the incoming data is directly written to the user level application memory.

The PastSet server and the application library using hardware supported VIA are otherwise basically identical to the one using M-VIA. In particular, blocking calls are used doing a little spinning to check if data already have arrived before doing the actual blocking.

3 Methodology and Experiment Design

This section describes the hardware and software details of the experiments, how the timing measurements were done, the micro-benchmarks, and the metrics used.

3.1 Hardware and Software

All experiments reported on in this paper were done using two HP LX-Pro Netservers, each having four 166MHz Pentium Pro CPUs. Each computer had 128MB main-memory, and dual peer 33MHz, 32 bit PCI buses. The level 2 cache size is 1MB per processor.

For the experiments, the computers were interconnected using either Giganet cLAN 1.25Gb/s [9] or Trendnet TE100-PCIA

(DEC Tulip 21143 chip set) 100 Mb/s network interface cards (NIC) connected to a hub. Both NICs were on PCI bus no. 0 on each server. In addition, a 100VG 100Mb/s NIC, also on PCI bus no. 0, was used to connect to the local area network of the Department of Computer Science. This network was used to manage the experiments and the servers.

Linux v. 2.2.14 with PastSet functionality added to the kernel was installed on each node participating in the experiments. We used M-VIA version 1.0 with a minor patch to the connection management.

We compiled M-VIA, cLAN, the PastSet Server, the PastSet Kernel, the PastSet Application Library, the Linux operating system, and the benchmarks using egcs 1.1.2.

Default compiler flags were used for M-VIA, cLAN, The PastSet Server, the PastSet Kernel, and the Linux operating system. We used the optimization flags "`-O6 -m486 -mjumps=2 -malignloops=2 -malignfunctions=2`." for the benchmarks and the PastSet library.

Because we experienced problems with M-VIA when using four processors, we redesigned the experiments to use only one processor per server, and we recompiled the Linux kernel to run as a single processor system.

3.2 Micro benchmarks and Metrics

To measure the latency of the PastSet operations, we used several micro benchmarks. In this paper we will only report on the `move` latency, that is, the time to invoke, complete and return from a `move` operation. The `move` operation blocks when waiting for an acknowledgement message from PastSet.

The client process calls `move` operations.

```
for (i = 0; i < 1000; i++) {
   save_timestamp;
   mv();
}
save_timestamp;
```

Figure 1: The `move` latency (Mvlat) benchmark

The client process running the benchmark and the PastSet server are on two different computers.

To determine the effect of using blocking vs. spinning when waiting for data, we used the `vnettest` micro benchmark taken from the M-VIA 1.0 distribution. This micro benchmark is a low level roundtrip ping-pong of data. We modified `vnettest` so we could choose to use either blocking or spinning when waiting for data.

To determine the effect of the underlying network technology, each micro benchmark used TCP/IP, software supported VIA (M-VIA), and hardware supported VIA (Giganet cLan).

Data size for the messages was varied from one to 31KB. The elapsed time for 1000 transmissions is measured for each packet size and then divided by 2000 to get the average latency of a message from address space to address space. We repeated each run of 1000 transmissions five times.

When doing the performance measurements each node supported no other workload except for the operating system and its various artifacts.

All necessary initializations were done before starting time- or cycle measurements.

3.3 Time Measurements

The Intel Pentium Pro RDTSC (read time-stamp counter) instruction and the Linux *gettimeofday()* system call were used to determine PastSet operation latencies.

Using RDTSC, as in [6], the cycle count was recorded for every `move` operation. Elapsed time in microseconds was calculated by dividing the registered cycle count by the specified processor frequency of 166 MHz. We did not verify the actual frequency of each individual computer, leaving open the possibility that the computed time may deviate slightly, but consistently, from the performance measured in cycles spent. Care was taken to avoid potential problems with register overwrites and counter overflow.

The *gettimeofday()* system call was used for aggregate measurements over many operation calls. Checks were made to ensure that RDTSC and *gettimeofday()* measurements were consistent.

Cache effects are not eliminated, but measurements are averaged over five runs of one-thousand iterations each, and no other workload is present.

4 Micro-benchmark Results

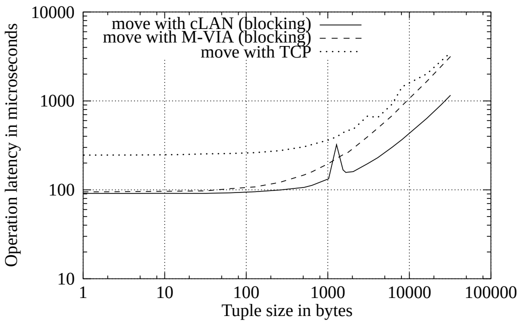
4.1 Move Latency Results

Figure 2 shows `move` latencies and bandwidth for intra-node communication using TCP/IP, M-VIA and cLan. Tuple sizes are varied from one byte to 31KB. The bandwith is computed from the latency since PastSet requires one operation to complete before the next can be initiated.

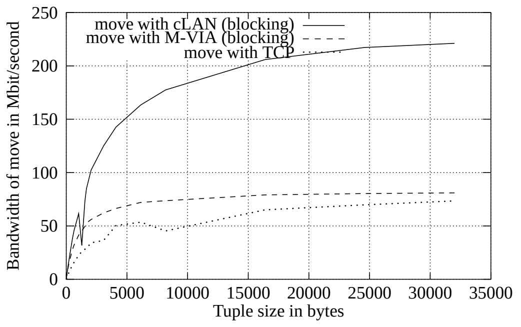
For small tuple sizes `move` latency using M-VIA is about 2.6 times faster than TCP/IP. M-VIA latency is about 1.1 times faster than TCP/IP for 31 KB tuples. For one byte tuples, the difference between using M-VIA and TCP/IP is 152 microseconds, while at 31KB the difference is 330 microseconds.

cLan performs slightly better than M-VIA on small tuple sizes (about 1.04 times faster than M-VIA). At larger tuple sizes, the performance of cLan is 2.7 times faster than M-VIA and 3 times faster than TCP/IP.

The observed bandwith using M-VIA is about 70 percent of the potential 100 Mbit/s that the hardware can support, while using cLan we achieve about 18 percent of the potential 1.25 Gbit/s.



(a) latency



(b) bandwith

Figure 2: The Mvlat benchmark results: operation latency and bandwidth of the PastSet `move` operation

Most of the performance improvement of M-VIA over TCP/IP comes from the implementation drawing advantage of local network properties. M-VIA skips ethernet checksums (done in hardware) and handles much of the protocol in the interrupt handler while TCP/IP has to send the data through several layers and compute checksums for ethernet frames, IP headers and TCP packets. M-VIA also uses faster traps to the kernel than TCP/IP.

The performance advantage with cLan is first visible at larger tuple sizes where the higher bandwith (1.25 Gbit vs 100 Mbit) becomes more important. At smaller packet sizes, the benefit from hardware support is masked by the overhead in the management of the blocking calls. As such, the current PastSet implementation does not show much of a performance benefit due to the hardware implementation of VIA.

4.2 Latencies of Polling and Blocking Message Passing

Figure 3 shows message passing latencies of cLan and M-VIA measured with vnettest using spinning (polling) and blocking VIA calls.

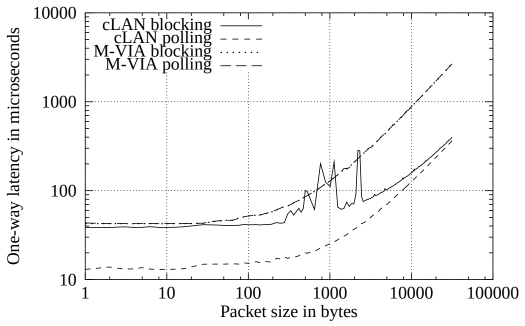


Figure 3: One-way latency over cLAN and M-VIA measured with vnettest

For basic ping-pong communication there is little difference in latency between spinning and blocking communication when using M-VIA, resulting in the graphs overlapping in the figure. This effect comes from the fact that the software M-VIA implementation has to handle interrupts and protocol implementation both for polling and blocking operations.

The extra overhead from the kernel traps (up to 2 ioctl calls per send or receive operation) are overlapped with the pysical transmission of data. This might hurt the performance of M-VIA during high load from multiple clients.

Using polling on cLAN gives a clear advantage, reducing the latency with 20-30 microseconds over all tested packet sizes compared to the blocking version.

4.3 Implications for PastSet implementation

For small tuple sizes the latency of PastSet move operations is about 100 microseconds. Using spinning on cLan achieves a one-way latency improvement of 20-30 microseconds as compared to blocking. This translates into a potential move latency improvement of 40-60 microseconds.

Achieving this requires modifications to the PastSet server and application library. The use of spinning must be carefully applied due to its CPU usage[7].

5 Related work

How to reduce waiting costs in user-level communication has been reported on in several papers, including [7]. This paper describes a mechanism for reducing the cost of waiting for messages in architectures that allow user-level communication libraries.

They document how blocking and spinning can affect the performance, and correlates well with our results.

VIA is currently being introduced for various message passing sub-systems. Systems that are based on the p4[5] communication library are candidates for porting to the VIA API, e.g. the M-VIA team have ported MPICH to use M-VIA instead. Distributed shared memory systems which uses VIA includes the page based HLRC DSM system [11].

Work on building DSM systems on top of user level communication libraries includes the Virtual Memory Mapped Communication system, VMMC [4].

The Orca object based DSM system has an associated communication library, PANDA, which also provides a high performance communication system that runs on Myrinet. PANDA is specifically designed for Orca which is highly dependent on multicast[2].

6 Conclusions

Based on the performance results we can conclude that:

- Hardware supported VIA gives a non-significant improvement in PastSet operation latency over software M-VIA for small tuple sizes. This is because the PastSet operation is blocking, and the cost of blocking is much higher than the advantage of the small protocol overhead in hardware supported VIA

- Hardware supported VIA gives a significant improvement in PastSet operation latency over software M-VIA for large tuple sizes. This is because the hardware supported VIA is a gigabit network versus the megabit network used by the software M-VIA

- Hardware supported VIA benefits significantly from using spinning instead of blocking when waiting for data. This is because the cost of blocking is avoided

- Software supported M-VIA does not benefit significantly from using spinning instead of blocking. This is because the protocol implies several traps to the kernel per data transfer, and this is much more expensive than the benefit coming from spinning

- By using spinning and hardware supported VIA, the PastSet move latency may be cut in half. However, carefully combining spinning and blocking seems to be needed to benefit from gigabit networks with hardware support for VIA while at the same time not using too much processor cycles

7 Acknowledgements

Giganet Inc. kindly provided us with a cLan environment. Ken Arne Jensen and Jon Ivar Kristiansen provided invaluable support under the preparation of this paper.

References

[1] ANSHUS, O. J., AND LARSEN, T. Macroscope: The abstractions of a distributed operating system. *Norsk Informatikk Konferanse* (October 1992).

[2] BHOEDJANG, R., RUHL, T., AND BAL, H. E. Efficient multicast on

myrinet using link-level flow control. In *International Conference on Parallel Processing* (Minneapolis, MN, August 1998), pp. 381–390.

[3] BILAS, A., IFTODE, L., AND SINGH, J. P. Evaluation of hardware support for automatic update in shared virtual memory clusters. In *12th ACM International Conference on Supercomputing* (July 1998).

[4] BLUMRICH, M., LI, K., ALPERT, R., DUBNICKI, C., FELTEN, E., AND SANDBERG, J. A virtual memory mapped network interface for the shrimp multicomputer. In *Proceedings of the 21st Annual Symposium on Computer Architecture* (April 1994), pp. 142–153.

[5] BUTLER, R., AND LUSK, E. User's guide to the p4 parallel programming system. Tech. Rep. ANL-92/17, Argonne National Laboratory, October 1992.

[6] CHEN, J. B., ENDO, Y., CHAN, K., MAZIERES, D., DIAS, A., SELTZER, M., AND SMITH, M. The measured performance of personal computer operating systems. *ACM Transactions on Computer Systems* (February 1996).

[7] DAMIANAKIS, S. N., CHEN, Y., AND FELTEN, E. Reducing waiting costs in user-level communication. In *11th International Parallel Processing Symposium (IPPS '97)* (April 1997).

[8] DRUSCHEL, P., AND PETERSON, L. L. Operating systems and network interfaces. In *Foster, Ian and Kesselman, Carl (Eds.), The Grid: Blueprint for a New Computing Infrastructure* (1999), Morgan Kaufmann.

[9] http://www.giganet.com/.

[10] http://www.nersc.gov/research/ftg/via/.

[11] RANGARAJAN, M., AND IFTODE, L. Software distributed shared memory over virtual interface architecture: Implementation and performance. Tech. Rep. DCS-TR-413, Rutgers University, Department of Computer Science, April 2000. To appear in Proceedings of The Third Extreme Linux Workshop, October 10-12, Atlanta.

[12] VINTER, B. *PastSet a Structured Distributed Shared Memory System*. PhD thesis, Tromsø University, 1999.