Benefits of Dedicating Resource Sharing Services in Data-Centers for Emerging Multi-Core Systems

K. VAIDYANATHAN, P. LAI, S. NARRAVULA AND D. K. PANDA

Technical Report
Ohio State University (OSU-CISRC-8/07-TR53)

Benefits of Dedicating Resource Sharing Services in Data-Centers for Emerging Multi-Core Systems*†

K. Vaidyanathan P. Lai S. Narravula D. K. Panda

Dept. of Computer Science and Engineering
The Ohio State University
{vaidyana, laipi, narravul, panda}@cse.ohio-state.edu

Abstract

Current data-centers lack in efficient support for intelligent services, such as data sharing, resource monitoring, controlling overload scenarios, etc., which has become a common requirement today. Researchers and several organizations have been working towards adding co-processing units (such as FPGAs) in the hardware to offload some of the features of such intelligent services. However, at this point of time, it is not clear on how much benefits such co-processing units can provide to the data-center applications. On the other hand, as multi-core systems are emerging, it opens up new ways to design, implement and emulate such co-processing units with the help of multiple cores in the system. To study the performance benefits, in this paper, we *onload* some of the prime functionalities of data-center services to dedicated cores. Specifically, we *onload* the *data sharing* and *resource monitoring* services to a dedicated core in a multi-core system and analyze the performance benefits in terms of Inter Process Communication (IPC) costs, overhead in response time and execution time of several applications. Our micro-benchmark results show that the performance of the dedicated *data sharing* service can be improved by 25% as compared to existing implementation. Evaluations with dedicated *resource monitoring* service show that the dedicated core can avoid fluctuations in response time compared to existing implementation. Distributed STORM and application checkpointing over the *data sharing* service show up to 15% and 33% improvement, respectively, as compared to the existing implementation.

1 Introduction

Over the years, there has been a tremendous growth of distributed applications in the fields of e-commerce, bio-informatics, genomics, etc., which generate multi-terabytes of data. With technology trends, the ability to store and share these datasets is also increasing, allowing scientists and organizations to create such large dataset repositories and making them available for use by others, typically through a web-based interface forming web-based data-centers [11]. The various

*This research is supported in part by DOE grants #DE-FC02-06ER25749 and #DE-FC02-06ER25755; NSF grants #CNS-0403342 and #CNS-0509452; grants from Intel, Mellanox, Cisco systems, Linux Networx and Sun Microsystems; and equipment donations from Intel, Mellanox, AMD, Apple, Appro, Dell, Microway, PathScale, IBM, SilverStorm and Sun Microsystems.

†We would like to thank Sivaramakrishnan Narayanan for providing us with the several significant details about the Distributed STORM application and helping us tremendously in our evaluations.

nodes in a traditional web-based data-center are logically partitioned to provide several services including web and messaging services (through proxy and web servers), transaction and query processing services (through application servers), databases, etc. Such data-centers are not only becoming extremely common today, but are also increasing exponentially in size, currently ranging to several thousands of nodes. With increasing interest in web-based data-centers, more and more datasets are being hosted online. Several clients request for either the raw or some kind of processed data simultaneously. However, current data-centers are becoming increasingly incapable of meeting such sky-rocketing processing demands with high-performance and in a flexible and scalable manner.

On the other hand, the System Area Network (SAN) technology is making rapid advances during the recent years. SAN interconnects such as InfiniBand (IBA) [3] and 10-Gigabit Ethernet (10GigE) have been introduced and are currently gaining momentum for designing high-end data-centers. Besides high performance, these modern interconnects are providing a range of novel features and their support in hardware, e.g., Remote Direct Memory Access (RDMA), Remote Atomic Operations and several others. Accordingly, researchers in the past have utilized these advanced features and proposed several novel solutions for intelligent services such as resource monitoring, data sharing, locking services, controlling overload scenarios, etc. However, with increasing demands from clients, researchers and industry organizations are also working towards designing a Field Programmable Gate Array (FPGA) based co-processing unit (and its associated software) to offload some of the prime functionality of previously proposed feature enhancements for data-center applications/services. This co-processing unit, as shown in Figure 1(a), is intended to interact with both the host system (running existing data-center applications) as well as the existing network interconnects (e.g., IBA and 10GigE), to not only utilize their capabilities, but also to supplement them with more advanced application relevant features that are not available today. However, at this point of time, it is not clear on how much benefits such co-processing units can provide to data-center applications/services.

Recently, multi-core architectures have been gaining popularity due to their low-cost per processing element and are getting deployed in several data-center environments. Currently dual-core and quad-core architectures are widely available from various industry leaders including Intel, AMD, Sun and IBM. While the main idea of these architectures is to provide multiple processing elements to function in parallel, it also opens up new ways to design, implement and to emulate the functionality of co-processing units, as shown in Figure 1(b), to enhance data-center system software using the multiple cores in the system.

In our previous work, we proposed two data-center services namely the *distributed data sharing substrate* and the *fine-grained resource monitoring service*. The *distributed data sharing substrate* helps to efficiently share the resource information across the different tiers in a data-center environment. The *fine-grained resource monitoring service* helps in accurately capturing the resource information on the back-end nodes in a data-center. In this paper, we aim to leverage these two data-center services to a dedicate core of a multi-core system and study the associated performance benefits in terms of Inter Process Communication (IPC) costs, overhead in response time and execution time of several applications. Our micro-benchmark results show that the performance of the dedicated *data sharing* service can be improved by 25% as compared to existing implementation. Evaluations with dedicated *resource monitoring* service show that the dedicated core can avoid fluctuations in response time compared to existing implementation. Distributed STORM and application checkpointing over the *data sharing* service show up to 15% and 33% improvement,

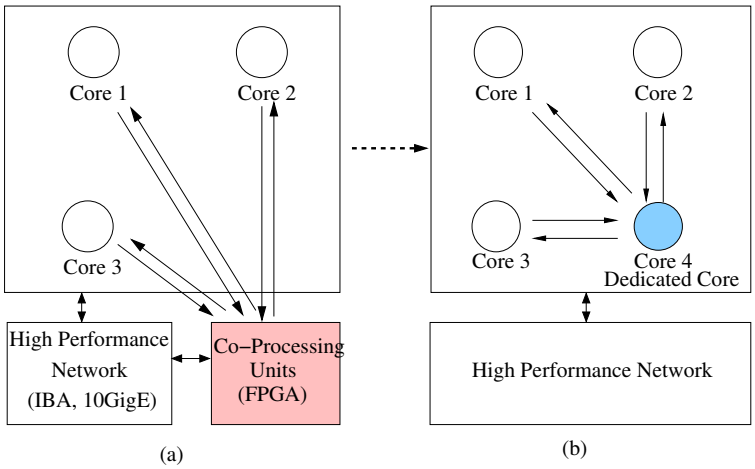


Figure 1: Data-Center Functional Offload Emulation

respectively, compared to the existing implementation.

The remaining part of the paper is organized as follows: Section 2 provides a background on previously proposed solutions for distributed data sharing substrate and fine-grained resource monitoring. In Section 3, we describe the mechanisms by which we dedicate a core to perform data-center related tasks. Section 4 deals with the evaluation of a number of micro-benchmarks and resource sharing services to show the improvements achievable by dedicated core. In Section 5, we discuss some of the issues related to dedicated a single core for data-center related components and we conclude the paper in Section 6.

2 Background

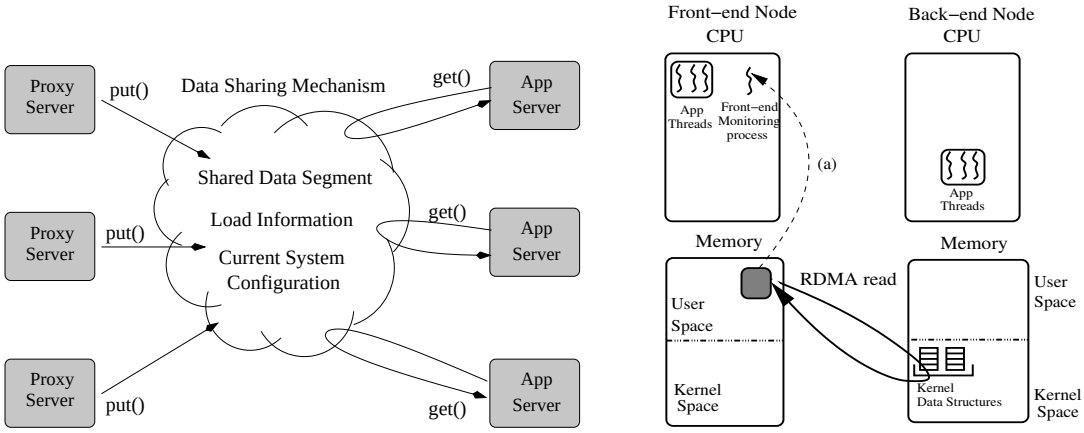
In this section, we provide a brief background of the two data-center services namely the *distributed data sharing substrate* and the *fine-grained resource monitoring* service.

2.1 Distributed Data Sharing Substrate (DDSS)

The basic idea of DDSS [15] is to allow efficient sharing of information across the cluster by creating a logical shared memory region. It supports two basic operations, *get* operation to read the shared data segment and *put* operation to write onto the shared data segment. Figure 2(a) shows a simple distributed data sharing scenario with several processes (proxy servers) writing and several application servers reading certain information from the shared environment simultaneously. DDSS also supports several other features such as data placement techniques (local or remote), locking services, coherence models such as *strict coherence*, *write coherence*, *read coherence*, *null coherence*, *delta coherence*, *temporal coherence* and also supports versioning of cached data and ensures that requests from multiple sites view the data in a consistent manner. More details are discussed in [15].

2.2 Fine-grained Resource Monitoring (RDMA-Sync)

Efficiently identifying the amount of resources used in data-center environments has been a critical research issue in the past several years. Traditionally, several techniques periodically monitor the resources used in the cluster and use this information to make various decisions such as load-balancing, reconfiguration, etc. However, these techniques relied on coarse-grained monitoring in order to avoid the overheads associated with fine-grained resource monitoring. On the other hand,



(a) Distributed Data Sharing Substrate

(b) Fine-Grained Resource Monitoring

Figure 2: Data-Center Services

the resource usage of requests is becoming increasingly divergent [6], thus increasing the need for fine-grained monitoring resources.

Fine-grained resource monitoring approach attempts to achieve three broad goals: (i) to get an accurate picture of the current resource usage in data-centers at very high granularity (in the order of milliseconds), (ii) to avoid the overhead of an extra monitoring processes on the node that is being monitored and (iii) to be resilient to loaded conditions in a data-center environment. This approach uses RDMA operations in kernel space to actively monitor the resource usage of the nodes. As shown in Figure 2(b), the necessary kernel data structures that directly monitor the resource usage are locked in memory and RDMA read operation is performed from the front-end node to capture the current load information. In this approach, the monitored information received in the front-end node matches closely with the load on the back-end. Hence, we call this approach as RDMA-Sync. Primarily, this design leads to two major advantages: (i) it removes the need for an extra process in the back-end server and (ii) it can exploit the detailed resource usage information in kernel space to report accurate load information. More details are discussed in [14].

3 Design Scope and Solution Space

In this section, we present the design scope and our solutions for dedicating services in a multi-core system.

3.1 Dedicated Core for Data-Center Services

As mentioned in Section 1, several industry organizations and researchers have been focusing on *offloading* feature enhancements to co-processing units such as FPGAs. Unfortunately, the benefits of *offloading* data-center services to such co-processing units is not clear due to the lack of such hardware. In order to understand the potential benefits, we propose to use a core in a multi-core system as a co-processing unit and try to dedicate several data-center services and show the associated performance benefits.

Figure 3 shows the overall architecture of data-centers for emerging multi-core architectures. In

Figure 3(a), we show the existing architecture that current data-centers use without any intelligent services. In Figure 3(b), we add an additional core (*core 4*) and dedicate the core for *onloading* several data-center services. All other cores (*core 1, 2 and 3*) communicate to the dedicated core for performing certain *onloaded* tasks. Figure 3(c) shows one possible way to communicate to the dedicated core using IPC mechanisms. As an example, if *core 2* needs to communicate to the dedicated core, it initiates an *IPC Send* message to the dedicated core. The dedicated core, upon receiving the request from *core 2*, does the processing and responds through an *IPC Recv* message. Figure 3(d) shows another mechanism for communicating directly to the dedicated core using shared memory region. A similar mechanism of communicating using memory mapped buffers has been proposed by [1]. *Core 2* produces a request to the request queue and the dedicated core consumes the request. After processing the request, the dedicated core replies to *core 2* by producing a response message in the response queue which gets consumed later by *core 2*. Since this operation requires the *core 2* to constantly poll for response messages in the response queue, the dedicated core can also optionally send an interrupt to *core 2* after processing the request. It is to be noted that both the mechanisms (as shown in Figure 3(c) and Figure 3(d)) are common today and in our evaluation we have compare against these two mechanisms and show the performance benefits.

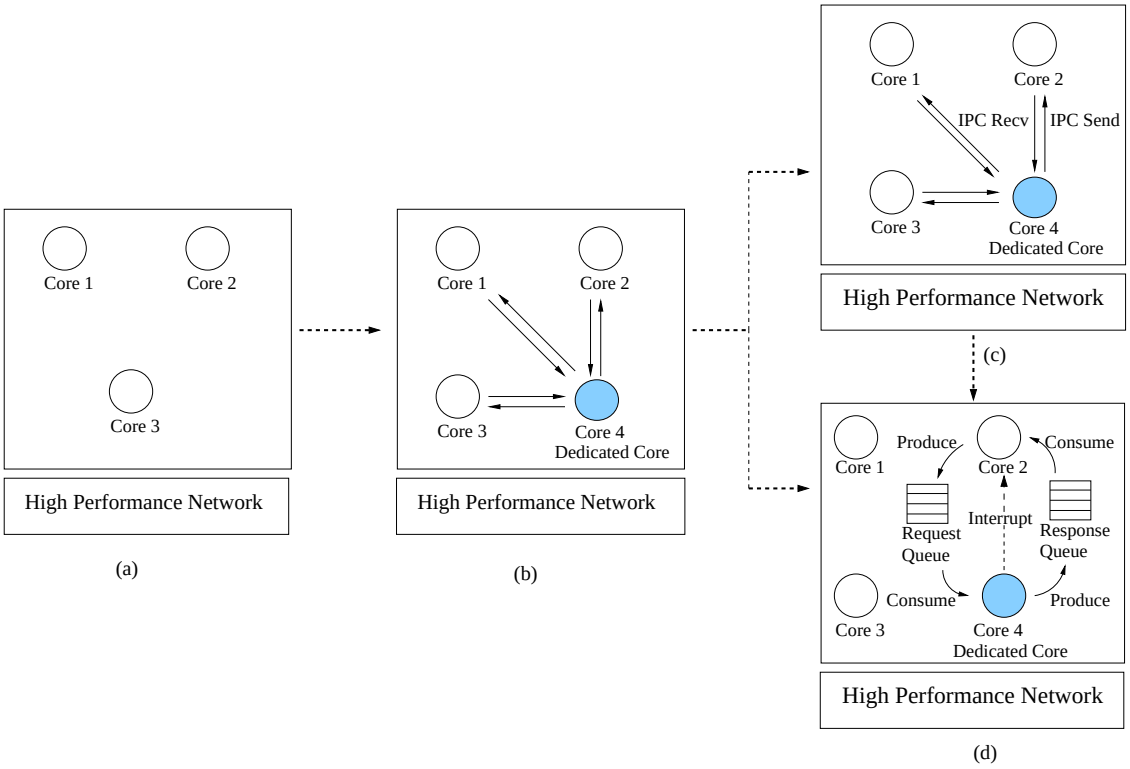


Figure 3: Data-Center Services for Emerging Multi-Core Architectures

3.1.1 Handling IPC Synchronization

As shown in Figure 3(c), *core 2* communicates with *core 4* using kernel-based IPC system calls. Kernel-based IPC system calls involve the operating system to trap the current running process and send an interrupt to the process that is waiting for receiving the IPC request. This creates a huge amount of delay in communicating with the dedicated core. As shown in Figure 3(d), we use the

shared memory region for communicating with the dedicated core (*core 4*). Application threads running on all other cores use the shared memory region to communicate to the dedicated core, as mentioned earlier. Through this mechanism, the operating system interaction is completely removed resulting in saving a lot of context switches and interrupt handling.

3.1.2 Handling Communication Progress

Apart from avoiding kernel-based IPC synchronization costs, dedicated cores also help in improving the progress of several of the *onloaded tasks* that are dedicated to the core. Due to the fact that the dedicated core constantly waits for incoming requests or checks for completion of *onloaded tasks*, it automatically helps in making faster progress of tasks and can ultimately help improve the application performance.

3.1.3 Handling Polling Vs Blocking Operations

Some of the applications wait for the dedicated core to complete the requested task and proceed while other applications may want to receive an interrupt when the requested task has been completed. Especially, for data-center applications, several data-center threads need not necessarily wait for the requested task to finish to proceed to the next task. Moreover, if several of these threads constantly poll on the response queue to check for completion, the performance of the data-center can degrade significantly. For this purpose, as shown in Figure 3(d), the dedicated core optionally can also send interrupts to the requested process upon completion of the requested task. This method not only provides a lot of scope for overlapping operations but also helps in avoiding wastage of CPU cycles and thus avoid any degradation in the data-center performance.

3.2 Dedicating Distributed Data Sharing Substrate Services

In this section, we present our designs in dedicating the *distributed data sharing substrate* service in a multi-core system.

3.2.1 Basic Architecture

The basic architecture for dedicating this service is very similar to the designs mentioned in the previous section. Due to the fact that several data-center services/applications share several common information, we *onload* the components of the *distributed data sharing substrate* as mentioned in Section 2 and use the shared memory region to communicate to the dedicated core. In this architecture, we *onload* tasks such as posting and receiving IBA-related tasks, book-keeping tasks such as *allocation* and *release* operations of the DDSS, locking services, etc.

3.2.2 Benefits of Dedicating Data Sharing Substrate

Dedicating the *distributed data sharing substrate* to a core can offer several benefits. Due to the fact that we avoid kernel-based IPC synchronization costs, multiple data-center application/service threads can observe improvement in *get()* and *put()* operations of the DDSS. Most importantly, several of the management related tasks such as *allocation* and *release* operations of the DDSS may show huge improvement in overall response time. Such designs also open up new scope for dedicating complex operations of the *distributed data sharing substrate* service.

3.3 Dedicating Resource Monitoring Services

As mentioned in Section 2, efficiently identifying the resource information of back-end nodes is an important aspect for managing the physical resources of the data-center. In [14], we proposed an efficient mechanism for fine-grained resource monitoring. Though this mechanism completely removed the overhead on the back-end node, it introduces a lot of overhead on the front-end to constantly perform network operations in getting the resource information. In this section, we present our designs for dedicating this resource monitoring service.

3.3.1 Basic Architecture

In this approach, as shown in Figure 4, we compare against the existing mechanisms for efficient resource monitoring service. Figure 4(a) shows a typical data-center environment where data-center applications execute without any data-center service on a multi-core system. We refer to this approach as Basic approach. In Figure 4(b), we include the data-center services such as resource monitoring which runs in parallel with the data-center application threads in a multi-core environment. It is easy to observe that the data-center application threads, apart from competing with other application threads for CPU cycles, it also needs to compete with the resource monitoring service for CPU cycles. In the experimental section, we use this setup and evaluate the RDMA-Sync approach. In Figure 4(c), we introduce an additional dedicated core and move the data-center service threads (e.g., resource monitoring) to the dedicate core. The dedicated core performs all other tasks such as communication progress, waiting for responding to user requests, etc. In the experimental section, we use this setup and evaluate the dedicated resource monitoring service. We refer to this approach as DC-RDMA-Sync approach.

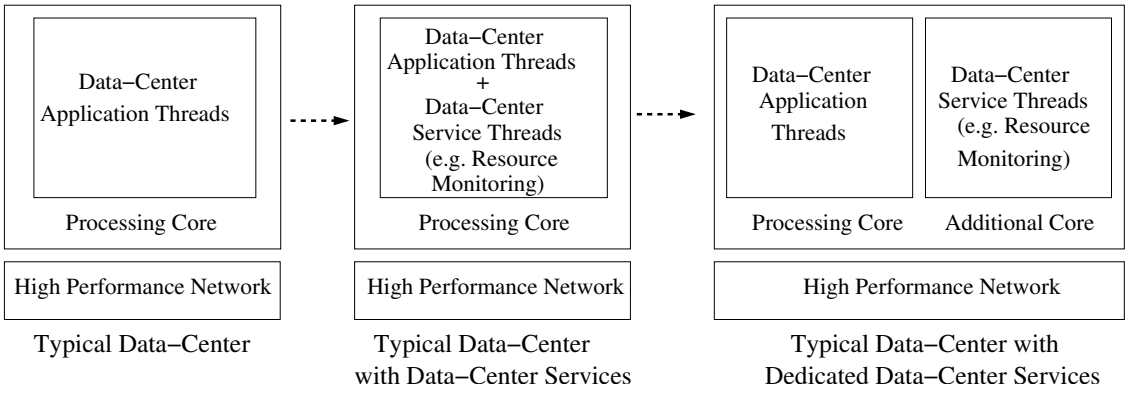


Figure 4: Dedicating Cores for Data-Center Services

3.3.2 Benefits of Dedicating Resource Monitoring Services

There are several benefits achievable in *onloading* the resource monitoring service to a dedicated core. For fine-grained resource monitoring service, it is important to deliver the resource information to appropriate processes in a timely manner. Dedicating such a service to the core results in avoiding kernel-based IPC calls for synchronization purpose and helps in delivering the resource information to the resource manager in a timely manner. More importantly, the performance of several application threads which run in parallel with the fine-grained resource monitoring gets unaffected. With this approach, we not only avoid all processing overheads in the back-end nodes

(using RDMA operations) but also avoid overheads of posting the network descriptors, processing the resource information, etc., in the front-end node.

4 Experimental Results

In this section, we present various performance results. First, we present the ideal case performance improvement achievable by the *distributed data sharing substrate* and then show the performance improvement achieved using applications such as distributed STORM and application checkpointing. Next, we show the performance improvement achieved in dedicating the *fine-grained resource monitoring* service.

For all our experiments we used two clusters whose descriptions are as follows:

Cluster1: A cluster system consisting of 4 nodes and each node has dual quad-core Intel Xeon 2.3 GHz processors with a 4 MB L2 cache and 4 GB of main memory. We used the RedHat 9.0 Linux distribution and Linux 2.6.9-42.ELsmp kernel.

Cluster2: A cluster system consisting of 48 nodes and each node has two Intel Xeon 2.6 GHz processors with a 512 KB L2 cache and 2 GB of main memory. We used the RedHat 9.0 Linux distribution and Linux 2.6.9-34.ELsmp kernel.

Cluster 1 was used to evaluate the *distributed data sharing substrate* service and Cluster 2 was used to evaluate the *fine-grained resource monitoring* service. We used apache version 2.2.4 in all our data-center experiments.

4.1 Dedicating Distributed Data Sharing Substrate

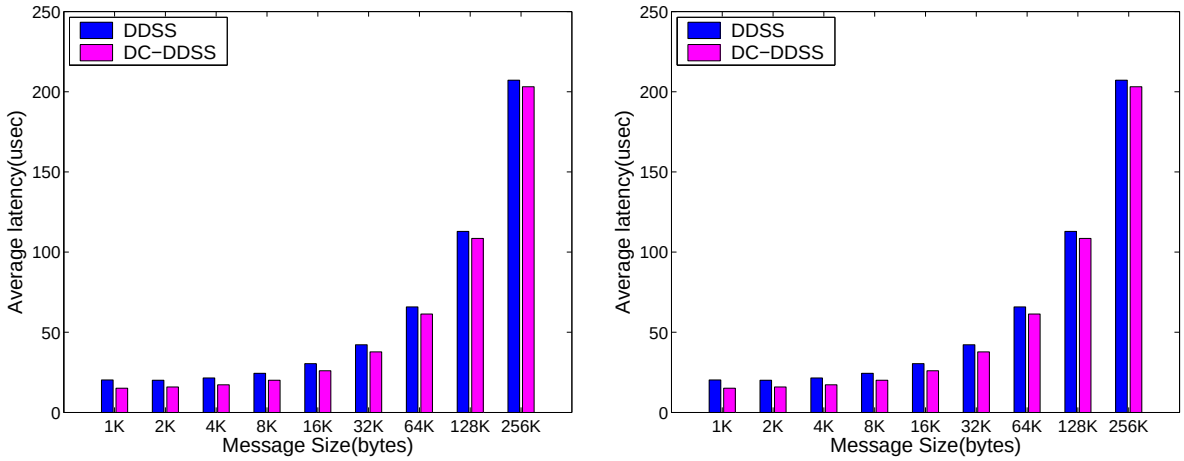
In this section, we present the benefits of dedicating *distributed data sharing substrate* service. We refer to the dedicating *distributed data sharing substrate* as DC-DDSS approach and compare it against the existing implementation (DDSS approach) proposed in [15].

4.2 Micro-benchmarks

Here, we present the ideal case performance benefits achievable by dedicating the *distributed data sharing substrate*, as shown in figure 5, especially the *get()* and *put()* operations of DDSS. As shown in figure 5(a), we see that the performance of *put()* operation using DDSS is a little worse compared to the performance of *put()* operation using DC-DDSS. We see that DC-DDSS can improve the latency of *put()* operation by 25% as compared to DDSS. However, for increasing message sizes, we observe that the improvement decreases since kernel based IPC operations no longer remain the bottleneck. The improvement seen for small message sizes is mainly due to avoiding kernel based IPC operations. We see similar trends for *get()* operation as shown in figure 5(b).

4.3 Applications with DDSS

Here, we present the application numbers using DDSS and DC-DDSS approaches, as shown in figure 6. STORM [10, 2] is a middle-ware service layer developed by the Department of Biomedical Informatics at The Ohio State University. It is designed to support SQL-like select queries on datasets primarily to select the data of interest and transfer the data from storage nodes to compute nodes for processing in a cluster computing environment. It is implemented using DataCutter [4] which is designed to enable exploration and analysis of scientific datasets. We had already shown that DDSS can improve the performance of STORM in [15] and hence we only show the performance improvement compared to DDSS. Figure 6(a) shows the performance of STORM over DDSS and DC-DDSS. As shown in the figure, we observe that the performance of STORM, using



(a) Performance of *put()* Operation

(b) Performance of *get()* Operations

Figure 5: Impact of Dedicating DDSS

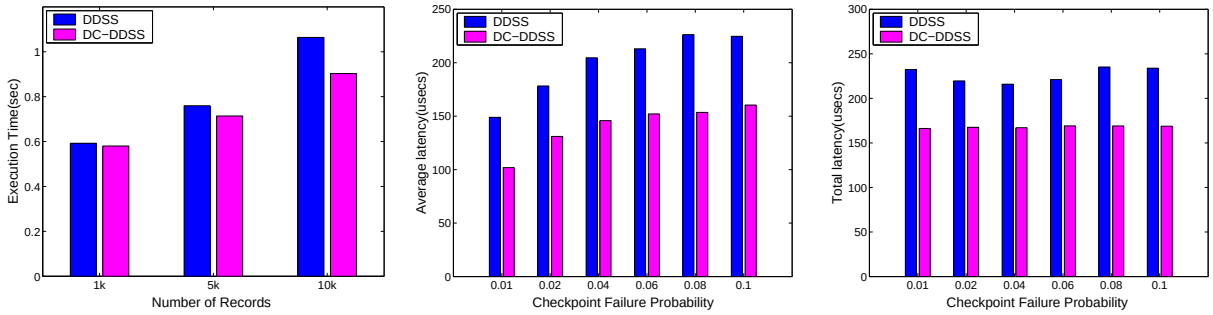
the DC-DDSS approach, is improved by around 6% and 15% for 5K and 10K record datasets, respectively, in comparison with DDSS approach.

We also evaluated an application check-pointing benchmark, as mentioned in [15] to show the performance of DDSS and DC-DDSS. In this experiment, every process attempts to checkpoint a particular application at random time intervals. Also, every process simulates the application restart, by attempting to take a consistent check-point and informing all other processes to revert back to the consistent check-point at other random intervals based on a failure probability. In this experiment, we vary the failure probability and show the performance of checkpointing over DDSS and DC-DDSS. In figure 6(b), we observe that the the average synchronization time taken by checkpointing using DC-DDSS is around 33% lesser in comparison to the time taken by DDSS. The main reason for this improvement is due to the kernel based IPC cost savings and since the benchmark has several small size operations with the *distributed data sharing substrate*, kernel based IPC costs get saved multiple times while accessing the substrate. We see similar trends in the total time observed in checkpointing as shown in figure 6(c).

4.4 Dedicating Fine-grained Resource Monitoring Services

As mentioned in Section 3.3, we dedicate the resource monitoring service to a dedicated core (DC-RDMA-Sync approach) and compare its performance against the other two approaches (RDMA-Sync and Basic approaches). In the Basic approach, we do not perform any resource monitoring. In RDMA-Sync approach, we perform resource monitoring which executes along with other data-center threads in the system. In order to identify the overheads caused as a result of resource monitoring, we use only one client to generate the requests and measure the response time achieved by the client.

We vary the experiments in terms of different file size requests, different number of server being monitored and different monitoring granularities and report the client response time measured across several iterations. We also measure the client response time without any resource monitoring service and use this as a baseline for comparisons. In all our experiments, we have one client that



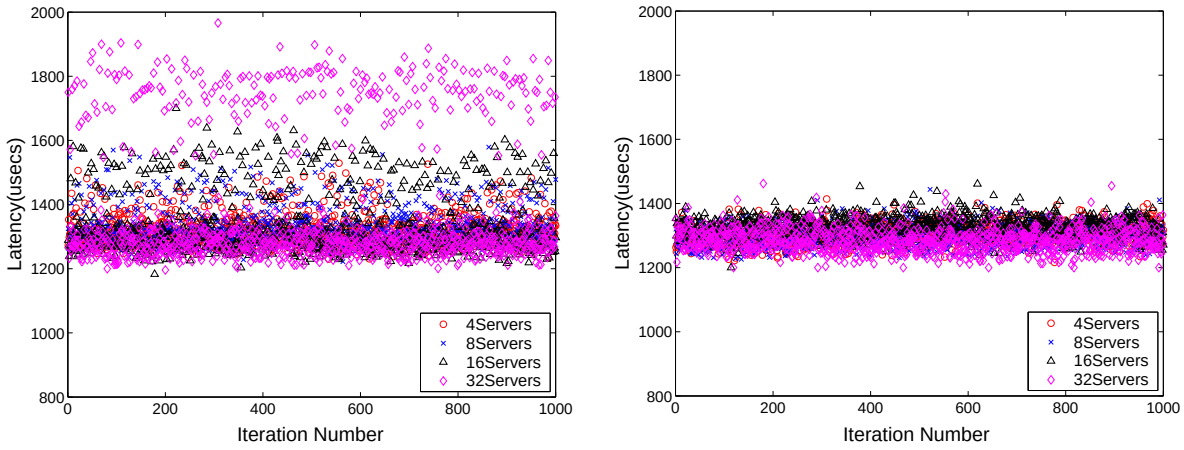
(a) Distributed STORM

(b) Checkpointing: Impact on Average Synchronization Time

(c) Checkpointing: Impact on Total Time

Figure 6: Applications with DDSS

generates the request, one proxy server which executes several apache threads and the resource monitoring service thread and several web servers in the back-end.



(a) Performance of RDMA-Sync

(b) Performance of DC-RDMA-Sync

Figure 7: Impact on Number of Servers being Monitored

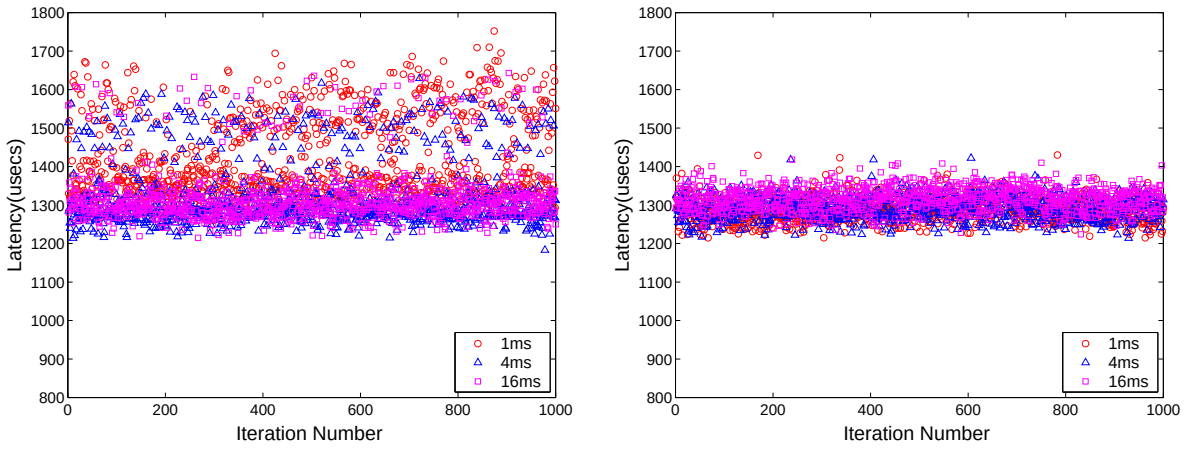
4.4.1 Impact on Number of Servers

In this experiment, we measured the response time for RDMA-Sync approach and DC-RDMA-Sync approach by varying the number of web servers (up to 32 servers) that are being monitored. Figure 7(a) shows the response time of file size 16 KB with resource monitoring granularity of 4ms for varying number of servers using the RDMA-Sync approach. As shown in the figure, we observe that the client response time fluctuates frequently across several iterations. The fluctuation is heavily contributed by the resource monitoring thread that is competing along with the data-center thread for performing resource monitoring service. This results in increasing the client response time. Further, we observe that the response time increases with increasing number of server being

monitored. This is expected, as monitoring more number of servers would mean that the resource monitoring service thread perform many network operations to capture the load information on all the servers, thus increasing the time that the resource monitoring thread competes for CPU with other application threads. We perform the same experiment in Figure 7(b), however, we dedicate the resource monitoring service to a dedicated core. We see that the response time of the client is almost unaffected throughout the experiment using the DC-RDMA-Sync approach.

4.4.2 Impact on Monitoring Granularity

In this experiment, we study the impact of different monitoring granularity for RDMA-Sync and DC-RDMA-Sync approaches. We use 16 KB file size and 16 web servers to be monitored by the resource monitoring thread and vary the monitoring granularity from 1 ms to 16 ms. As shown in figure 8(a), we see that the response time of client requests fluctuate from 1200 μs to 1700 μs using the RDMA-Sync approach. In addition, we also observe that the response time fluctuates more frequently as the granularity interval decreases to 1 ms. The frequency of monitoring granularity determines the number of times the response time gets affected. However, as shown in figure 8(b), we observe that the response time is unaffected by the monitoring granularity using the DC-RDMA-Sync approach. Due to the fact that the dedicated core performs the resource monitoring service, we observe very little change in the client response time. We saw similar trends for other monitoring granularities.



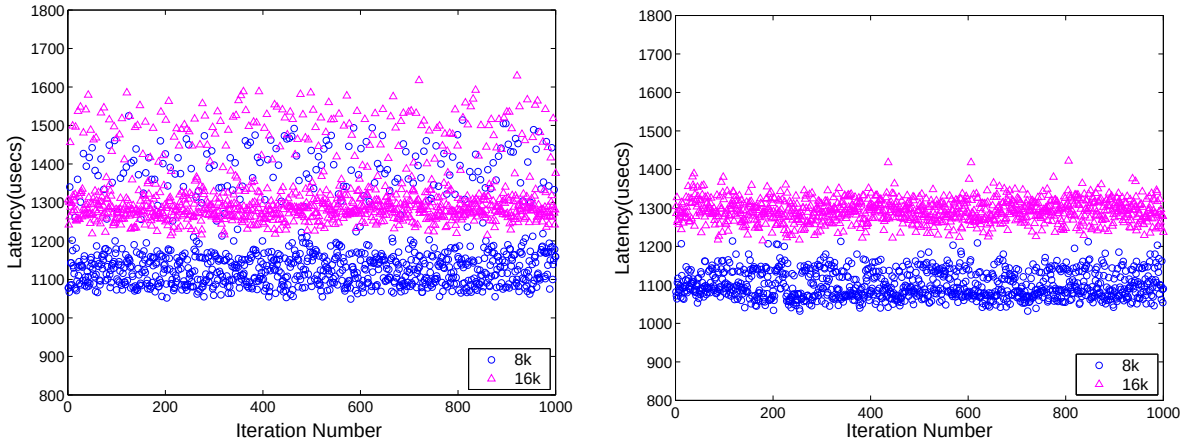
(a) Performance of RDMA-Sync

(b) Performance of DC-RDMA-Sync

Figure 8: Impact on Monitoring Granularity

4.4.3 Impact on Requested File Size

Here, we study the impact on the requested file sizes and fix the monitoring granularity to 4 ms and the number of servers monitored to 16 servers and report the response time in figure 9(a). We see similar trends as observed in the previous experiment. As shown in figure 9(a), we observe that the response time of file size 8 KB and 16 KB fluctuates using the RDMA-Sync approach. However, we see very little fluctuation with the response time using the DC-RDMA-Sync approach, as shown in figure 9(b).



(a) Performance of RDMA-Sync

(b) Performance of DC-RDMA-Sync

Figure 9: Impact on Different File Sizes

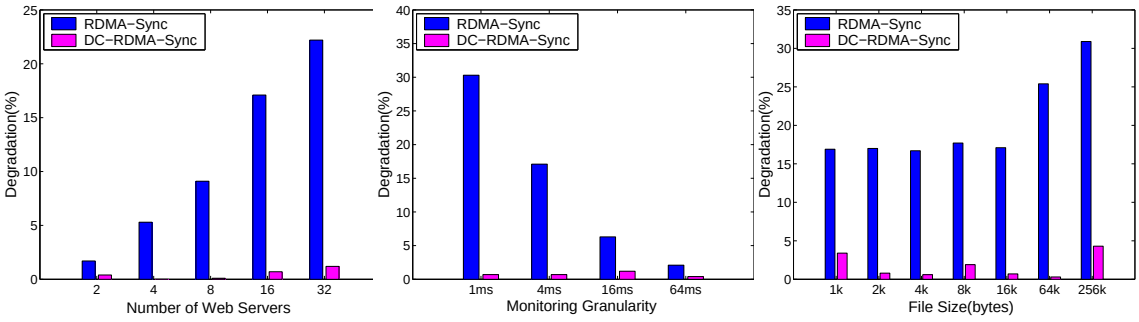
4.4.4 Benefits of Dedicating Resource Monitoring Service

In this section, we analyze the benefits of dedicating resource monitoring service by comparing against the Basic approach which does not perform any resource monitoring service and show that the response time does not increase with dedicated resource monitoring service. We introduce a performance degradation metric to explain our experimental results. For each experiment, we measure the average response time (T_{avg}) and define a α value which determines the fluctuating threshold. We fix this α value to 10% and pick all the candidate response times which are greater than $(\alpha + 1) * T_{avg}$. We define the performance degradation metric as the ratio of number of candidate response times which are greater than $(\alpha + 1) * T_{avg}$ to the total number of requests performed during the experiment. Larger values of this metric indicates a large fluctuation in response time and smaller values indicate the response time during the experiment does not get affected.

We report the performance degradation metric in figure 10. For varying number of servers monitored, as shown in figure 10(a), we see the performance degradation metric using RDMA-Sync approach as compared to the Basic approach is up to 22% whereas the performance degradation metric using DC-RDMA-Sync approach remains less than 2%. This indicates that DC-RDMA-Sync approach very rarely affects the response time. For varying monitoring granularity, as shown in figure 10(b), we observe that the degradation metric using RDMA-Sync approach as compared to the Basic approach is up to 30%. However, for DC-RDMA-Sync approach, the degradation metric is less than 2%. For varying file sizes, as shown in figure 10(c), we see that the degradation metric is as high as 30% using the RDMA-Sync approach. However, DC-RDMA-Sync approach shows only 4% degradation. The results in figure 10 further strengthens the claim for offloading data-center services to co-processing units such as FPGAs.

5 Discussion and Related Work

Auxiliary co-processing units have become an attractive option for specialized jobs in High End Computing (HEC) systems. FPGA's and other additional processing units have started making



(a) Impact on Number of Servers being Monitored

(b) Impact on Monitoring Granularity

(c) Impact on Requested File Size

Figure 10: Data-Center Performance

inroads into different components of HEC systems like programmable NIC's, acceleration boards, etc. While FPGA's can perform certain kind of operations at wire speed, the kind of processing that can be performed efficiently with FPGA's is limited. In addition, these additional processors lack the capability and performance of normal host processors. On the other hand, these additional units usually being disjoint from the host OS can in some cases continue to operate after certain host OS failures, possibly aiding the monitoring of failures and subsequent recovery.

With the modern processors providing increasing number of cores, the applications now need to be multi-threaded to leverage the additional processing capability offered by these processors. This leads into the possibility of dedicating the cores to perform specialized operations. While this approach can yield effective results in cases where the total number of cores in the system are large and the system can afford to dedicate a core for a specialized purpose, these advantages can easily be lost in scenarios where systems have very few cores and dedicating a core could impact the overall application performance.

Modern processors are seeing a steep increase in the number of cores available in each packaged die [7]. Researchers [8, 9, 16, 13] are actively looking at studying and improving performance of HEC systems in the context of these multi-core systems.

Some modern interconnects [5] provide programmable processors on the NIC that can be used for certain scenarios. Clear Speed compute boards [12] provide additional computing power for specialized compute codes. While utilizing dedicated processors as Graphics Processing Units GPU's has become very common, the benefits in HEC systems need to be studied.

6 Conclusions

Researchers and several organizations have been working towards adding co-processing units (such as FPGAs) in the hardware to offload some of the features of data-center services. To understand the benefits of such hardware components, in this paper, we used the multi-core architecture to emulate the behavior of FPGAs, *onloaded* data-center services and studied the associated performance benefits. Specifically, we dedicated a core of a multi-core system and studied the benefits of *data sharing* and *resource monitoring* service in terms of Inter Process Communication (IPC) costs, overhead in response time and execution time of several applications. Our micro-benchmark results showed that the performance of the dedicated *data sharing* service can be improved by 25%

as compared to existing implementation. Evaluations with dedicated *resource monitoring* service show that the dedicated core can avoid fluctuations in response time compared to existing implementation. Distributed STORM and application checkpointing over the *data sharing* service show up to 15% and 33% improvement, respectively, compared to the existing implementation. Our experimental results strengthen the claim for offloading data-center services to co-processing units such as FPGAs.

References

[1] Hp research labs. http://www.hpl.hp.com.

[2] The STORM Project at OSU BMI. http://storm.bmi.ohio-state.edu/index.php.

[3] Infiniband Trade Association. http://www.infinibandta.org.

[4] M. D. Beynon, T. Kurc, U. Catalyurek, C. Chang, A. Sussman, and J. Saltz. Distributed Processing of Very Large Datasets with DataCutter. *Parallel Computing*, October 2001.

[5] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W. K. Su. Myrinet: A Gigabit-per-Second Local Area Network. http://www.myricom.com.

[6] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin Vahdat, and Ronald P. Doyle. Managing energy and server resources in hosting centres. In *Symposium on Operating Systems Principles*, 2001.

[7] Intel Corporation. http://www.vnunet.com/vnunet/news/2165072/intel-unveils-tera-scale, Sep 2006.

[8] Max Domeika and Lerie Kane. Optimization Techniques for Intel Multi-Core Processors. http://www3.intel.com/cd/ids/developer/asmo-na/eng/261221.htm?page=1.

[9] Kittur Ganesh. Optimization Techniques for Optimizing Application Performance on Multi-Core Processors. http://tree.celinuxforum.org/CelfPubWiki/ELC2006Present ations?action=AttachFile&do=get&target=Ganesh-CELF.pdf.

[10] S. Narayanan, T. Kurc, U. Catalyurek, and J. Saltz. Database Support for Data-driven Scientific Applications in the Grid. In *Parallel Processing Letters*, 2003.

[11] H. V. Shah, D. B. Minturn, A. Foong, G. L. McAlpine, R. S. Madukkarumukumana, and G. J. Regnier. CSP: A Novel System Architecture for Scalable Internet and Communication Services. In *the Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems*, pages pages 61–72, San Francisco, CA, March 2001.

[12] ClearSpeed Technologies. http://www.clearspeed.com.

[13] Vernon Turner. The Implications of Multi-core Processors for Software Companies and Corporate IT. www.intel.com/software/insight/IDC/308499-001_intelTurner4.pdf.

[14] K. Vaidyanathan, H. W. Jin, and D. K. Panda. Exploiting RDMA operations for Providing Efficient Fine-Grained Resource Monitoring in Cluster-based Servers. In *Workshop on Remote Direct Memory Access (RDMA): Applications, Implementations, and Technologies (RAIT 2006), in conjunction with Cluster Computing September*, 2006.

[15] K. Vaidyanathan, S. Narravula, and D. K. Panda. DDSS: A Low-Overhead Distributed Data Sharing Substrate for Cluster-Based Data-Centers over Modern Interconnects. In *International Conference on High Performance Computing (HiPC)*, 2006.

[16] Ashlee Vance. What 2007 means to your data center. http://www.theregister.co.uk/2004/06/12/smp_revival_partone/.