InterSense: Interconnect Performance Emulator for Future Scale-out Distributed Memory Applications

Qi Wang[1,2], Ludmila Cherkasova[1], Jun Li[1], Haris Volos[1]
[1]Hewlett-Packard Labs
[2]The George Washington University
interwq@gwu.edu, lucy.cherkasova@hp.com, jun.li@hp.com, haris.volos@hp.com

***Abstract*—A common approach for improving application performance is to process its working set from memory. For datasets that do not fit into DRAM of a single machine this leads to a design of scale-out applications, where the application dataset is partitioned and processed by a cluster of machines. Performance of distributed memory applications, implemented using MPI (Message Passing Interface), inherently depends on performance of communication layer, which is largely defined by performance characteristics of underlying interconnect. During last couple years, many Big Data applications, e.g., Hadoop, Spark, Memcached, were re-written to take advantage of Remote Direct Memory Access (RDMA) technology and RDMA-capable interconnects which provide fast and high-bandwidth communications. The application analysis of potential performance improvements due to faster and higher bandwidth interconnects is a challenging task. Does the existing application implementation take a full advantage of the underlying interconnect or not? Will the application performance get worse if the interconnect has X% increased latency or Y% lower bandwidth? In this work[1], we introduce a novel emulation framework, called *InterSense*, which is implemented on top of existing high-speed interconnect, such as InfiniBand, and which provides two performance knobs for changing the interconnect bandwidth and latency. This approach offers an easy-to-use framework for a sensitivity analysis of complex distributed applications to communication layer performance instead of creating customized and time-consuming application models to answer the same questions. We evaluate the emulator accuracy with popular OSU MPI benchmarks: *InterSense* emulates the specified *bandwidth and latency* values with less than 2% error between the expected and measured values. We apply *InterSense* for sensitivity analysis of two new benchmarks, such as *GUPS* and *Graph 500* to demonstrate the emulator's ease of use in getting non-trivial insights.**

I. Introduction

With the exponential growth of online data and proliferation of data-centric applications (Big Data analytics), there is an increasing pressure to revisit assumptions and requirements of the future system design [6], [2]. This forces the application designers to get ready for related challenges: to re-implement, tune, and optimize current and future applications to efficiently utilize underlying hardware and its performance characteristics.

With scale-out applications, each machine handles a portion of the complete dataset, and needs to communicate with each other to synchronize the main processing phases. Message passing interface (MPI) is the standard programming paradigm for scale-out, distributed memory applications. Performance of distributed memory applications inherently depends on performance of communication layer in the cluster. Over the last decade, there is a trend to replace traditional networking by high-speed interconnects with Remote Direct Memory Access (RDMA) technology for optimizing performance of distributed memory applications. In last two years, many Big Data applications, such as Hadoop, Spark, Memcached, etc., were re-written to take advantage of high-performance RDMA-capable interconnects [14], [15], [10], [9]. Moreover, depending on the performance characteristics of the underlying interconnect, different implementation decisions on communication style and size of transfers are made as a part of program optimization [8], [7], [11].

Complex MPI-based programs might interleave communication portions with computational ones in different patterns which makes it difficult to perform an accurate analysis of a communication layer impact on application performance and predict scaling properties of these programs. Building an accurate application model for predicting the application performance as a function of bandwidth and latency of underlying interconnect could be a very challenging and time-consuming task. Typically, such customized application model requires deep understanding of application functionality and its implementation, and could additionally necessitate detailed application profiling. Currently, it is practically impossible to analyze the application sensitivity to performance characteristics of the underlying interconnect and to answer the question: what impact the changed interconnect latency or/and bandwidth may have on performance of these applications?

To enable the analysis of an application dependency on performance characteristics of the underlying interconnect, we aim to offer an emulation framework, called *InterSense*, with *two performance knobs* for changing the interconnect perceived *bandwidth* and *latency*. In the paper, we discuss challenges for implementing the low-overhead emulation for high-speed interconnects and for decoupling latency and bandwidth emulations. We evaluate the emulator accuracy with popular OSU MPI benchmark suite [4] and two cluster-testbeds deployed with different generation InfiniBand interconnects (DDR and FDR): *InterSense* emulates the specified *bandwidth and latency* values with less than 2% error between the expected and measured values. To demonstrate the *InterSense*'s ease of use, we present a case study, where we apply *InterSense* for sensitivity analysis of two new popular benchmarks, such as *RandomAccess memory benchmark (GUPS)* [3] and *Graph 500* benchmark [1]. The remainder of the paper presents our results in more detail.

II. *InterSense* Emulator: Design and Implementation Challenges

InfiniBand is the state of the art approach for high-speed interconnect between multiple machines. It can achieve bandwidth above 100 Gb/s and latency lower than 1 microsecond. These performance characteristics prevent the use of existing (traditional) networking emulators, such as *ModelNet* [13], *Netbed* [16], or $netem$ [5], for the interconnect emulation due to their high overhead. The high-speed interconnect emulation poses a number of challenges:

[1]This work was completed during Qi Wang's internship at HP Labs.

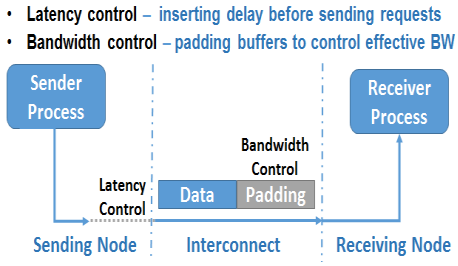

Fig. 1. Emulation Mechanism Overview.

Padding Layer	Pluses and Minuses
Application	- : Application specific - : Modification complexity
MPI library	+: Portable for MPI applications +: MPI library support – MPI information available - : High complexity of MPI implementations
IB driver library	+: Accuracy – padding right before sending - : Complexity – missing high-level library support - : Non-portable – device specific

Fig. 2. Bandwidth Control Layer.

Delay Insertion Layer	Pluses and Minuses
Application	+: Trivial to implement - : Application specific - : Inaccurate for asynchronous requests
MPI library	+: Portable for MPI applications - : High complexity of MPI implementations - : Not close to hardware (Infiniband)
IB driver library	+: Accuracy – delay right before sending - : Non-portable – device specific

Fig. 3. Latency Control Layer.

- Because of the high-speed nature of interconnect, the emulation overhead needs to be minimized.
- Latency and bandwidth characteristics of the interconnect emulation need to be orthogonal to each other, i.e., the emulation mechanism of one characteristic should not interfere with the other.
- There is no bandwidth or latency control support in existing hardware, e.g., there are no hardware knobs in InfiniBand adapters/switches that can be used for emulation of different performance characteristics.

To implement the low-overhead emulation and to decouple latency and bandwidth emulations, we designed the following software techniques for bandwidth and latency control as summarized in **Figure 1**:

- *For bandwidth emulation*, we add padding packets to reduce an effective bandwidth for applications.
- *For latency emulation*, we insert a software emulated delay before sending the application's messages.

Bandwidth Control: we impact effective bandwidth of the interconnect by sending padding packets. The ratio between padding packets and data packets determines the effective bandwidth for applications. It is defined in a software emulation layer so that we can achieve a fine-grained control over effective bandwidth. There are multiple layers in software, where bandwidth can be impacted: 1) an application, 2) a communication library, i.e., MPI library, or 3) a device driver.

Figure 2 summarizes advantages and disadvantages of each layer for bandwidth throttling. If we implement an interconnect bandwidth emulation in the *application layer* this requires only modification to the application itself. However, this may cause many lines of changes to the application code (and in many cases, the application source code might not be available). Also, for some MPI operations that perform both computation and communication, e.g., *MPI_Allreduce*, it is not possible to impact the interconnect bandwidth accurately because the communication pattern, implemented in the underlying MPI library, is transparent to the application.

On the other end, in the *device driver layer* (e.g., driver library of InfiniBand adapters), we would communicate with devices directly. However, at this layer, we are losing a higher level operation view, and have no information from MPI library. In addition, the driver library is *device specific*, and the process of setting up and sending of padding packets are two rather complex operations. This means that if the bandwidth control is done in the driver layer, migrating the emulation platform to a different hardware would be difficult.

We believe the **MPI library layer** offers the *best trade-off* among portability, accuracy and complexity: it works for all MPI-based applications, it is close to hardware (right above the driver layer), and it provides additional flexibility of seeing the MPI operators used by the program. This information can be used in the emulator for exploiting different bandwidth values across the interconnect links to mimic special interconnect topologies or resources available to the program.

To emulate correct bandwidth impact on packet latencies (i.e., trying to isolate interconnect bandwidth throttling from interfering with the interconnect latency), a padding packet is sent before the corresponding data packet. To avoid impacting the latency of small packets, their padding packets are batched (combined together) until the total size reaches a pre-defined threshold (64 KB in our experiments). The threshold parameter should be large enough to amortize software overhead, while not too large that bandwidth is not controlled in time. We found that for InfiniBand FDR interconnect with 56 Gbits/sec, 64 KB threshold is sufficient to ensure both no latency interference for small packets and the accurate bandwidth control.

Latency Control: since high-speed interconnects have an ultra low native latency (e.g., 1 μs for InfiniBand), the emulation overhead needs to be very low as well. This means that expensive operations like context switches cannot be involved. Our approach for emulating the interconnect latency is to generate an additional delay by *spinning*, i.e., by introducing an extra *idle time* for desired latency. A spinning approach has the following advantages: 1) a high accuracy (close to a cycle granularity), 2) a low implementation complexity, and 3) a low overhead (no other operations are needed). The only disadvantage is that a spinning process results in additional consumption of CPU computation cycles. However, we believe that the computation cycles spent on spin is acceptable because the desirable emulation targets of the interconnect latency emulation are at nanosecond or microsecond level (depending on the interconnect).

Similar to the bandwidth control implementation, multiple layers in the software stack can be used for impacting the interconnect latency. **Figure 3** summarizes advantages and disadvantages of each layer. For latency control of InfiniBand, because of the spin implementation simplicity, we choose the **driver library layer**. In addition, this approach aims to maximize the emulation accuracy (despite the driver's portability limitation). When migrating the emulation to a different hardware platform, the effort of re-implementing the spin-based latency control in a different driver library is minimal – it is approximately 20 lines of code in our case.

III. Evaluation

We evaluate the effectiveness and accuracy of our emulator by using a popular OSU MPI benchmark suite [4] and two clusters with different generation interconnects: DDR InfiniBand (20 Gbits/s) and FDR InfiniBand (56 Gbits/s):

- *Cluster_1* with 4 nodes, where each node is based on HP DL380 servers (two sockets Xeon E5-2697 with 12 cores per socket), and connected with FDR InfiniBand (56 Gbits/s);
- *Cluster_2* with 8 nodes, where each node is based on HP Proliant BL460c servers (two sockets Xeon E5345 with 12 cores per socket), and connected with DDR InfiniBand (20 Gbits/s).

The Ohio MPI Microbenchmark suite [4] is a collection of independent MPI message passing performance microbenchmarks developed and provided as an open source by the Network-Based Computing Laboratory of the Ohio State University. In particular, it includes some simple benchmarks for performance measurements of latency and bandwidth for basic MPI communications.

OSU MPI bandwidth test is implemented by having a sender sending out a fixed number (equal to a window size) of back-to-back messages to a receiver and then waiting for a reply from the receiver. The receiver sends the reply only after receiving all the messages. This process is repeated for several iterations and the bandwidth is calculated based on the elapsed time (from the time sender sends the first message until the time it receives the reply back from the receiver) and the number of bytes sent by the sender. The objective of this bandwidth test is to determine the maximum sustained data rate that can be achieved at the network level.

From the experimental results on both clusters, we observe that the interconnect *bandwidth emulation* is supported with a high accuracy: less than 2% error between the expected, emulated interconnect bandwidth and the measured interconnect bandwidth in the experiments (packet size = 1 MB).

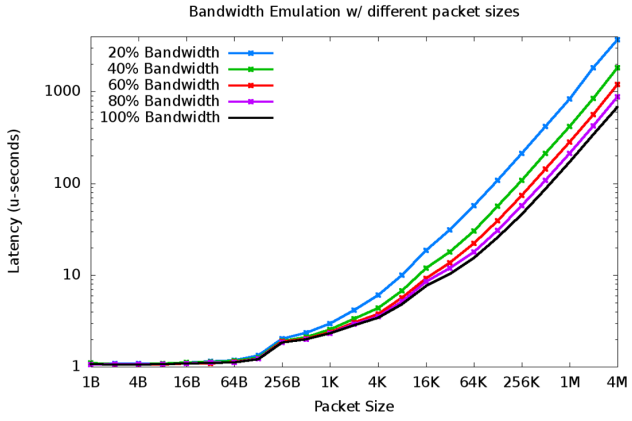

Fig. 4. Bandwidth Impact on Packet Latency.

We also measure the latency of different size packets under different emulated interconnect bandwidth. We find that these packets are controlled accurately in the emulator, and the bandwidth emulation is orthogonal to latency emulation. **Figure 4** shows that small packets (not limited by bandwidth) have no latency change under different emulated interconnect bandwidth control, while the latency of large packets (limited by bandwidth) is impacted correctly.

OSU MPI latency test is carried out in a ping-pong fashion. A sender sends a message with a certain data size to a receiver and waits for a reply from the receiver. The receiver receives the message from the sender and sends back a reply with the same data size. Many iterations of this ping-pong test are carried out and average one-way latency numbers are obtained.

The *latency emulation* results, measured with OSU MPI benchmark, again show high accuracy on both clusters: $\leq 2\%$ error between expected (emulated) latency and measured one.

Moreover, the designed mechanism works effectively for packets with different sizes as demonstrated in **Figure 5**. The bottom line on this graph shows the measured latency of different size packets. As we can see, when we emulate the increased interconnect latency (i.e., *latency+2*μs, *latency+4*μs, etc.) the measured packet latency closely follows the original latency pattern. It means that the designed latency emulation mechanism does not impact the interconnect bandwidth, and therefore the emulation mechanisms for latency and bandwidth do not interfere with each other.

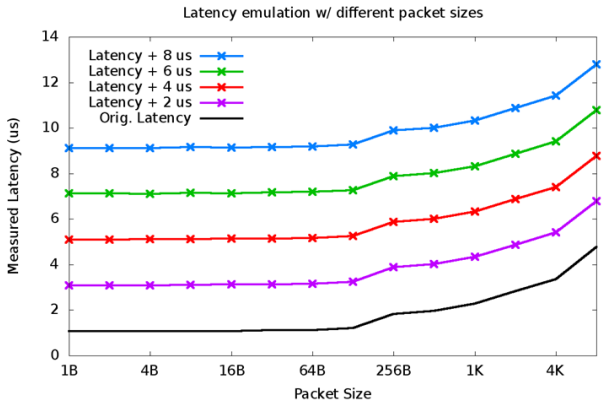

Fig. 5. OSU benchmark: Latency Emulation with Different Packet Sizes.

The results obtained in the DDR InfiniBand-based *Cluster_2* (with 20 Gbits/s links) show similar accuracy results. We omit them due to a paper space limitation.

To demonstrate the emulator's ease of use, we present a case study, where we apply *InterSense* for a sensitivity analysis of two new popular benchmarks, such as *RandomAccess memory benchmark (GUPS)* [3] and *Graph 500* benchmark [1]. Note, that the only other way to get the results below is to build customized performance models of these benchmarks as a function of interconnect latency and bandwidth, which is a difficult and challenging task even for a skilled performance analysts.

Figure 6 shows the sensitivity of *GUPS* to the interconnect latency. *GUPS* is a new benchmark proposed by IBM Research [12] a few years ago for measuring how frequently a computer can issue updates to randomly generated RAM locations. Giga-updates per second (*GUPS*) is a measure of random memory access capability of multicores platforms. *GUPS* is latency sensitive, but in a very special way. Its sensitivity is defined by the number of outstanding concurrent requests. If the number of outstanding requests is limited ($\leq 1K$) then *GUPS* performance is 30-100% worse with increased interconnect latency. However, for outstanding requests $\geq 4K$ there is no difference in performance: the pipeline of processed requests is constantly full, and it hides the increased interconnect latency.

GUPS is not bandwidth sensitive in our experiments: it operates with very small size requests and cannot utilize the available interconnect bandwidth in our *Cluster_1* testbed. Small transfers are limited by the interconnect message rate.

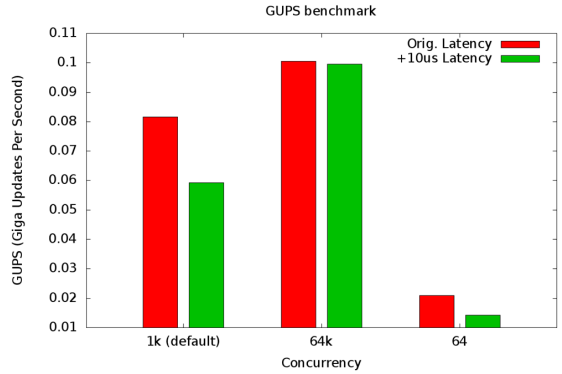

Fig. 6. GUPS: Emulated Latency.

So, if *GUPS* (hypothetically) is an application a user is interested to execute in a given cluster then he/she could had a lower bandwidth interconnect for getting a similar performance.

Figure 7 shows performance of *Graph 500* benchmark [1] with emulated interconnect bandwidth. *Graph 500* is a new benchmark which implements a *Breadth First Search* algorithm on large graphs. This benchmark is used for assessing system performance based on processing efficiency of their memory and interconnect. Producing winning results is a goal for many companies with leading hardware and system design. Figure 7 shows that only 30% of FDR InfiniBand bandwidth is effectively used (for selected graph and cluster sizes). With 10% of interconnect bandwidth, the execution time (red line) is increased by 30%, and a fraction of communication time (blue line) is doubled. Again, this type of sensitivity analysis is very useful for understanding the cost/performance trade-offs.

Graph 500 was not sensitive to a higher (10x) interconnect latency in our experiments (due to using asynchronous communication and high messaging volume).

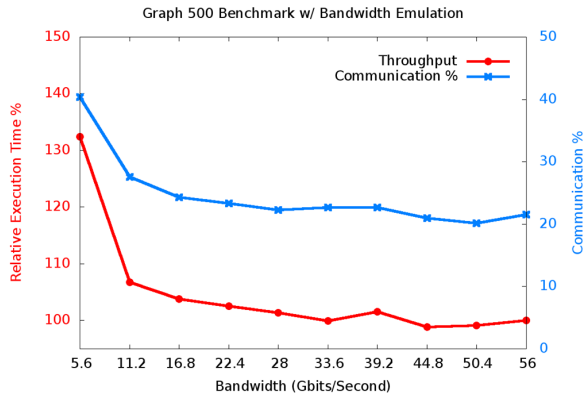

Fig. 7. Graph500: Emulated Bandwidth.

InterSense can also be used for answering capacity planning questions which require careful estimates of the available "remaining" interconnect capacity for supporting a higher load/volume service without compromising its performance.

This concludes our case study with *InterSense*. The goal of this study was to show the ease of tool use and the appeal of the designed interconnect emulator for performing the sensitivity analysis of emerging benchmarks and modern applications instead of creating customized and time-consuming application models to answer the same questions.

IV. Conclusion and Future Work

In this work, we introduce novel bandwidth and latency control mechanisms for performance emulation of the high-speed interconnects. *InterSense* can assist researchers and engineers in emulating a variety of performance characteristics of future large-scale interconnects and conducting the application sensitivity and scalability analysis dependent on these characteristics. We are working on augmenting the proposed approach with additional profiling, modeling, and prediction technique. By performing the emulation in small deployments with increased interconnect latency and decreased bandwidth we aim to derive the predictive models for application performance when processing larger data amounts in large-scale distributed environments. We believe that the *InterSense* ability to accurately reflect the *needed* interconnect bandwidth and to show the application sensitivity to the increased interconnect latency will help in applications' optimization and their redesign.

References

[1] Graph 500 Benchmark. www.graph500.org/ .

[2] HP Labs. The Machine: A new kind of computer. http://www.hpl.hp.com/research/systems-research/ .

[3] HPCC RandomAccess (GUPS) Benchmark. http://icl.cs.utk.edu/projectsfiles/hpcc/RandomAccess/ .

[4] MVAPICH Ohio State University Micro benchmark. http://mvapich.cse.ohio-state.edu/benchmarks/.

[5] netem, http://www.linuxfoundation.org/collaborate/workgroups/networking/netem .

[6] K. Asanovic. FireBox: A Hardware Building Block for 2020 Warehouse-Scale Computers. In *Proc. of FAST*, 2014.

[7] F. Checconi and F. Petrini. Traversing Trillions of Edges in Real Time: Graph Exploration on Large-Scale Parallel Machines. In *Proc. of Intl. Parallel and Distributed Processing Symposium, IPDPS'14*, 2014.

[8] F. Checconi, F. Petrini, J. Willcock, A. Lumsdaine, A. R. Choudhury, and Y. Sabharwal. Breaking the Speed and Scalability Barriers for Graph Exploration on Distributed-Memory Machines. In *Proc. of Conference on High Performance Computing Networking, Storage and Analysis, SC'12*,, 2012.

[9] J. Jose, H. Subramoni, M. Luo, M. Zhang, J. Huang, M. Wasi-ur Rahman, N. S. Islam, X. Ouyang, H. Wang, S. Sur, and D. K. Panda. Memcached Design on High Performance RDMA Capable Interconnects. In *Proc. of the 2011 International Conference on Parallel Processing*, ICPP '11, 2011.

[10] X. Lu, M. Wasi-ur Rahman, N. S. Islam, D. Shankar, , and D. K. D. Panda. Accelerating Spark with RDMA for Big Data Processing: Early Experiences. In *Proc. of Hot Interconnects*, 2014.

[11] X. Que, F. Checconi, and F. Petrini. Performance Analysis of Graph Algorithms on P7IH. In *Proc. of the 29th Intl. Conference on Supercomputing, ISC'14*, 2014.

[12] V. Saxena, Y. Sabharwal, and P. Bhatotia. Performance evaluation and optimization of random memory access on multicores with high productivity. In *Proc. of Intl. Conference on High Performance Computing (HiPC)*, 2010.

[13] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostić, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. *SIGOPS Oper. Syst. Rev.*, 36(SI), Dec. 2002.

[14] M. Wasi-ur Rahman, N. S. Islam, X. Lu, J. Jose, H. Subramoni, H. Wang, and D. K. D. Panda. High-Performance RDMA-based Design of Hadoop MapReduce over InfiniBand. In *Proc. of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing Workshops and PhD Forum*, IPDPSW '13, 2013.

[15] M. Wasi-ur-Rahman, X. Lu, N. S. Islam, R. Rajachandrasekar, and D. K. Panda. MapReduce over Lustre: Can RDMA-Based Approach Benefit? In *Proc. of the 20th International Conference EuroPar*, 2014.

[16] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. *SIGOPS Oper. Syst. Rev.*, 36(SI), Dec. 2002.