An Extensible Message-Oriented Offload Model for High-Performance Applications*

Patricia Gilfeather and Arthur B. Maccabe
Scalable Systems Lab
Department of Computer Science
University of New Mexico
pfeather@cs.unm.edu, maccabe@cs.unm.edu

Abstract

In this paper, we present and validate a new model designed to capture the benefits of protocol offload in the context of high performance computing systems. Other models capture the benefits of offload or the performance of parallel applications. However, the extensible message-oriented offload model (EMO) is the first model to emphasize the performance of the network protocol itself and models it in a message-oriented rather than flow-oriented manner. EMO allows us to consider benefits associated with the reduction in message latency along with benefits associated with reduction in overhead and improvements to throughput.

In order to validate EMO, we use the tool to model a very common offload technique, interrupt coalescing. We discuss the assumptions of our model and show the modeled offload and latency performance of interrupt coalescing and no interrupt coalescing. We then present preliminary results to validate that our model is accurate.

1 Introduction

Network speeds are increasing. Both Ethernet and Infiniband are currently promising 40 Gb/s performance, and Gigabit performance is now commonplace. Offloading all or portions of communication protocol processing to an intelligent NIC (Network Interface Card) is frequently used to ensure that benefits of these technologies are available to applications. However, determining what portions of a protocol to offload is still more of an art than a science. Furthermore, there are few tools to help protocol designers choose appropriate functionality to offload.

Currently, there are no models that address the specific concerns of high-performance computing. We create a model that explores offloading of commodity protocols for individual messages which allows us to consider offloading performance for message-oriented applications and libraries like MPI.

In this paper, we first briefly outline our new model, the extensible message-oriented offload model (EMO), that allows us to evaluate and compare the performance of network protocols in a message-oriented offloaded environment. [2] provides an in-depth introduction to EMO, along with a comparison of this model to other popular performance models, LAWS [4] and LogP [1] and a case study for using the model to develop new offloaded network protocols. Second, we use EMO to model the latency and overhead of messages using no interrupt coalescing, using default interrupt coalescing and using maximum default interrupt coalescing. Third, we discuss the methods used to measure the actual latency and overhead of various aspects of a protocol. Finally, we present our preliminary results in validating the model by comparing modeled latencies for interrupt coalescing with actual results.

2 Extensible Message-Oriented Offload Model

We created a performance model that is not specific to any one protocol, but our choices were informed by our understanding of MPI over TCP over IP.

Figure 1 shows the communication architecture used for EMO. The latency and overhead that is necessary to communicate between components must include the movement of data when appropriate.

The variables for this model are as follows:

- C_N = # cycles of protocol processing on NIC
- R_N = Rate of CPU on NIC
- L_{NH} = Time to move data and control from NIC to Host OS

*Los Alamos Computer Science Institute SC R71700H-29200001

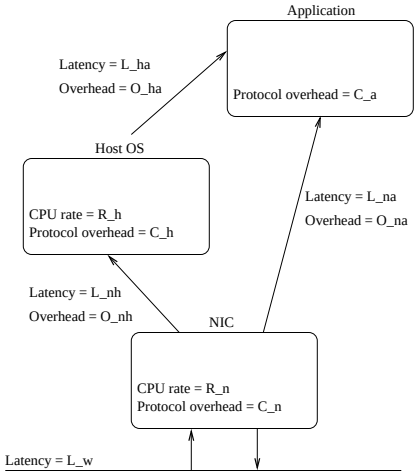

Figure 1. The Extensible Message-oriented Offload Model

- C_H = # cycles of protocol processing on Host
- R_H = Rate of CPU on Host
- L_{HA} = Time to move data and control from Host to App
- L_{NA} = Time to move data and control from NIC to App
- C_A = # cycles of protocol processing at Application
- O_{NH} = # host cycles to move data and control from NIC to Host OS
- O_{HA} = # host cycles to move data and control from Host OS to App
- O_{NA} = # host cycles necessary to communicate and move data from NIC to Application

2.0.1 Extensibility

The model allows for extensibility with respect to protocol layers. We hope this model can be useful for researchers working on offloading parts of the MPI library (like MPI_MATCH) or parts of the matching mechanisms for any language or API. We constructed the model so that it can grow through levels of protocols. For example, Our model can by extended, or telescoped, to include offloading portions of MPI. We simply add C_m, L_{am} and O_{am} to the equations for overhead and latency.

2.0.2 Overhead

EMO allows us to explore the fundamental cost of any protocol implementation, its overhead. Overhead occurs at the per-message and per-byte level. Our model allows us to estimate and graphically represent our understanding about overhead for various levels of protocol offload.

Overhead is modeled as

$$\text{Overhead} = O_{NH} + C_H + O_{HA} + C_A + O_{NA}$$

. However, all methods will only use some of the communication patterns to process the protocol. Traditional overhead, for example, will not use the communication path between the NIC and the application and does no processing at the application.

$$\text{Traditional\_Overhead} = O_{NH} + C_H + O_{HA}$$

2.0.3 Gap

Gap is the interarrival time of messages to an application on a receive and the interdeparture time of message from an application on a send. It is a measure of how well-pipelined the network protocol stack is. But gap is also a measure of how well-balanced the system is. If the host processor is processing packets for a receive very quickly, but the NIC cannot keep up, the host processor will starve and the gap will increase. If the host processor is not able process packets quickly enough on a receive, the NIC will starve and the gap will increase. If the network is slow, both the NIC and host will starve. Gap is a measure of how well-balanced the system is. As we minimize gap, we balance the system.

$$Gap = max(\frac{C_N}{R_N}, \frac{C_H}{R_H}, L_W) \\ -min(\frac{C_N}{R_N}, \frac{C_H}{R_H}, L_W)$$

2.0.4 Latency

Latency is modeled as

$$\text{Latency} = \frac{C_N}{R_N} + L_{NH} + \frac{C_H}{R_H} + L_{HA} + L_{NA} + \frac{C_A}{R_H} + L_W$$

. However, all methods will only use some of the communication patterns to process the protocol. Traditional network protocols, for example, will not use the communication path between the NIC and the application and does no processing at the application.

$$\text{Traditional\_Latency} = \frac{C_N}{R_N} + L_{NH} + \frac{C_H}{R_H} + L_{HA}$$

3 Interrupt Coalescing using EMO

There are several options for reducing interrupt pressures due to communication. Most conservatively, one can coalesce interrupts. The use of algorithms to reduce (or coalesce) the number of interrupts has been widespread. In this approach, the receiving NIC only interrupts the host after a specified amount of time or number of arriving packets. We model the latency and overhead of a message when there is no interrupt coalescing occurring, when an ideal interrupt coalescing pattern is occurring and when the specified number of packets to wait before an interrupt is very high.

3.1 Overhead

Figure 2 graphically represents the protocol processing overhead for traditional UDP using no interrupt coalescing and for UDP using maximum interrupt coalescing.

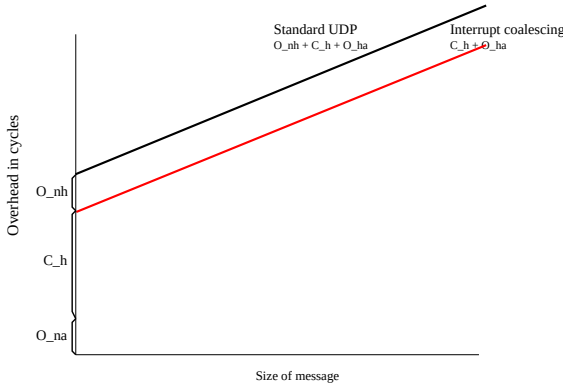

Figure 2. Extensible Message-Oriented Offload - overhead

Recall that traditional overhead is modeled as

$$\text{Traditional\_Overhead} = O_{NH} + C_H + O_{HA}$$

Interrupt coalescing amortizes the cost of the O_{NH} over many messages. In order to model advantages of interrupt coalescing in EMO, the overhead is measured in the limit as O_{NH} approaches zero.

$$\text{Interrupt\_coalescing\_Overhead} = C_H + O_{HA}$$

Interrupt coalescing still requires the copy between the operating system and the application and so overhead is still linear in the size of the message.

3.2 Latency

Figure 3 graphically represents the latency of traditional UDP, and provides a range of latencies for UDP messages with interrupt coalescing.

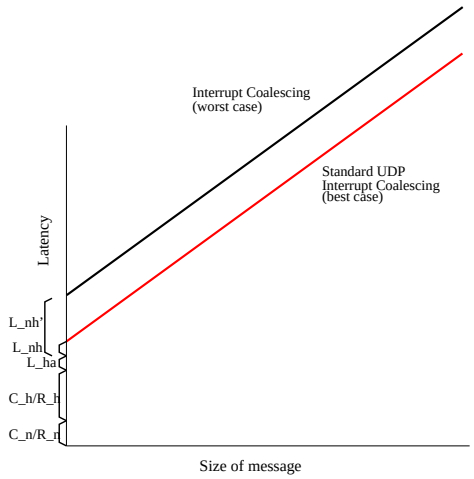

Figure 3. Extensible Message-Oriented Offload - Latency

Interrupt Coalescing decreases the overhead by waiting to interrupt the operating system until a number of messages have arrived for processing. While this decreases the amount of protocol processing overhead, it actually increases latency. Figure 3 graphically represents the best case latency for the interrupt coalescing method as the same as the latency of a message processed using a traditional UDP stack. Another latency for interrupt coalescing is also represented. Here, the slope of the line remains the same but the exact offset from the traditional latency will depend on the amount of time the interrupt coalescing mechanism waits before interrupting the host. Thus, L_{nh} will depend on the implementation of the interrupt coalescing mechanism.

4 Model Verification - Initial Results

We measured latencies by creating a ping-pong test between Host A and Host B. Host A remains constant throughout the measurements. Host A is a 933 MHz Pentium III running an unmodified Linux 2.4.25 kernel with the Acenic Gigabit Ethernet card set to default values for interrupt coalescing and transmit ratios. Host B is the same machine connected to Host A by cross-over fiber. Host B also runs an unmodified version of the Linux 2.4.25 kernel.

We measured overhead by modifying our ping-pong test. Host A continues the ping-pong test, but Host B includes a cycle-soaker that counts the number of cycles that can be completed while communication is in progress.

4.1 Latency

In order to validate the model for latency, we measured actual latency and approximated measurements for the various parts of the sum for our equation:

$$\text{Traditional\_Latency} = \frac{C_N}{R_N} + L_{NH} + \frac{C_H}{R_H} + L_{HA}$$

Our model is verified to the extent that the sum of the addends approximates the actual measured latency.

4.1.1 Application to Application Latency

In order to measure the traditional latency, we ran a simple UDP echoserver in user space on Host B. Host A simply measures ping-pong latency for various size messages. We measured this latency from 100 byte messages through 8900 byte messages. We remain within the jumbo frame size to avoid fragmentation and reassembly or multiple packets, but exercise the crossing of page boundaries. The page size for the Linux 2.4.25 kernel is 4KB.

We configured two 933 MHz, Linux 2.4.0(release) servers with version 0.49 of Jes Sorenson's Acenic driver (patched only to support tracedumps). The machines were connected by a cross-over fiber cable (with no switch).

The Acenic driver has four configurable parameters, the receive-side interrupt threshold, the send-side interrupt threshold, the receive-side maximum interrupt inter-arrival time and the send-side maximum interrupt inter-arrival time. The default parameters were used for the default interrupts. To disable interrupt coalescing, the receive-side interrupt threshold and the send-side interrupt threshold were set to 0 and the maximum interrupt inter-arrival times were set to 0. For maximum interrupts, the receive-side and sender-side interrupt thresholds were set to 1000 and the maximum interrupt inter-arrival times were set to 100μs.

4.1.2 Application to NIC Latency

In order to measure application to NIC latency we moved the UDP echo server into the Acenic firmware. This allows us to measure the latency of a message as it travels through Host A, across the wire, and to the NIC on Host B. This latency should not reflect the cost of the interrupt on Host B, the cost of moving through the kernel receive or send paths on Host B, nor the cost of the copy of data into user space on Host B. The UDP echoserver exercises all of the code in the traditional UDP receive and send paths in the Acenic firmware with the exception of the DMA to the host. Because of the structure of the Acenic firmware, we had to include a bcopy (byte copy) of the entire message from the receive buffer to a transmit buffer. This copy is performed at the speed of the Acenic processor (88 MHz). We assume that the application to NIC latency measurement includes both the copy of entire message and the $\frac{C_N}{R_N}$ portion of the latency calculation. It is important to note that the startup time for the DMA engine on the Acenic cards is approximately 5*mu*s. This will be accounted for in the L_{NH} portion of the calculation.

4.1.3 Application to Kernel Latency

For our initial results, we chose to measure the latency between an application on Host A and the kernel on Host B using the UDP echo server utility already provided by the inetd daemon. This should give a reasonable approximation of the latency between the application and the kernel. While the UDP echo message does not travel the exact code path as a UDP message in the kernel, it does exercise the same IP path and very similar code at the level above IP. The UDP echo server does not perform a copy of data to user space and does not perform a route lookup.

The application to kernel latency was measured with an unmodified Host A with default interrupt coalescing and with Host B with default interrupt coalescing, no interrupt coalescing, and maximum interrupt coalescing. We expect that the differences between interrupt coalescing and no interrupt coalescing should be present at this level.

4.1.4 Results

We validate EMO in two ways. First, we show that the assumptions we have made about the variables in our latency equation are accurate. Second, we validate that our comparison of interrupt coalescing schemes is accurate.

Figure 4 represents our verification that the overall latency is reflected in the variables of its equation and shows the minimum latency of various parts of the protocol using default interrupts on both Host A and Host B. The ToNIC latency includes the bcopy of data from the receive buffer on the Acenic to the transmit buffer. The projected ToNIC latency is calculated by determining the approximate *u*sec required to perform the bcopy and subtracting that from the actual latency. We assumed the number of cycles needed per byte to perform the copy was 4.

The latency represented in the results can be modeled through EMO as:

$$X + \frac{C_N}{R_N} + L_{NH} + \frac{C_H}{R_H} + L_{HA}$$

where X is the latency of the message as it travels the protocol stack during send and receive on Host A and as it travels the wire. The projected ToNIC latency includes both the processing on the NIC, $\frac{C_N}{R_N}$, and the time on Host A and the wire, X. The difference between the projected ToNIC latency and the the ToKernel latency is L_{NH} and $\frac{C_H}{R_H}$

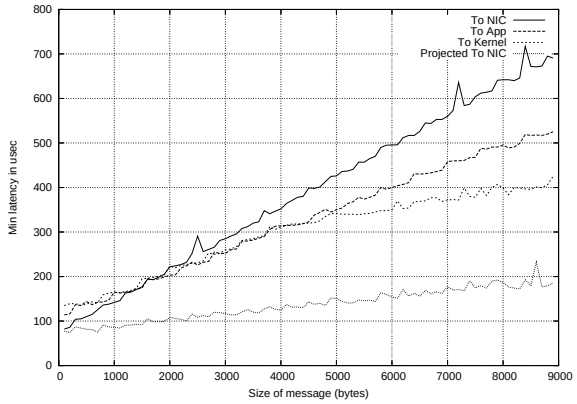

Figure 4. Latency with Default Coalescing

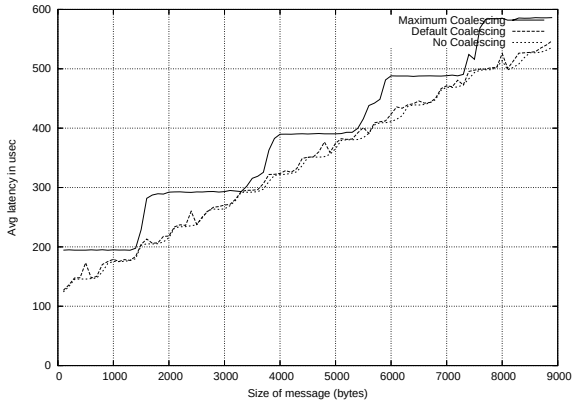

Figure 5. Application to Application Latency

and the difference between the ToKernel latency and the ToApp latency is L_{HA}. As we expected, the slope of the projected ToNIC and ToKernel are the same since we expect the interrupt latency to be constant. Also as we expected, the slope of the ToApp latency increases more quickly as it is dependant on the message size during the crossing of the wire X which is reflected in the ToKernel slope and on the copy from kernel to user space L_{HA}. The model, and the assumptions we made about the variables, are verified by the results.

Figure 5 represents our verification that the comparison of offload schemes using EMO is accurate. The expectation is that the average latency will be generally smaller when there is no interrupt coalescing. This is shown in our model. Interrupt coalescing can be seen as a move to decrease the overhead effect of the interrupt O_{NH} at the cost of the time that an interrupt will reach the host. This means we expect that L_{NH} is the variable affected. Figure 5 shows that generally latency is slightly lower when Host B disables interrupt coalescing and the latency for messages is higher when the maximum interrupt coalescing is enabled.

Moreover, Figure 6 further isolates this phenomenon by measuring the ping-pong measurement without the final move from kernel to application space. If we let X be the latency of all communication on Host A, the wire and the NIC on Host B, then Figure 6 represents a comparison between:

$$X + L_{NH} + \frac{C_H}{R_H}$$

and the interrupt coalescing latency

$$X + L'_{NH} + \frac{C_H}{R_H}$$

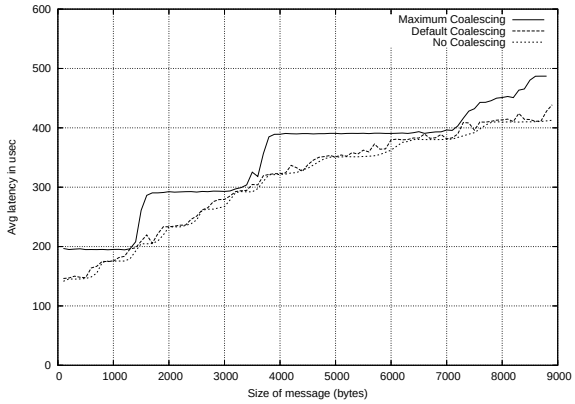

Figure 6. Application to Kernel Latency

4.2 Overhead

In order to validate the model for overhead, we measured actual overhead and approximated measurements for the various parts of the sum for our equation:

$$\text{Traditional\_Overhead} = O_{NH} + C_H + O_{HA}$$

Our model is verified to the extent that the sum of the addends approximates the actual measured overhead.

4.2.1 Application to Application Overhead

We measured the amount of cycle-soak work Host B can do without any communication occurring. Then we measured the amount of cycle-soak work Host B can do with standard ping-pong communication of various sized messages occurring between an application on Host A and a UDP echoserver on Host B. The difference between these to amounts of work is the overhead associated with the com-

munication. It is the the number of cycles being taken away from calculation.

We measured the overhead of application to application communication with default interrupts on Host A and with default interrupts on Host B, no interrupt coalescing on Host B, and maximum interrupt coalescing on Host B. We expect that the overhead of application to application communication when Host B is using interrupt coalescing will be lower than when Host B is not using interrupt coalescing.

4.2.2 Kernel to Application Overhead

In order to measure the overhead for kernel to application communication, Host A ran a ping flood on Host B and Host B ran the cycle-soak work calculation. We expect that interrupt coalescing will still make a difference at this level of communication so that Host B with no interrupt coalescing will have higher overhead than Host B with default interrupt coalescing. However, we do not expect the size of the message to make as much of a difference in the communication overhead at this level as it does at the application to application communication level.

4.2.3 NIC to Application Overhead

In order to measure the overhead for application to NIC communication, Host B is run with the modified Acenic firmware with the UDP echoserver at the NIC level. Host A runs the UDP ping-pong test and Host B runs the cycle-soak work calculation. We expect quite low overhead on Host B as there is no host involvement with the communication and therefore no communication overhead.

4.2.4 Results

As expected, there was no communication overhead when the UDP echoserver runs at the NIC level. This verified our modeled expectations. We expected that the overhead for communiction would be lower when Host B employed interrupt coalescing. Figure 7 shows that the difference is negligible at best, but this reflects general results regarding interrupt coalescing and its efficacy in lowering overhead[3]. Moreover, Figure 7 shows that this effect occurs at thekernel to application communication path as expected since the interrupt is still present in this path. As the size of the message approaches 9000 bytes, the number of interrupts decreases and so the overall communication processing overhead also decreases. This is true for all schemes but the maximum interrupt coalescing scheme which utilizes large delays in processing interrupts to flatten out the overhead curve.

Figure 8 shows the gap that represents O_{HA} increases as the size of the message increases as expected. This verifies the assumptions in our EMO model. However, the overall overhead for application to application messages remains

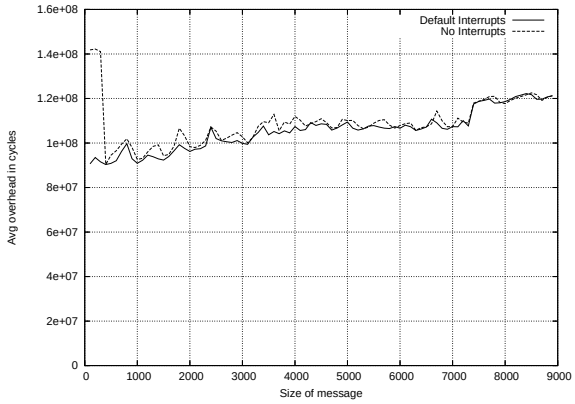

Figure 7. Kernel to Application Overheads

constant. We expect that as messages become larger, the increase cost of overhead associated with the copy of data to the application (O_{HA}) is just offset by the decrease cost of overhead associated with fewer interrupts (O_{NH}). Measurements for much larger messages should reveal application to application overheads that begin to slope up with the size of the message, especially since the savings in interrupts in maximized at 9000 bytes. These results should also bring a more clear understanding of the role of the memory subsystem in EMO.

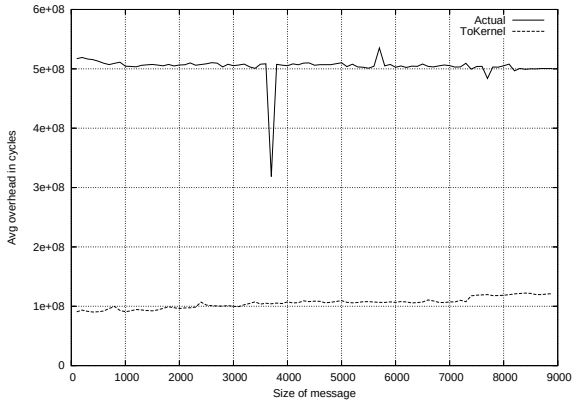

Figure 8. Overhead with Default Interrupt Coalescing

5 Conclusions and Future Work

The extensible message-oriented offload model (EMO) allows us to explore the space of network protocol implementation from the application messaging layers through to the NIC on a message by message basis. This new model

gives us a fresh understanding of the role of offloading in terms of overhead, latency and gap in high-performance systems.

The preliminary work begins to validate the EMO model and its assumptions. Generally, both the latency and the overhead equations are verified by actual measurements. Also, the comparison between various interrupt coalescing schemes further verifies EMO. Future work will include verification through gap measurements and overhead measurements for large messages.

EMO as a model for exploring offload design is already being used. We plan on using EMO to bound the resource requirements for NICs or TCP offload engines at 10Gb/s and 40 Gb/s speeds. We plan also to extend EMO to include memory management considerations such as caching.

References

[1] D. E. Culler, R. M. Karp, D. A. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a realistic model of parallel computation. In *Principles Practice of Parallel Programming*, pages 1–12, 1993.

[2] P. Gilfeather and A. Maccabe. Modeling protocol offload for message-oriented communication. In *Proceedings of IEEE Cluster 2005*, Boston, MA, September 2005.

[3] P. Gilfeather and T. Underwood. Fragmentation and high performance ip. In *Proc. of the 15th International Parallel and Distributed Processing Symposium*, April 2001.

[4] P. Shivam and J. Chase. On the elusive benefits of protocol offload. In *SIGCOMM workshop on Network-I/O Convergence: Experience, Lessons, Implications (NICELI)*, August 2003.