

Improving Communication Performance on InfiniBand by Using Efficient Data Placement Strategies

Robert Rex, Frank Mietke, Wolfgang Rehm
Technical University of Chemnitz, Dept. of Computer Science
09107 Chemnitz, Germany
{rex,mief,rehm}@informatik.tu-chemnitz.de

Christoph Raisch, Hoang-Nam Nguyen
IBM Deutschland Entwicklung GmbH
71032 Boeblingen, Germany
{raisch,hnguyen}@de.ibm.com

Abstract

Despite using high-speed network interconnection systems like InfiniBand, the communication overhead for parallel applications is still high. In this paper we show, how such costs can be reduced by choosing appropriate data placement strategies. For large buffers, we propose a transparent placing in hugepages as it can dramatically decrease memory registration overhead and may increase network bandwidth. Thus, we developed a new library that can be preloaded for applications at load time and cares about drawbacks of using hugepages. So we believe that it is the most suitable one in the HPC area for Linux today. But we do not only refer to large buffers as small communication buffers also play a significant role for application behaviour. We show that transfer latencies vary, depending on data placement. All current communication library implementations for InfiniBand do not utilize scatter-gather lists for send and receive operations, but we show that this feature can have a positive impact on latency for small buffers and data aggregation can perform better. Our results show that communication performance of applications may improve more than 10 % using the presented improvements.

1 Introduction

High speed interconnects like InfiniBand use DMA engines and user level communication protocols to achieve high bandwidth and low latency. Thus, the user application can directly send a communication request - which includes information like starting address and length of a commu-

nication buffer - to the appropriate network interface controller (NIC) without kernel intervention. This request is processed by this NIC and data will be transferred to the communication partner. As the DMA engine only handles physical addresses, a process must register its buffers beforehand, which means a translation of virtual addresses into physical ones. This is called memory registration and constitutes a very time consuming component in the communication path [7]. To reduce this overhead, several strategies have been proposed (e. g. lazy deregistration [9]) and implemented in communication libraries like MPICH2-CH3-IB [4]. There, a pool of already registered memory is hold, so that memory registration is done only once for each virtual address. Thus, it can dramatically increase performance of parallel applications [7]. This strategy has drawbacks too. One of them is that memory remains allocated to the application during their whole runtime. This can lead to less available physical memory as well as to an inflation of libc structures due to more memory management outlay. For reducing high memory registration costs and the former mentioned problems, we propose the transparent utilization of hugepages and show its effects on communication behaviour of application benchmarks in the HPC area. For small buffers, we consider an aligned data placement and the use of scatter-gather lists in communication libraries. Here, we also expect to decrease interfering overheads that occur on CPU and InfiniBand adapter communication.

The rest of the paper is organized as follows: Section 2 discusses the related work. The details how the hugepage library is designed and implemented is explained in section 3. Section 4 shows possible communication improvements for small buffers and Section 5 presents our obtained results,

which leads to our conclusions in Section 6. The last Section deals with future work and describes, where succeeding work can be directed to.

2 Related Work

Proposing hugepages for HPC is not a new suggestion and the support was introduced for Linux with kernel version 2.6. But a transparent use for applications was made feasible not before kernel 2.6.16 as this version allows private mappings of hugepages. Since the end of 2005, there have been two libraries freely available that transparently map in hugepages. We encountered drawbacks that we wanted to avoid for our applications. The first library *libhugepagealloc* [11] is not thread safe and does not assure locality between allocated buffers since every buffer is mapped into a separate hugepage. The second one [3] is named *libhugetlbfs* and wraps the internal libc function *morecore()*. There are two potential drawbacks: This library assures that every buffer that is allocated by the libc resides in hugepages. Furthermore, the libc allocator manages all requests. The former issue matters for the number of TLB misses. These may increase when using hugepages (see section 5) as some processors provide a large set of TLB entries for 4 KB pages (e. g. AMD Opteron: 544) but only a small number for hugepages (AMD Opteron: 8). Thus, we want to use these pages with caution. The latter mentioned issue influences the allocator performance. The libc allocator is a general purpose allocator and may not cover requirements that matter for HPC applications. For some instrumented applications we measured allocation benefits of up to 10 times with our library (e. g. for Abinet [10]). As the changed *morecore()* function of *libhugetlbfs* showed segmentation faults and hangups with some MPI applications, where we could not clarify the origin, it was not possible for us to compare this functionality with our library. To sum up, we believe that our library is the most suitable one for HPC applications on Linux today.

3 Hugepage Library

As already stated in section 1, the use of hugepages may show performance improvements for applications. We expect that the communication performance should increase, because the time consuming memory registration is mitigated. In this process, three important steps have to be done (see [8]):

1. All pages of the communication buffer have to stay in memory and must be pinned.
2. The virtual start address of each page has to be translated into a physical one.

3. The address translations have to be sent to the NIC.

With hugepages, less addresses have to be translated and - if the adapter supports the hugepage size - less address translations have to be sent to the NIC. Thus, this overhead decreases obviously. Other improvements of hugepages refer to a better utilization of system busses, CPU and memory (see section 5 for deeper details), e. g. prefetching units can benefit by better physical locality here.

3.1 Library Design

Our hugepage library will intercept all allocation calls that are issued by an application. If a request is smaller than 32 kB, the library calls the libc to handle it. Otherwise we map in hugepages. The library manages these memory areas in terms of free memory and used memory. The library follows a strict tier model. This modular design guarantees an easy interchangeability for each module that it consists of and simplifies the exchange of algorithms. These layers are the following:

1. *Layer for transparency*: Responsible for overwriting allocation functions (*malloc()* etc.) and for initialization, where the eponymous libc function symbols are resolved. For allocation requests less than 32 Kilobyte it calls the original libc functions.
2. *Layer for mapping/unmapping hugepages*: Responsible for communication with the *HugeTLBfs*, especially to map in/out hugepages to/from a process address space. It must leave a reserve of hugepages that are needed when forking processes for Copy-on-Write reasons.
3. *Layer for management of hugepages*: Here, all hugepage mapped memory is managed in terms of used and free memory areas.

In figure 1, the layering of the different functionalities is depicted schematically.

3.2 Library Details

We implemented our library in C. The order of execution steps is depicted in Figure 2, whereby the following conditions are met:

1. Requests with less than 32 kB are not mapped into hugepages due to our empirical memory registration measurements which showed better performance characteristics with small pages in this area. Another important point is that some processors show limitations in using hugepages (see Section 2). By only using hugepages, this will lead to an increase of TLB misses

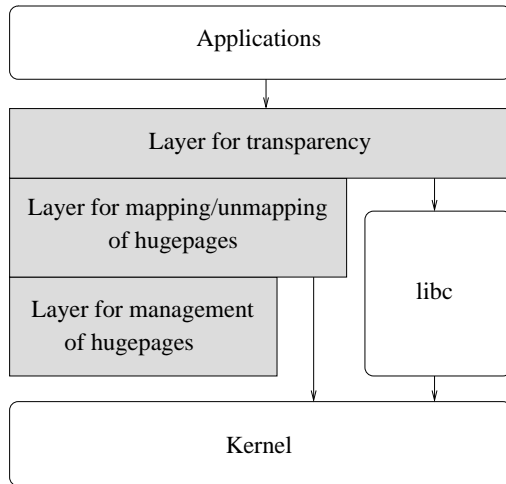


Figure 1. Relations of hugepage library

for applications that show irregular memory access patterns and thus will also show worse communication behaviour.

2. The library uses an address-ordered first fit allocator, which shows best performance values due to a good locality (see [12]). This is a different approach than the implementation of the *libhugetlbf*s, which utilizes the libc *morecore()* function and thus uses the libc allocator, and we measured that some HPC applications like Abinit raised a thrashing behaviour into the libc memory allocator. With Abinit, the time consumption of allocation/deallocation functions is significantly lower with our library compared to the libc allocator and it improved application runtime by 1.5 %.
3. The memory management structures are not located as a header or footer for each allocated buffer but in a "cache" that is created at initialization time and thus ensuring good locality when traversing the freelist.
4. To improve memory allocation time, we manage hugepages in chunks with a size of 4 Kilobyte. Using chunks of fixed size simplifies the memory management data structures and ensures a fast access in a complexity of $O(1)$.
5. The allocator does not coalesce free memory areas on *free()* calls. This avoids useless coalescing/splitting patterns, when applications allocate and deallocate buffers with the same size in a short time frame.

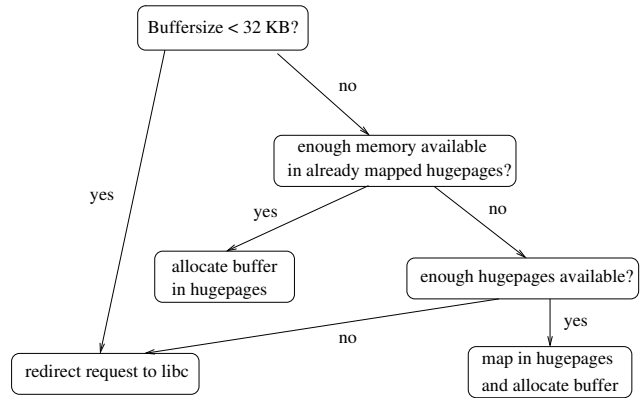


Figure 2. Order of executions for memory allocation with hugepage library

4. Communication Improvements with Small Buffers

This section deals with small communication buffers as these have other requirements to the communication network. For large buffers network bandwidth is more important, while for small buffers low latency is essential. We already mentioned the high memory registration overhead impacts the performance of protocol offloading network adapters. For small buffers other overhead gets important, especially the communication between CPU and network adapter. The detailed sequence of a work request is depicted below:

1. The consumer posts a send or receive work request.
2. The network adapter transfers the specified data to the communication partner.
3. After completion the adapter generates a completion queue entry.
4. The consumer is notified about work completion by polling the completion queue or by an interrupt.

In order to optimize this execution flow the so called scatter-gather mechanism provided by InfiniBand verbs and adapters can be used for sending multiple buffers with only one work request. The advantages are obviously:

- The consumer has to post only one work request.
- The network adapter can fetch buffers from the memory subsystem simultaneously without involving the CPU.
- Only one completion queue entry has to be polled for.

We implemented a test case that measures the duration of send and receive operations over OpenIB between two dedicated systems in terms of reliable connection based on the following parameters:

- *offset*, which is the start address of each data buffer in a memory page.
- *sge_size*, which denotes the size of a data piece in a scatter-gather element (SGE) in bytes.
- *sges*, which is the number of SGEs to be processed by a send operation or a receive operation, respectively. Thus, the total message size equals (*sges* * *sge_size*).

For each combination of those parameters this test case measures the elapsed time in time base register (TBR) ticks for post and poll operations separately. The post operation covers step 1, while the poll operation measures steps 2 - 4. We ran this test case on two IBM low-end System p with IBM InfiniBand eHCA driver on Linux. The time consumption of post operations is approximately constant for small and for large messages (1 byte - 512 kbytes) and varies between 72 - 135 TBR ticks, so we can assume a relatively constant overhead. With the usage of multiple SGEs, this overhead does not increase linearly, e. g. the time consumption by using 128 SGEs is only three times higher than with one SGE. Figure 3 shows our results with up to 8 SGEs. Time consumption is depicted in TBR ticks. The outlay for 1 SGE is relatively constant up to 512 Bytes and then grows linearly with buffer size. We see that up to 128 Byte, the sending of 4 SGEs with same sizes - the overall message size is 4 times higher than with one SGE - is only 14 % more costly. Thus, we believe that MPI implementations for InfiniBand may benefit in a perceptible way by using this feature. Especially *MPI.Pack()* and *MPI.Unpack()* may be mapped directly to this InfiniBand interface. A similar approach for MPI-I/O was analysed in [13].

We repeated our measurements with different buffer offsets in the first page, utilizing 1 SGE. Figure 4 shows the results for buffers with a size between 8 and 64 bytes. Between the offset range 1 to 128 Byte we see that the time consumption for posting a send request and polling for its completion differs up to 8 percent. It appears that the memory access of the InfiniBand apter or the underlying system I/O bus is optimized for certain offsets, e. g. at offset 64.

5 Benchmarks with Hugepage Library

For our benchmarks, we used several test systems with InfiniBand adapters and MVAPICH2 in version 0.9.2 as MPI library implementation:

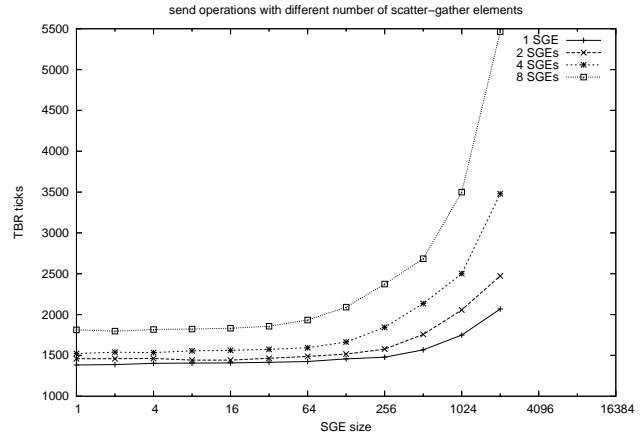


Figure 3. Work request duration with different number of SGEs

- AMD Opteron system with Mellanox InfiniHost on PCI-Express, 2 GB RAM, 2 dual-core processors (2.2 GHz)
- Intel Xeon system with Mellanox InfiniHost on PCI-X, 2 GB RAM, 2 hyperthreading processors (2.4 GHz)
- IBM low-end System p with IBM Infiniband eHCA on GX bus, 16 GB RAM, 8 processors (1.65 GHz)

The OpenIB stack is not able to detect hugepages as the kernel pretends 4 KB pages instead. So we modified it in a way to send hugepages to the adapter when those are used (the appropriate patch was sent to the OpenIB mailing list in August 2006). Furthermore, we used two benchmarks to measure the effect of hugepages: The first one - the IMB (Intel MPI benchmarks) [5] - is a microbenchmark, which tests MPI operations and presents its results in terms of bandwidth and latency. We decided to run the SendRecv test, as we wanted to see the maximum bandwidth numbers with and without hugepages. The second one - the NAS benchmark [1] in version 3.1 - provides representative problems for many HPC applications, so we could see a more complex program behaviour with different communication patterns.

5.1 Intel MPI Benchmarks

As stated above, we used the SendRecv test of the IMB and measured network bandwidth. We analysed two cases: One time we activated lazy deregistration and only measured the time for sending and receiving a message over InfiniBand. Another time we deactivated this feature so that we additionally measured memory registration over-

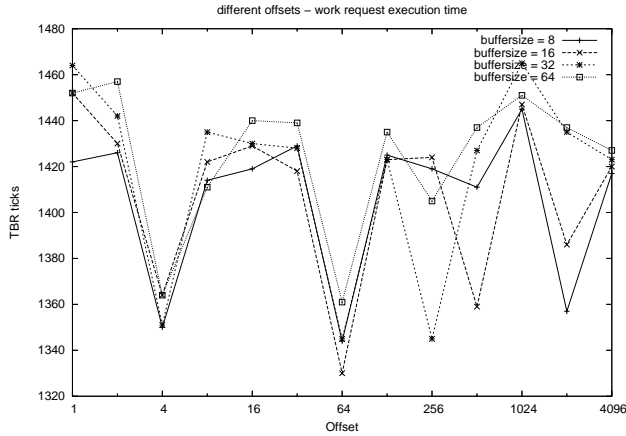


Figure 4. Work request duration with different offsets

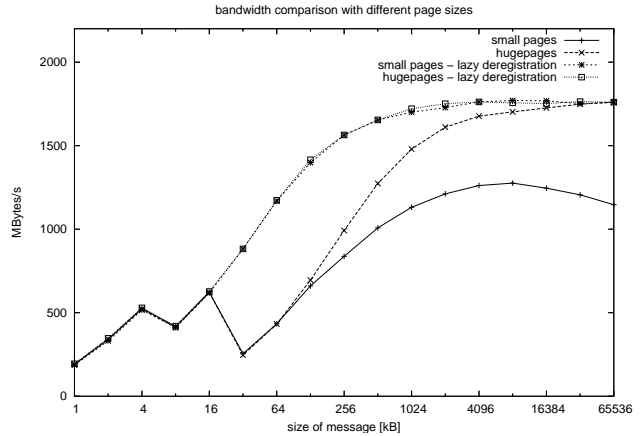


Figure 5. Intel MPI Benchmarks on AMD Optron with Mellanox InfiniHost

head for each test. The network efficiency of real applications is somewhere between these two cases - depending on the reuse of buffers for send or receive operations. Figure 5 shows the results that are explained below:

1. In the first test we deactivated lazy deregistration. The MPI library uses eager send up to a buffer size of 8 KB and the rendezvous protocol for greater buffers. For buffers larger than 16 KB, it uses the RDMA feature of InfiniBand so we only see memory registration effects for those buffers. Here, the effect of hugepage utilization is enormous, as memory registration time decreased extremely (down to 1 % of the time as with small pages as our performed measurements show). With hugepage mapped buffers greater than 4 MB size, we almost reach the maximum bandwidth of approximately 1750 MB/s. Even if lazy deregistration is enabled, the first use of a buffer results in a memory registration with an equal time consumption, according to these results.
2. In the second test we activated lazy deregistration. Here we only measure the time that is used for sending/receiving messages. The results show the same numbers for small pages as for hugepages. This is the contrary to our expectations as ATT (Address Translation Table) misses should have decreased on the InfiniHost adapter. This may be due to other bottlenecks in the system like memory bandwidth. We repeated our measurements on an Intel Xeon with lazy deregistration enabled and hugepage mapped buffers: One time, we used the unmodified OpenIB driver, so the adapter saw 4 KB pages, another time the modified OpenIB driver was used and 2 MB pages were sent.

The bandwidth with 2 MB pages increased up to 6 %, what could be due to less ATT misses on the InfiniHost adapter in this system.

5.2 NAS Benchmarks

In this section, we present our results regarding the NAS benchmark suite. We benchmarked 2 nodes with 4 processes each, so that we had an overall process count of 8. As the NAS benchmarks use huge amounts of the ELF BSS segment, we did not only preload our library for hugepage tests, but also used a linker script and a constructor function of the *libhugetlbfs* to map this segment into hugepages at startup time. We decided to run five of the class C benchmarks - only MG on AMD Optron represents a class B result - that are depicted in Figure 6. We obtained our measurements by utilizing the *mpiP* library [6], which is able to instrument MPI functions, giving a useful output at the end of each run depicting the time consumption. Thus, we are able to distinguish between communication and computation time. Except for MG and IS, all benchmarks show communication performance benefits of more than 8 % implying a significantly better network utilization. Overall, all benchmarks benefited from using hugepages - except for IS. One reason for communication improvement is the higher network bandwidth and lower message latencies due to more effective memory registration and better communication between the memory controller and the network adapter as the number of address translations decreases. This confirms our results in Section 5.1. As we see, there are also other benefits that are not caused by a decreasing communication time, but reducing computation time of

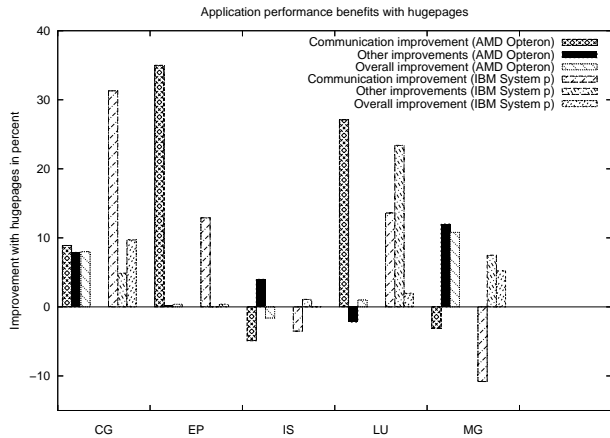


Figure 6. NAS benchmarks with hugepages

each process. To look for these improvements, we instrumented an AMD Opteron system with PAPI [2] to read the processor performance counters. We measured that TLB misses increased dramatically with hugepages (up to eight times with EP) except for LU. This shows that TLB misses are not responsible for less application time here and the improvement must be somewhere else. Maybe, the memory prefetching unit can benefit from larger physical contiguous areas. The effects on computation time are subject to further research, since the current measurements do not provide sufficient insight to explain the observations.

6 Conclusions

This paper showed how data placement strategies can significantly decrease communication overhead. For large buffers, we proposed a placement in hugepages, which can be done transparently with the library presented in section 3. We showed how protocol offloading network adapters can benefit from using greater pages, especially by decreasing memory registration overhead. Nevertheless we believe that less ATT (Address Translation Table) misses on the adapter for send/receive operations can also result in bigger network bandwidth due to less dispatched stalls as already showed for Myrinet adapters in [14], but we could reconstruct these effects only with the Intel Xeon system. Despite of higher expectations for microbenchmarks in this area (here: IMB) we showed in section 5.2 that real applications may benefit in a perceptible way. We could show performance improvements for communication time as well as for computation time. Thus, hugepages are a promising way for HPC applications as they may result in a better utilization of system resources. The results show time improvements of more than 10 % and we believe that with a further analysis (see section 7), remaining

bottlenecks can be made visible.

7 Future Work

As our presented results did not cover all aspects of hugepage utilization - we only stressed communication effects in this paper - we believe that more analysis needs to be done. In section 5 we showed that hugepages can have bad effects on computation time, but a deeper investigation of this effect needs to stress the point on system architecture. Especially the processor internals and the memory bus architecture must be observed to show requirements for applications that use hugepages. Yet, we showed that the TLB of AMD Opteron does not suit perfectly here, because TLB misses will increase. To explain the side effects, using our hugepage library, on computation time of applications, we plan to analyse their runtime behaviour more detailed. Moreover, we plan to implement the use of scatter-gather lists in the MPICH2-CH3-IB device to show the performance benefits of this approach.

Acknowledgements

This work was significantly improved by the valuable discussion with Mario Trams, former member of the research staff of the Chair of Computer Architecture, and by Torsten Mehlan from Chemnitz University of Technology, who reviewed our work carefully and provided new ideas and suggestions.

List of Trademarks

IBM and IBM System p are trademarks of International Business Machines Corporation in the United States, other countries, or both.

Xeon is a trademark of Intel Corporation in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

References

- [1] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, D. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A.

- Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrisnan, and S. K. Weeratunga. The NAS Parallel Benchmarks. *The International Journal of Supercomputer Applications*, 5(3):63–73, Fall 1991.
- [2] Browne, S., Deane, C., Ho, G., Mucci, P. PAPI: A Portable Interface to Hardware Performance Counters. *Proceedings of Department of Defense HPCMP Users Group Conference*, June 1999.
- [3] David Gibson and Adam Litke. Libhugetlbfs. <http://sourceforge.net/projects/libhugetlbfs>.
- [4] R. Grabner, F. Mietke, and W. Rehm. An MPICH2 Channel Device Implementation over VAPI on InfiniBand. *Workshop on Communication Architecture for Clusters*, 2004.
- [5] Intel GmbH, Hermuelheimer Str. 8a, D-50321 Bruehl, Germany. *Intel MPI Benchmarks – Users Guide and Methodology Description*.
- [6] Lawrence Livermore National Laboratory. mpiP: Lightweight, Scalable MPI Profiling. <http://www.llnl.gov/CASC/mpip/>.
- [7] F. Mietke, R. Rex, T. Mehlan, T. Hoefler, and W. Rehm. Reducing the Impact of Memory Registration in InfiniBand. In *KiCC - Workshop Kommunikation in Clusterrechnern und Clusterverbundsystemen*. Department of Computer Science, Chemnitz University of Technology, 2005.
- [8] R. Rex. *Analysis and Evaluation of Memory Locking Operations for High-Speed Network Interconnects*. October 2005.
- [9] H. Tezuka, F. O’Carroll, A. Hori, and Y. Ishikawa. *Pin-down Cache: A Virtual Memory Management Technique for Zero-copy Communication*. 1998.
- [10] The ABINIT Group. Abinit. www.abinit.org.
- [11] J. Treibig. Libhugepagealloc. <http://www10.informatik.uni-erlangen.de/Research/Projects/DiME/libhugepagealloc.html>.
- [12] P. R. Wilson, M. S. Johnstone, M. Neely, and D. Boles. *Dynamic Storage Allocation: A Survey and Critical Review*. Department of Computer Sciences, University of Texas at Austin, 1995.
- [13] J. Wu, P. Wyckoff, and D. Panda. Supporting Efficient Non-contiguous Access in PVFS over InfiniBand, 2003.
- [14] X. Zhou, Z. Huo, N. Sun, and Y. Zhou. Impact of Page Size on Communication Performance. 2005.