UNIVERSITY OF CALIFORNIA, SAN DIEGO

A case for effective utilization of Direct Cache Access for big data workloads

A thesis submitted in partial satisfaction of the
requirements for the degree
Master of Science

in

Computer Science (Computer Engineering)

by

Harsha Basavaraj

Committee in charge:

Dean Tullsen, Chair
George Porter
Leonard Porter

2017

The thesis of Harsha Basavaraj is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

Chair

University of California, San Diego

2017

DEDICATION

Dedicated to my parents and grandparents

EPIGRAPH

Not everything that can be counted counts,
and not everything that counts can be counted.

– Albert Einstein

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to the following people for the help, support and encouragement they provided me during the course of my graduate studies.

Professor Dean Tullsen, for all the guidance and support throughout my graduate studies as my advisor. He has been a great source of inspiration for continued excellence. The opportunity to be part of his research group will be an invaluable and cherished experience.

Dr Sheng Li, for his continuous guidance and crucial support. He was instrumental in guiding the overall direction of this work and without his support this thesis would not have been possible.

Professor George Porter and Professor Leo Porter, for taking out their valuable time to review this work and providing insightful and constructive feedback to improve this work.

Ashish Venkat, for support as a great mentor and Nishant Bhaskar, for all the pep talks.

Professors, staff, lab-mates, friends and peers at UCSD for making UCSD feel like home and making my stay at UCSD a cheerful and memorable one.

I wouldn't have gained all successes so far in life without the silent prayers and support from my family. I am grateful for their love and faith in me, and owe all my accomplishments to them.

VITA

2009-2013	Bachelor of Engineering (B.E) Telecommunication Engineering M S Ramaiah Institute of Technology, Bengaluru (Visveswaraiah Technological University, Belgavi, India)
2013-2015	Software Development Engineer Cisco Systems, Bengaluru
2016-2017	Graduate Teaching Assistant University of California, San Diego
2015-2017	Master of Science (M.S) Computer Science (Computer Engineering) University of California, San Diego

ABSTRACT OF THE THESIS

A case for effective utilization of Direct Cache Access for big data workloads

by

Harsha Basavaraj

Master of Science in Computer Science (Computer Engineering)

University of California, San Diego, 2017

Dean Tullsen, Chair

The exploration of techniques to accelerate big data applications has been an active area of research. Although we have highly efficient computing cores and high-speed networks, the bottleneck in most big data applications has been the latency of data access. The foremost contributors to this latency are the network communication, storage systems, software stack and data transfer. Heterogeneous co-processors, FPGA accelerators, and flash based storage accelerators try to overcome this latency by offloading processing from the primary processor, but these cause additional overheads to an already costly data-center server and increase the total deployment cost. With an ever growing size of data, the need to exploit the available resources in the primary processor while achieving the best possible performance becomes increasingly necessary. A humble performance improvement of even 1% goes a long way in a typical

data center environment. Consequently, this work evaluates the effectiveness of Data Direct Input Output (DDIO) commonly known as Direct Cache Access (DCA) for I/O intensive big data workloads. We begin with a survey of various kinds and characteristics of big data workloads and then present the performance gain/loss due to DCA for I/O intensive workloads on Xeon E5 based servers. The big data applications are considerably different from the workloads traditionally used in architectural studies hence micro-benchmarks are used to emulate workloads which could gain/lose considerable performance when using direct cache access. Also, we present the performance of I/O intensive tasks from state of the art Cloudsuite benchmark suite. We finally make a case for the dynamic use of DCA in the processor for better performance of big data applications (change the percentage of cache available for DDIO to use or the cache levels DCA can access).

Chapter 1

Introduction

The exponential growth of data [3, 4] poses a continuous challenge to effective execution of big data applications. Big data has become pervasive in business and research; its ever increasing size has increased the complexity of methods to process, analyze, extract, store and collect it. The scope of research to accelerate such applications covers the entire stack of processing, starting from high level software optimization, to the hardware design of the processors. The effective use of data at large scale needs infrastructure which is flexible, while conforming to constraints of space, power and cooling in data centers. The processor compute infrastructure lies at the heart of a data center, hence computer architects have a crucial role to play in optimizing the data centers.

Big data workloads are characterized as data and communication intensive [5, 6, 7, 8, 9]. The prime contributors to latency in these applications are storage systems, software stack, data movement and communication [8, 9]. Data movement is a common operation in data centers [10, 11, 12] and an integral task of big data applications as well. The latency due to data movement accounts for around 30% of the total application latency of an application using Linux TCP/IP stack [11]. There are several techniques proposed and deployed to accelerate the big data workloads with most of them using additional peripherals such as GPUs, FPGAs [13, 14] and custom ASICs [15] along with the general purpose CPUs. While the trend of using peripheral devices or custom architectures is prevalent, they might come with additional costs in terms of programmability, ease of deployment and the energy footprint.

The acceleration techniques with additional hardware and software optimization have been at the forefront of research, but there has been less focus on effectively using the existing micro-architectural features of the microprocessor. The concept of Direct Cache Access [16] as introduced by Ravi, et al. overcomes latency in the I/O data path by providing the network with direct access to the processor's cache. The implementation of this feature in Intel Xeon processor architecture is known as Data Direct I/O (DDIO) [17]. DDIO allows the network interface card to directly access the processor cache, minimizing the number of main memory accesses required for network packet processing and improves the efficiency of data delivery. Big data applications that are I/O intensive and streaming applications will certainly see performance gains while accessing the cache hierarchy directly, but the gains may not be uniform across the diverse set of applications. This motivates us to evaluate the effectiveness of Direct Cache Access (DCA) for big data workloads by demonstrating the variance of its utility across different applications or different phases within the same application and explore techniques for its dynamic usage.

Analyzing big data workloads across the big data application landscape to evaluate the effectiveness of DDIO is a difficult task given the scale, size and diversity of these applications. To do so, we develop few I/O intensive micro benchmarks which are representative of the characteristics that can benefit due to DDIO. We also evaluate DDIO with data caching benchmark of Cloudsuite [18] as it is a network I/O intensive application. The same set of benchmarks are used to demonstrate the variation in performance across different kinds of applications, and different phases of the same application, with help of hardware performance counters. The performance counters track metrics such as LLC misses, instructions executed and number of cycles consumed to evaluate the behavior of the application with DDIO enabled and disabled. Finally we conclude with a case for dynamic usage of DDIO based on the utility variance.

The thesis is organized as follows: Chapter 2 builds a background on the different communication mechanisms between network, memory and compute subsystems, details of DDIO and related work on big data application acceleration. Chapter 3 presents details of the experiments and evaluation. The results of the experiments are presented in chapter 4, we conclude and present future directions in chapter 5.

Chapter 2

Background and Related work

This section gives a high level overview of network infrastructure of the data center, memory hierarchy of the processor, how data is transferred from network I/O subsystem to the compute subsystem and the Big Data workload characteristics.

2.1 Network, Memory and Computation

Network, storage, memory and compute are the fundamental building blocks of a data center [19, 20]. The communication networks in data centers have evolved rapidly over the past decade with 10GbE and 40GbE infrastructure being widely used, and devices supporting 100GbE making inroads into commercial market. The memory subsystem, while lagging in speed, has caught up with the latest commercially available DDR4 DRAM technology that can support speeds up-to 48GBps, while alternate memory technologies such as NVM, MTTRAM, etc. [21] continue to emerge. Speed of computation has been the dominant among these three fundamental subsystems, and reached abilities to process at frequencies greater than 3GHz. With the end of Dennard scaling, multi-core scaling vowing to Moore's law has maintained the proportional increase in computation capacity, but may not continue to do so [22]. Here in this chapter we recall some of the important interaction techniques between these components, discuss characteristics of big data workloads and related work in big data acceleration. The data centers built with commodity components and running commodity software incur huge latency penalties despite all the different mechanisms discussed further. In this

work we are concerned about the communication latency due to the OS network stacks and data transfers between the processor and the network interface cards (NICs).

2.1.1 MMIO and DMA

Memory mapped I/O (MMIO) is a mechanism in which the I/O devices communicate using a preassigned memory address mapped to the I/O device [23]. MMIO is a protocol where the CPU reads and writes to specific memory address but the memory address is an I/O device. The behavior of any MMIO device is similar to that of the DRAM or any other memory subsystem expect that its managed by a special I/O instruction or a control signal.

DMA is a very common technique used by network interface cards (NIC) for transferring of packets arriving from the network to memory. The data transfer mechanism uses a dedicated DMA controller bypassing the CPU. The DMA controller offloads data transfer (memory) operations from the CPU and makes data transfers cheap without needing any processor time. Figure 2.1 shows the high level view of the transactions during DMA I/O.

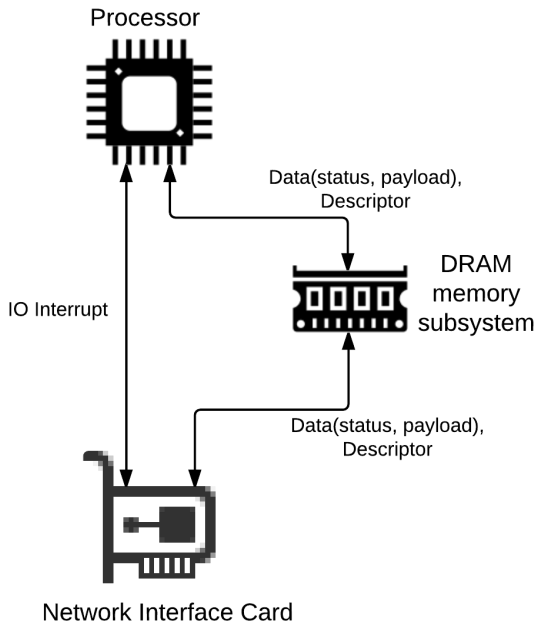

Figure 2.1: Typical Direct Memory Access I/O

2.1.2 Direct Cache Access

Direct Cache Access (DCA) is an enhancement to the system interconnect coherency protocol to move the incoming data through network I/O directly to processor cache bypassing the main memory [16]. The prime motive of DCA is to overcome the latency due to visits to main memory while processing the data obtained through network I/O. DCA provides mechanism for incoming and outgoing data through network I/O to be streamed directly from the processor's cache. The primary benefit of DCA is reduction in average latency of network packet processing and effective utilization of memory bandwidth. For a typical TCP/IP based application the number of memory accesses with DCA is just one memory access for a N cacheline payload whereas in case of DMA it requires 2N+5 [11, 12] memory accesses. The growing network speeds with 400GbE being standardized and 100GbE, 40GbE being commercially deployed makes it important to process the packets at the same speed as they are transmitted or received to achieve optimum utilization of the network speed and processor speed. Apart from offload engines, DCA can also help us achieve optimum utilization by minimizing number of visits to the main memory. Figure 2.2 shows the high level view of the transactions during DCA I/O.

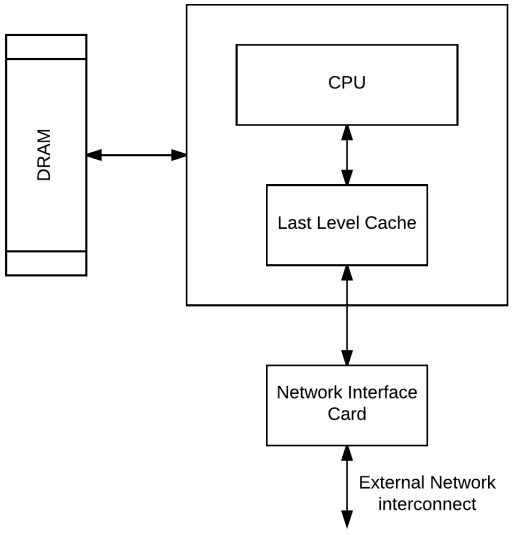

Figure 2.2: Transactions in typical Direct Cache Access I/O

2.1.3 Intel Data Direct I/O

Intel Data Direct I/O (DDIO) is the only commercially available implementation of DCA. The terms DDIO and DCA have been used interchangably in this thesis. With increasing cache sizes (>20MB), DDIO became a practical and realistic feature to gain performance. DDIO enables direct communication between last level cache of the processor and the network interface card there by reducing the number of hops to main memory and back while performing I/O operations. DDIO not only improves the latency of the processor but also improves the power consumption of the processor by reducing the number of trips to main memory. DDIO is analogous to DCA but limits the access of the NICs to the last level cache of the processor. The path of the packets from the NIC to processor is similar to that of the concept of DCA and the packets are available in the last level cache of the processor. DDIO is limited to use 10% of the total last level cache size and restricted to use only two ways in a multi-way set associative cache.

DDIO can improve the efficiency of both data consumption and data delivery through I/O devices. The network data structures such as packet buffers, transmit and receive queues are all allocated in the LLC of the processor and the corresponding information is shared with the NIC as RX and TX descriptors. The NIC is triggered by the core to read the updated packet buffers during data transmission, whereas in DDIO the data to be transmitted is forwarded from the cache without any cache misses or evictions. Similarly when the data is received on the NIC, the NIC does a look up in its RX queue (generally based on a hash function) to find the corresponding RX queue on the core and the data is forwarded to the LLC with DDIO. The data is updated or allocated space in LLC based on the existence of memory addresses (eg., RX descriptors) in the cache hierarchy. The processor may be notified via an interrupt after the completion of transfer. When DDIO is not available, a similar set of operations are performed however, in this case, the data structures are all in main memory. All data receive operations with DMA invalidates the addresses if present in caches and delivers the data to main memory. Thus with DDIO, trips to main memory are not needed for both data consumption and data delivery operations of the NIC.

As mentioned earlier, DDIO has access to only 10% of the last level cache but

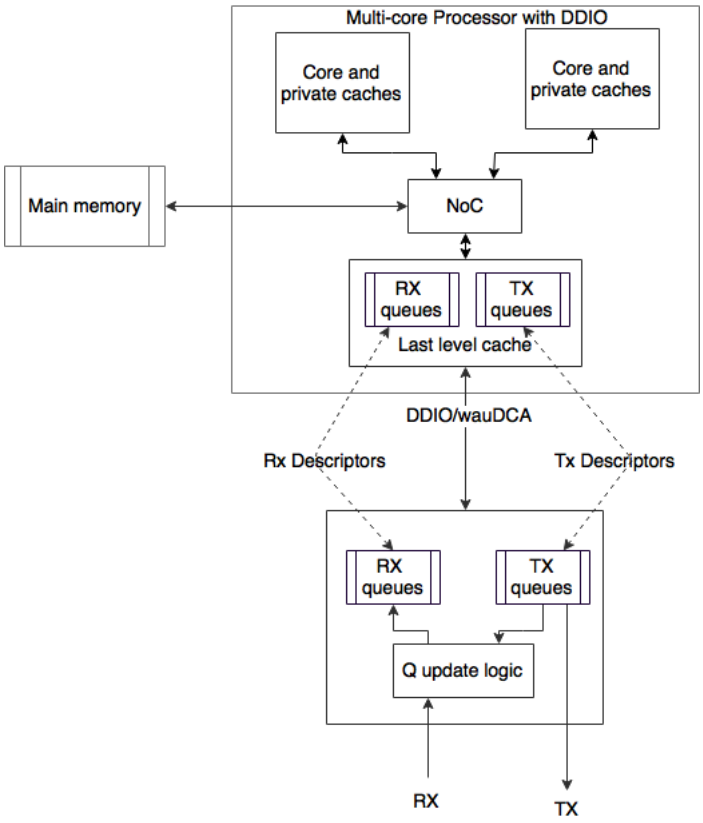

Figure 2.3: A system with write-allocate-write-update Direct Cache Access (wuaDCA)/ Intel DDIO

it is not dedicated for DDIO alone and available to other applications, the allocation is fixed and tracked by the system interconnect and cache management protocol. In cases when DDIO have used up the entire allocation or in case of set conflicts with new incoming data, the new incoming packets are allocated space in cache by evicting the old data onto the main memory according to the write-allocate, write-update cache policy. All Intel processors based on Haswell and newer architectures have DDIO enabled by default with no software or hardware dependencies. Figure 2.3 depicts the high level overview of system with DDIO as elaborated by Li, et al. in [24].

2.2 Yet Another Resource Negotiator

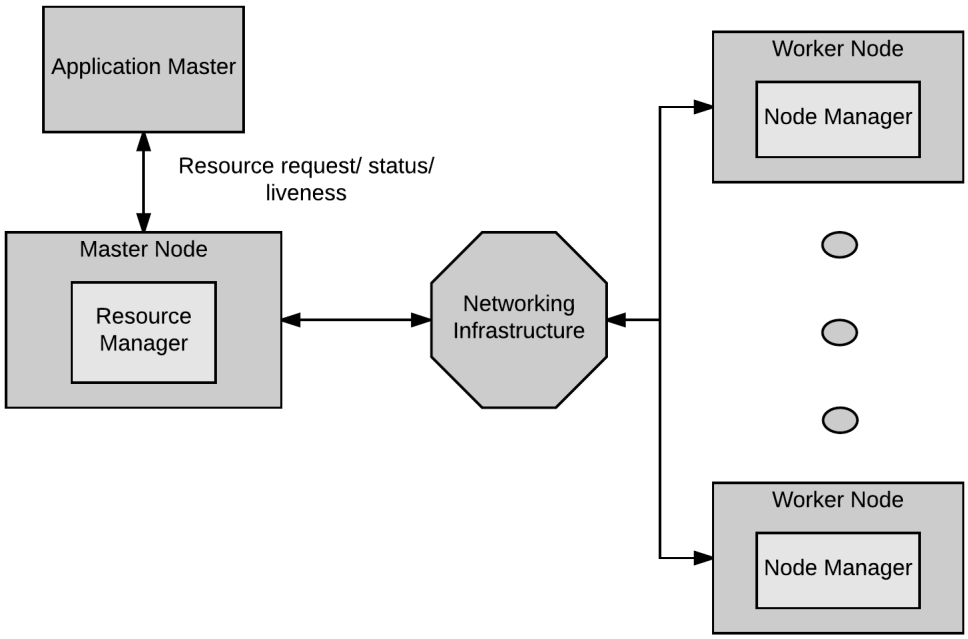

Figure 2.4: YARN Scheduler Architecture as described by Vavilapalli, et al. [1]

Big data frameworks, such as Hadoop and Spark, are most commonly used for developing big data applications. These frameworks employ a resource management framework called Yet Another Resource Negotiator (YARN). YARN is a scalable and efficient version of Hadoop which also provides services to manage the shared cluster [1]. YARN is composed of a Resource Manager (RM) per cluster that runs on the master node of the cluster and a Node Manager (NM) per worker node that manages the availability of resources, fault monitoring, and scheduling on each worker node.

The details of YARN architecture are well documented by Vavilapalli, et al. in [1], and the skeleton of the same is shown in Figure 2.4. A brief description of functions of the different components of YARN are as follows:

- Application Manager (AM) is an application which runs on the cluster, and co-ordinates the resources for execution of a particular application on the cluster. AM

is composed of set of static processes or a logical description of work or a service. It communicates periodically with the RM to maintain the records on the RM and to dynamically request resources for applications when necessary.

- Resource Manager is an arbiter which has interface to AM and NM. RM communicates with clients needing to run applications on the cluster via AM and communicates with NMs for cluster and resource access management. RM maintains a global cluster state against the digest of the applications running on the cluster.

- Node Manager runs on the worker nodes, its primary functions involve managing the containers, dependencies and monitoring the execution of the applications on the worker nodes. NM also maintains a heartbeat connection with RM for status updates and receiving instructions. NM also provides auxiliary services for data transfers, if the data is needed after the lifetime of the application. The auxiliary services can be requested by the AM on need basis.

YARN's architecture is particularly interesting as a potential target for enhancement for supporting dynamic usage of DDIO as explored in this thesis.

2.3 Big data workload characteristics

Characterizing the workload leads to a better understanding of the trade offs involved, and in turn aids better system design. This section talks more about the behavior of some of the workloads from the big-data applications that give us interesting insights into designing optimal systems. The workload characterization is carried out using the benchmarks of a particular area of applications and those characteristics of the benchmarks are used to drive system design. Big data as a field is very diverse in nature and the figure 2.5 derived from [2] gives an insight into the different kinds of data analyzed in big data applications.

Big data systems are continuously evolving and with the classification as discussed by IBM and as shown in figure 2.5 the numerous types of data makes it very difficult to come up with benchmarks which are comprehensive of all the characteristics of such applications. Over the last few years various benchmark suites have been

Figure 2.5: Big data classification as described by Mysore, et al. [2]

released such as Cloudsuite, Hibench, LinkBench, Hive and BigDataBench. Big data applications are generally built upon the MapReduce, Hadoop and Spark frameworks and characterizing these frameworks can give greater insights on techniques to optimize computing platform's performance.

The big data workloads are considered to be scale out workloads [18] and differ from the traditional computing workloads. Though tailoring the processor design to a specific set of applications is not desirable in a general computing environment, the knowledge of characteristics of workloads to which the systems cater to will help gain maximum performance out the system. The scale-out workloads are a diverse set of non-overlapping application and management software for the distributed infrastructure operating on large data sets which are stored in distributed file systems. The examples of scale out workloads (essentially different kind of applications which run in the cloud) are:

- Data Serving workloads comprising of web applications which store data and retrieve the data based on the requested key (using Key-value store frameworks).
- Analytic applications use map-reduce frameworks. These applications are limited by communication resources as the tasks perform intensive data read/write operations in a distributed file system. Analytic applications are dominated by branch instructions (with more than 90%) [25].
- Streaming applications such as online media streaming occupy close to 50% of the internet traffic. These applications continuously fetch, encode/decode, packetize and transmit relevant data at high speeds to support high bandwidth, and high quality subscriptions [26].
- Search engine and other web services make use of the cloud infrastructure to provide the requested service. The important criteria in such applications is latency and load balancing in the back end for optimum usage of available resources.

The frameworks mentioned earlier have significant differences and also noticeable impact on performance of a particular application. Big Data applications are data intensive and significant disparities exist in the instruction footprints of different sub-classes of

applications using different software stacks/frameworks. The summary of observations on the big data workload characteristics as made in [6, 7, 18, 25, 27, 28] are as follows:

- Read operations in HBase (a non-relational and distributed database) are I/O intensive as reads are converted into get operations using a key, and use the map-reduce framework to traverse through the data.
- Hive based interactive analysis determine set differences over E-commerce transactions data-set, this is observed to be I/O intensive as each record is a key-value pair.
- Filtering of data is an important operation in interactive data analysis, and uses queries to filter data. With E-commerce dataset, such filtering operations are observed to be I/O intensive.
- Sorting the E-commerce transactions data-set, projecting implications of the data using relational algebra is also an I/O intensive task when done using Shark and CPU intensive using Spark.
- Counting of words in a data-set is a basic and foremost operation in data analysis and statistics. Implementing counting on a huge data-set like Wikipedia, where each record is a 64KB key-value pair is observed to be I/O intensive with Spark and CPU intensive with Hadoop.
- Finding relevant information from Wikipedia data-set using grep is I/O intensive when implemented using Spark and CPU intensive when implemented using Hadoop.

Thus to summarize even similar operations on similar data-sets differ in their system behavior when implemented with different framework. The frameworks used have a big impact on the performance of applications on same underlying hardware infrastructure. The software stacks used also have a significant impact on the characteristics such as CPI, LLC, Icache and Dcache behavior. Most applications have intense data movement operations and analytical applicatons are dominated with branches. The disparate nature of big data workloads thus calls for more innovative co-design of the hardware and software infrastructure to derive maximum possible performance.

2.4 Related work

The growing application of big data analytics calls for an overhaul of all the different subsystems and more coherent design of the data centers where most of these applications reside . This section gives insight into different methods propounded for accelerating big data applications and frameworks.

2.4.1 Storage systems optimization

High performance storage system which scales and supports the characteristics of scale out workloads is essential for latency sensitive applications. One such solution is RAMcloud [29] where clusters are built with high DRAM capacities distributed over a high-speed interconnect network. The state of the art distributed memory mechanisms such as memcached which needs the infrastructure to be scalable are shown to be served well by RAMcloud's approach. RAMCloud is distinct in the way that all the storage is in the DRAMs which are distributed over numerous servers. With innovations in DRAM technology the TCO of such architecture would be cheaper than any other mechanisms to accelerate storage. The duplicates needed to handle the refresh of each DRAM cell increases the overall cost for every bit to be stored in RAMCloud. The high cost and energy usage per bit in RAMcloud makes it worse than the HDD based systems when the application is not latency sensitive.

Triple-A [30], a flash array based storage solution for sustaining low latency I/O workloads proposed by Jung, et al. aim to address the issues with SSD based solutions by using NAND flash clusters, that can be optimized and reshaped based on the resource contentions of the application it caters to. Moon, et al. demonstrated in [31] the critical system configurations in Hadoop based systems for optimal use of SSD performance and also show that SSDs as temporary storage devices and HDDs as permanent storage for Hadoop Distributed file system is most cost-effective configuration.

Another idea which has gained traction is in-store processing which provides low latency and scalable architectures. One of the recent proposals is Blue DBM[32] which proposes to use flash storage for storing large data-sets of size 5TB to 20TB for sustaining high performance random accesses instead of traditional HDD based storage.

BlueDBM will not need 1000s of servers to hold the data as in case of RAMcloud, but can hold a equivalent large dataset in a rack with 20 servers. The custom architecture of BlueDBM with integrated network and flash design which provides uniform access latency. The power consumption of each server node in BlueDBM is 20% greater than the typical data center servers. The software modifications to effectively use in-store processing infrastructure restricts its usage.

2.4.2 Re-configurable optimization

FPGA based memcached acceleration proposed in [33] pushes the functionality of the software based key-value caching mechanism onto the FPGA to provide low latency access which is critical for web services. The FPGA based solution is to accelerate Memcached [13] and provides performance benefits with advantage of tightly integrate network, memory and compute onto one appliance. High performance FPGA solutions come with higher cost as compared to traditional server infrastructure. Lavasani, et al. [34] propose a hybrid architecture with FPGA based accelerator to process the network packets and for speculative execution based on traces obtained from the profiling of the application. When the speculation is wrong then the packet is forwarded to CPU for processing.

MemcachedGPU [35] by Hetherington, et al. propose a GPU-based apparatus for low latency and high throughput network processing. The solution is shown to service line rates close to 10GbE conforming to the latency requirements of key-value store applications. Exploiting the parallelism available on GPUs for accelerating key value stores and hiding the latency is compelling but the reliance on CPU for I/O, high programmability and energy costs of GPU is restrictive. MegaKV [36] proposed by Zhang, et al. also makes a case for GPUs to be used to accelerate the data and network I/O intensive operations in key-value stores.

Tsai, et al. propose Jenga [37], a configurable memory hierarchy which can specialize itself to needs of the application and show significant improvements in EDP up to 85%. The software defined hierarchies in Jenga are built using a simple hardware mechanism to provide flexible hardware substrate and changes to the OS run-time to configure the hierarchy. The technique is also shown to improve performance by elimi-

nating unnecessary accesses due to rigid hierarchies.

2.4.3 Holistic optimization

Scale-Out NUMA [38] proposed by Daglis, et al. facilitates the interaction between application, OS and the networking infrastructure for distributed in-memory processing with a new hardware remote memory controller architecture, communication protocol and programming model. The enhancements are shown to improve the latency and bandwidth performance of the remote data access by 5x while eliminating the kernel, network and bus overloads.

MICA [39] proposes software centric improvements to Key-value store operations to provide high throughput rates. MICA explores the design choices to exploit the parallelism in multi-core systems, reducing overhead of network stack and optimum memory management for fast data access. MICA reports consistent performance across different kinds of applications and one major drawback is for the optimization to be carried out holistically else the performance degrades if there is a missing component.

Similarly Li, et al. [24] provide an insight into holistic software and hardware optimization for achieving high speed key-value stores(KVS) and propose design principles for full system architecture, widely used software such as key value stores in data center services. This explores the design space to reduce the round trip delay while achieving higher throughput by profiling the workloads and optimizing the network, I/O subsystem and memory hierarchy.

The common thread in all the proposals has been to accelerate either the framework or the underlying infrastructure or re-configuring the system based on the application's need. The major bottlenecks identified are the data transfer overheads between storage, network and the compute. Most proposals overcome this by additional hardware, through software optimization or using a completely different system architecture with corresponding trade offs.

Chapter 3

Evaluation and Experimentation

The study of the big data workload characteristics shows that these workloads benefit when the network, compute and memory work in close tandem with each other. In this work we evaluate the benefits of DDIO using simple micro-benchmarks and network I/O intensive big data benchmarks. We also explore the phases of the applications where DDIO offers a real advantage and phases where DDIO is a disadvantage. With such identified application characteristics, we can employ a dynamic model for enabling and disabling DDIO. When the workload is known to be CPU intensive we can disable DDIO as its primary operation is to improve I/O performance. However, when the workload is I/O intensive we can use the availability of DDIO in a more effective ways such as increasing the cache allocation DDIO can use, cache ways DDIO can access and also the levels of cache DDIO can source/sink data from/to. Similar efforts are made to use caches and SRAM banks in a more efficient way based on the application's behavior, Jenga [37] and Jigsaw [40] are early such efforts to use memory hierarchy as a resource rather than a rigid structure.

3.1 Experiment Setup

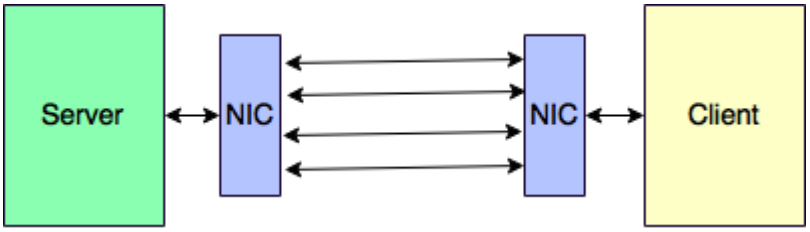

Figure 3.1: Experimental setup

Our experimental setup consists of two dual-socket systems with Intel Xeon E5-2697 v2 processors with specifications as shown in table 3.1. The processors include an implementation of write-allocate write-update direct cache access (wuaDCA) also known as Intel DDIO, and thus have the ability to transact network I/O from the last level cache. Each system has DDR3 memory of size 128GB and 4 Intel X520-QDA1 network interface cards with four 10GbE ports on each. The two systems are connected in simple loop-back mode as we are only concerned about the network I/O characteristics at the producer/consumer ends. The software infrastructure is composed of CentOS 7.0 (kernel 3.10.0-123.8.1), Intel Data Plane Development kit 2.2.0 [41] and Intel Vtune [42] for monitoring the hardware performance counters. Figure 3.1 is a pictorial representation of the setup and table 3.2 outlines the details of the setup.

Table 3.1: Processor Configuration

Cores	12 Cores, 2.3 GHz, Haswell Architecture
Number of threads	2 per core (2-way hyper-threading)
Frequency	2.7 GHz (3.5 GHz Turbo)
L1 Cache/s	32KB, 8-way set associative I and D-caches per core
L2 Cache/s	256KB, 8-way set associative unified cache per core
L3 Cache/s	30MB, 20-way set associative unified cache

Table 3.2: System Configuration

Sockets	2
NUMA-Domains	2
DRAM Memory	128GB
NIC	4 x Intel 4x10GbE NIC
Storage	Local 1TB HDD

3.2 Micro-Benchmark

One of the micro-benchmark which we use here is a TCP/IP based server client application which communicates over our loopback network. The first micro-benchmark is a ping-pong application for exchanging the messages between the two systems, and helps determine the end to end latency with and without DDIO. The first benchmark is ran with smallest to largest payload size possible to determine the effect of data payload size variations on the performance.

The second micro-benchmark is a streaming application, where data is continuously streamed from one machine to another. The benchmark is used with different stride access distances on the receiving end to evaluate if there is variance in performance due to stride distances with and without DDIO.

The OS network stack can be a major bottleneck to utilize the available network efficiently, and can mask the performance of DDIO. Hence we developed a DPDK [41] based server client benchmark with similar functions as previous micro-benchmarks, to overcome the overheads of using OS network stack. DPDK is used for fast packet processing and can complete a packet send/receive cycle in less than 100 CPU cycles. DPDK bypasses the linux kernel network stack using DPDK libraries and poll-mode drivers which are in user space (Figure 3.2). The DPDK based benchmarks helped to determine the performance of DDIO with respect its function and eliminates the overheads due to OS.

The micro-benchmarks are all profiled with Intel's Vtune Amplifier which uses hardware performance counters to track the metrics important for our experiments such as: Instructions per cycle (IPC), Last level cache (LLC) misses and hits, Instructions

committed and CPU utilization.

Since the experiments were carried out on a real machine and not in a controlled simulation environment, we use LMBench to validate off the shelf latency and bandwidth performance of the CPUs, NICs and other components in the experimental setup.

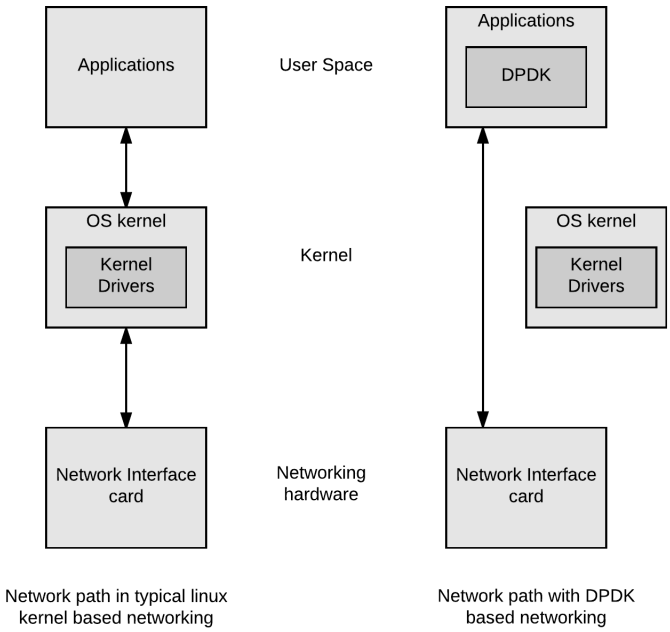

Figure 3.2: Linux TCP/IP vs DPDK packet processing paths

3.3 Cloudsuite data caching benchmark

The workloads that benefit maximum by using DDIO are those that consume data arriving through network I/O, as soon as it arrives and have very low temporal locality. Real world applications that perform such network I/O intensive tasks include financial trading, Key-value stores, streaming and graph processing engines operating on large data-sets. In this work we use the data caching benchmark from the Cloudsuite benchmark suite which simulates the Twitter data caching application using Twitter data-set. The metrics of interests for our experiments are the latency of each request and requests served per second with DDIO enabled and disabled.

The benchmark consists of a memcached data caching server, simulated using a scaled up Twitter data-set of size 10GB. The server can be configured to use any number of threads, but the benchmark is known not to scale well beyond four threads. The benchmark can use multiple NICs if available by assigning different IP addresses and starting different server processes. On the client side, the client sends a key-value pair request to the server and waits for the server's key-value pair response for each connection established. The client can send requests to multiple server or to same server through multiple ports. The though-put and the metrics are measured on the client side.

The server is warmed up with target server memory, corresponding scaling factor for data-set and number of client worker threads. All the other experiments are carried out after the server warms up to a stable state.

Chapter 4

Results

We evaluated the performance characteristics with and without DDIO on the setup and with benchmarks as described in chapter 3. The results of the experiments performed are presented in this chapter.

4.1 Experiments with server client micro-benchmark

Figure 4.1 shows the LLC miss-rate of the data transfer using Linux kernel stack for different payload sizes. Figure 4.2 shows the LLC miss-rate of data transfer with DPDK framework for different payload sizes.

The difference in performance metrics across different data payload sizes with DDIO and without DDIO cases shows that: with DDIO we can overcome the trips to main memory, which is not possible otherwise without DDIO. There is a significant difference in LLC miss-rate and the OS kernel stack version in Figure 4.1 shows a LLC miss-rate of close 20% which is due to the OS kernel network stack overhead. In Figure 4.2 the LLC miss-rate is of the micro-benchmark based on DPDK, the graph shows the LLC miss-rate is close to zero with DDIO enabled (DDIO-on), and with DDIO disabled (DDIO-off) almost all the packets to be processed are missed at LLC.

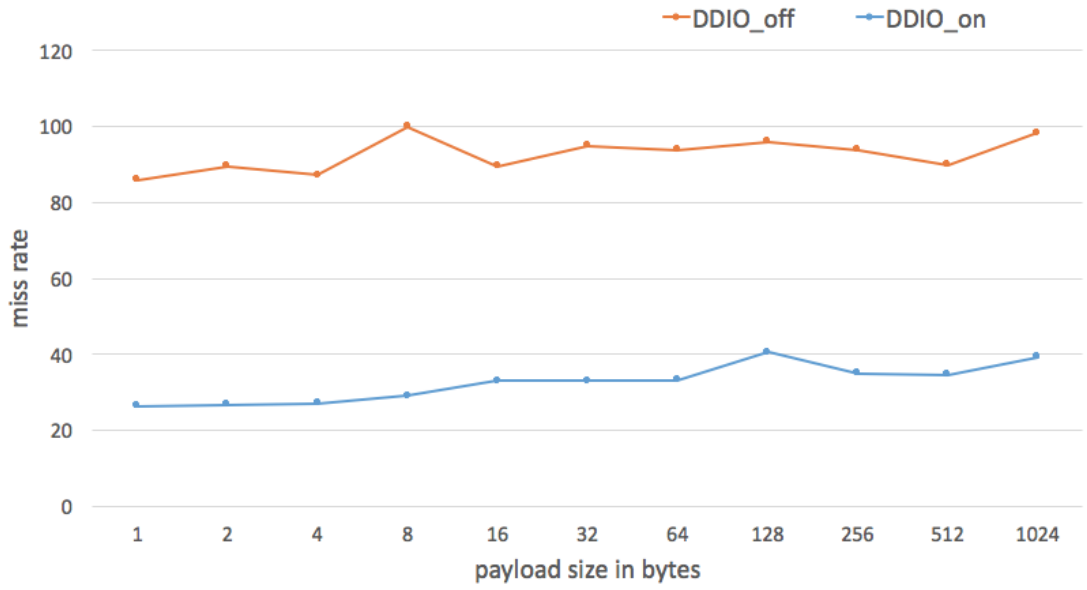

Figure 4.1: LLC miss rate with 1GB data transfer of micro-benchmark using linux kernel network stack

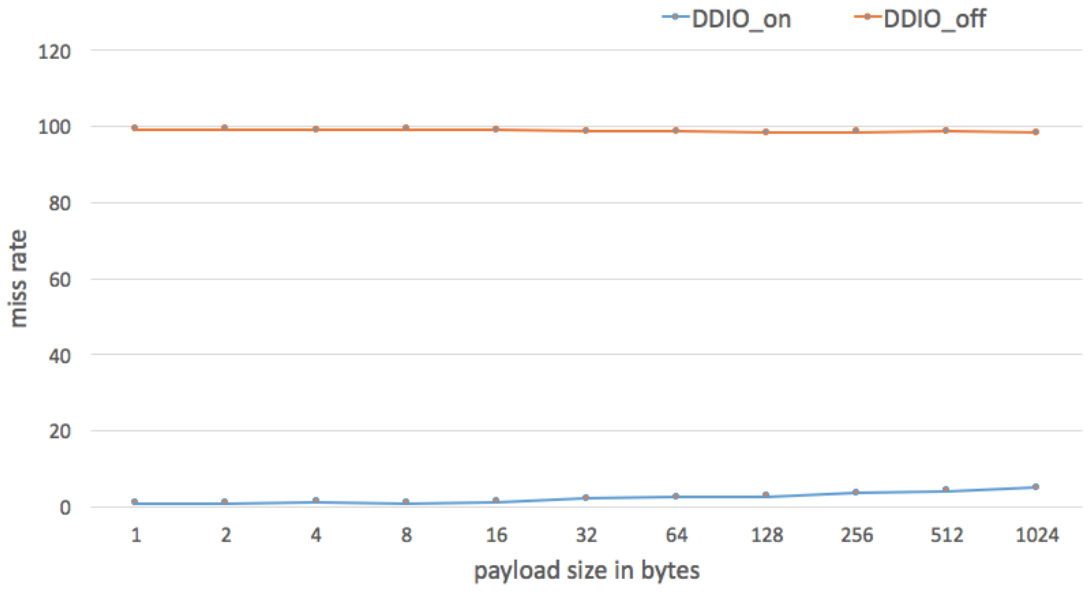

Figure 4.2: LLC miss rate with 1GB data transfer of micro-benchmark using DPDK libraries

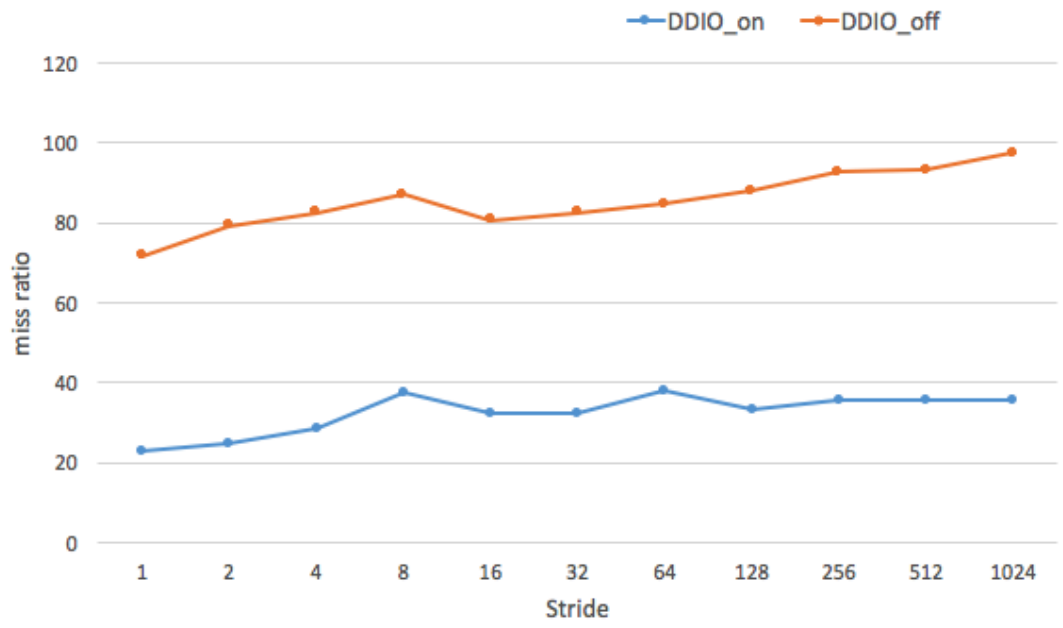

Figure 4.3: LLC miss rate of different stride accesses of micro-benchmark using Linux network stack

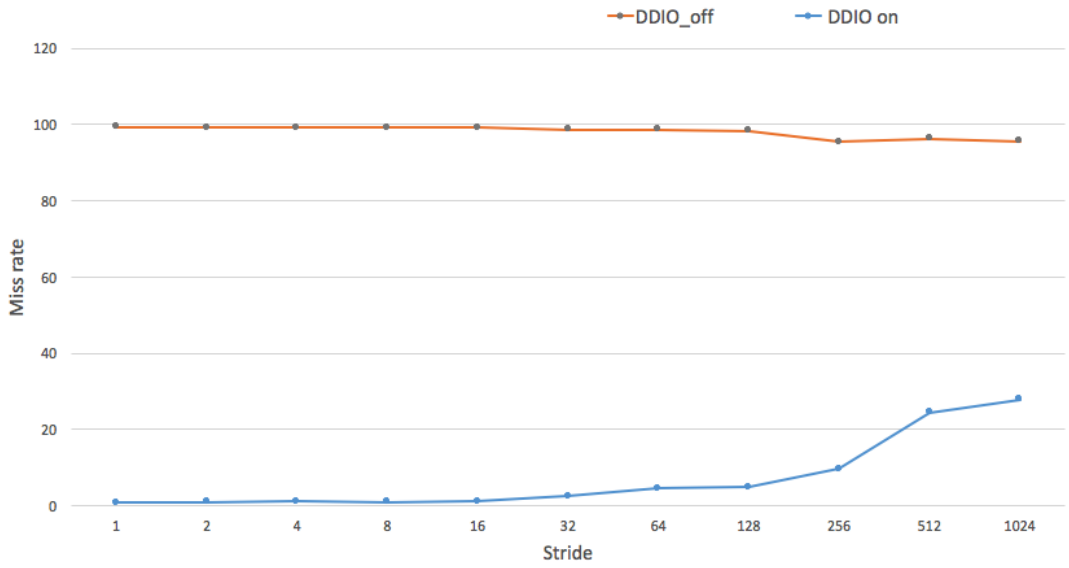

Figure 4.4: LLC miss rate of different stride accesses of micro-benchmark using DPDK libraries

Figure 4.3 and 4.4 show the behavior of stride access micro-benchmark. This helps us evaluate if the DDIO performance is related to the use/reuse distance of the

data arriving from the network. There are two versions one using Linux OS network stack and other using DPDK libraries, and the corresponding behaviors with DDIO enabled and disabled are depicted in the graphs in Figure 4.3 and 4.4 respectively. We can see that the overhead due to OS network plays a prominent role in LLC behavior. We can also see that the LLC miss rate increases when the stride distance increases, this indicates that if the data arriving from the network is not used before its pushed out the last level cache DDIO is not effective.

4.2 Experiments with Cloudsuite data-caching benchmark

The Data caching benchmark is network I/O intensive and requires high network speeds. Our setup is not network bound as it is a loop back configuration with bandwidth of 160Gbps. The data caching application is composed of a memcached data caching server which simulates Twitter caching server, and multiple clients with data-set ranging from 1-30GB. The metrics measured are the latency and the LLC miss-rate on the client side for different requests per second(RPS) rate specified by the client to the server.

The first experiment had one server and one client application running on two different hosts, and the application was simulated with a specified RPS rate for duration of one minute and 100 iterations after server had warmed up. Figure 4.5 shows the average latency and average miss-rate for different RPS values with DDIO enabled and Figure 4.6 shows the behaviour of same experiment with DDIO disabled.

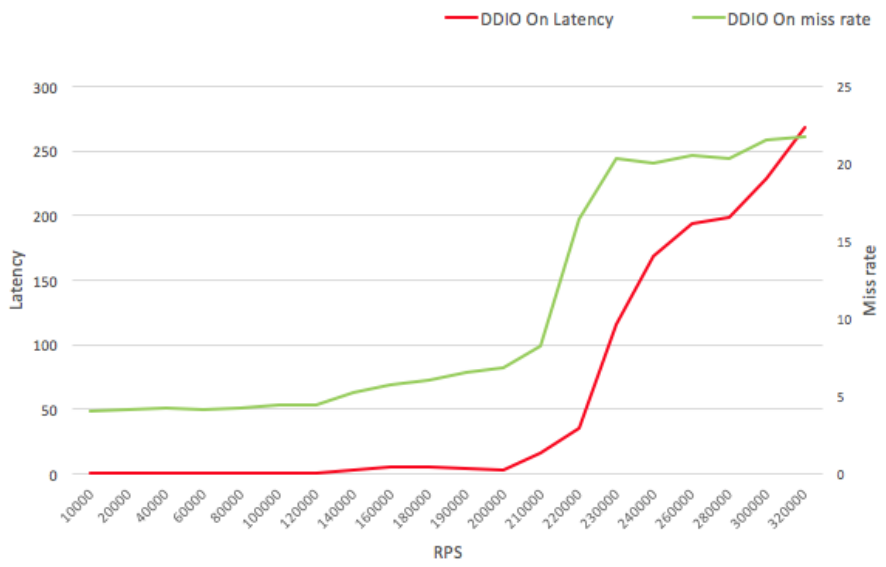

Figure 4.5: Latency and miss rate vs RPS on client side with DDIO enabled with Cloudsuite data caching benchmark

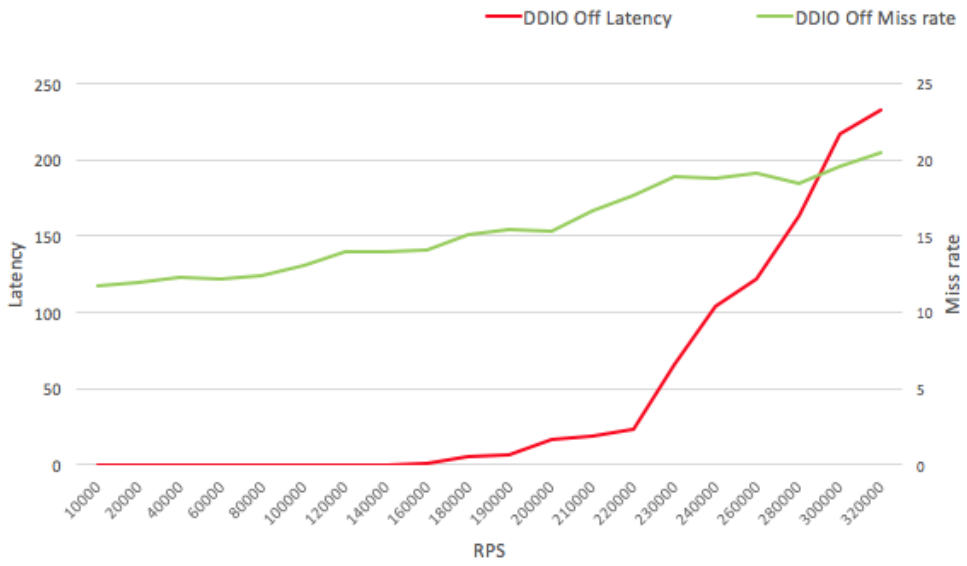

Figure 4.6: Latency and miss rate vs RPS on client side with DDIO disabled with Cloudsuite data caching benchmark

Figure 4.7 shows the miss rate and Figure 4.8 shows the latency for different RPS rates of the data caching server client with DDIO disabled and DDIO enabled. This

shows the difference in miss rate at lower RPS rates with DDIO disabled and enabled is not very significant, and the latency of the requests being served is almost similar. At higher RPS rates the performance of the benchmark with DDIO enabled is worse than of the DDIO disabled version, this is because of the older responses being replaced by newer responses before the client could use it, increasing the last level cache misses and in turn the latency. The data-caching benchmark uses the Linux kernel stack and the TCP/IP packet reconstruction could also interfere with the LLC behavior.

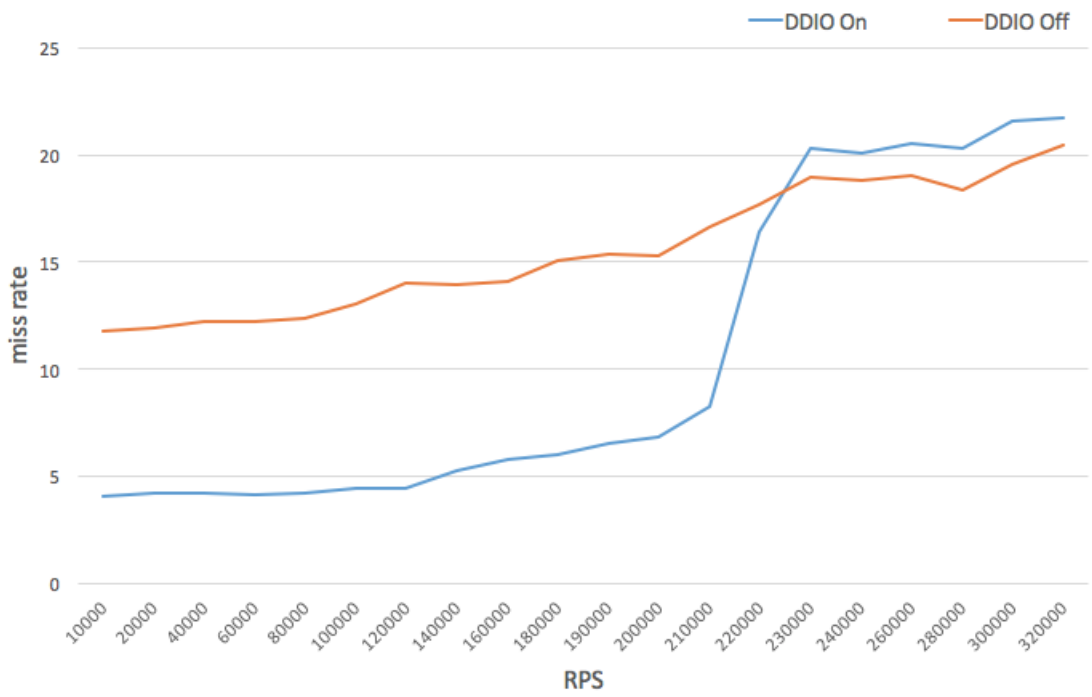

Figure 4.7: Miss rate of Cloudsuite data caching benchmark with single server client with DDIO enabled and disabled

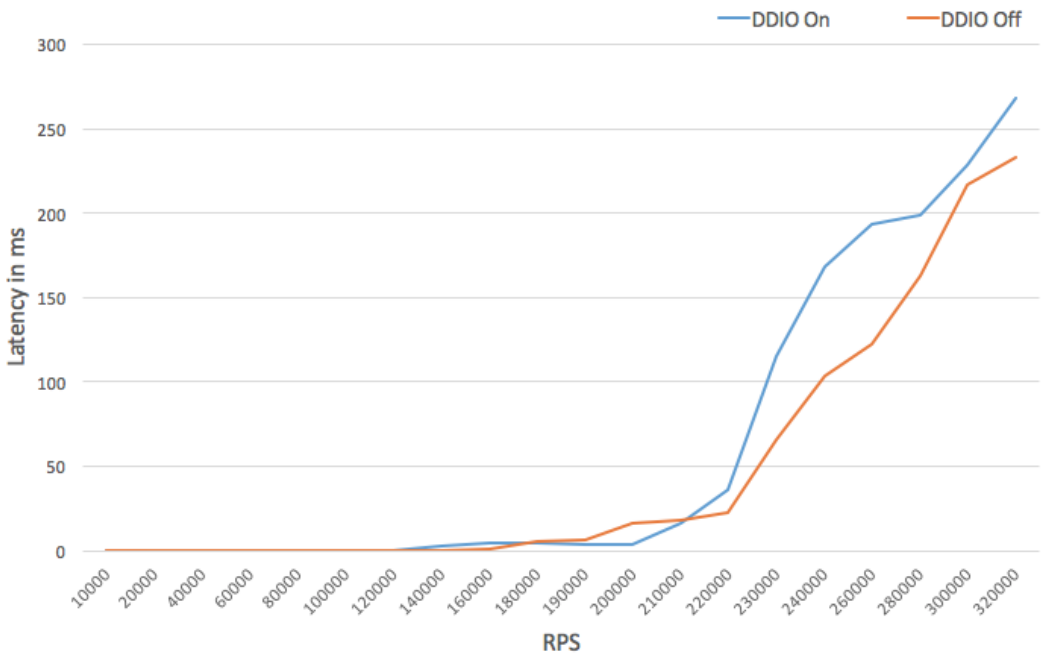

Figure 4.8: Latency of Cloudsuite data caching benchmark with single server client with DDIO enabled and disabled

When the Latency for DDIO enabled and disabled scenarios are plotted it shows a clear difference of DDIO enabled performance degrading at RPS rates lower than the that of the DDIO disabled case, but at lower RPS rates system's performance is similar in both DDIO enabled and disabled scenarios. This shows that there is a case for DDIO to be disabled in certain cases, for this benchmark it can be disabled with no loss in performance at lower RPS rates, and can also obtain better performance with DDIO disabled at higher RPS rates.

Similar behavior can be seen when there are multiple client and server processes running in Figure 4.9 where the performance with DDIO enabled is lower than when its disabled.

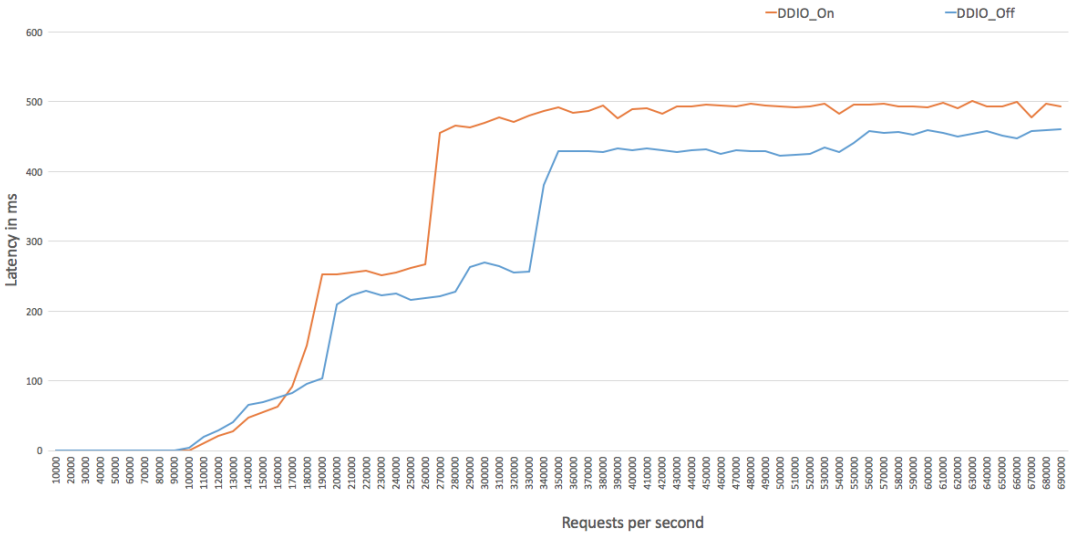

Figure 4.9: Latency vs RPS of Data caching benchmark of Cloudsuite with multiple server-client processes

The degradation in performance at high speed communication is because the limited LLC allocation for DDIO cannot hold all the associated RX/TX queues and data buffers. When the new packets arrive the LLC evictions causes memory traffic to increase.

Chapter 5

Conclusion and Future Directions

5.1 Conclusion

The number of cloud based applications is continuously rising, and researchers are propounding new techniques to accelerate these applications. In this thesis, we navigated through a different path of evaluating whether existing features are being used effectively. We evaluated the effectiveness of DDIO in context of big data applications which are deployed in the cloud. DDIO is useful when there is a strong producer consumer relation between the source and the destination of data. With high speed network and high rate of network bound data to be processed, we have shown that DDIO can cause the performance to degrade because incoming data into cache will push out unprocessed data from cache. This thrashing of cache increases the memory traffic and in turn negates the performance gained with DCA.

Disabling and enabling of DCA is an unique feature in our setup and used in our experiments, which can be used to avoid performance degradation caused due to DCA (enabled by default) by statically enabling/disabling DCA. We used a set of micro-benchmarks and Cloudsuite's data caching benchmark to identify, and validate the scenarios where we lose performance due to DCA. From the results, we observe that there is scope for DCA to be used dynamically based on the kind of application and its characteristics. The usage may not be limited to just being disabled or enabled, but can also be extended to overcome the fixed resource allocation to DCA by making it dynamic on need basis.

5.2 Future Directions

In order to evaluate the dynamic model of DCA, we have to use a simulation environment with necessary changes to support it. With DCA as an example the next step will be to evaluate the implications on overall system design and performance of dynamically managing the micro-architectural features, and the trade-offs involved. The targeted domain of big data applications in this thesis is one of the largest occupants of data center infrastructure.

The behavior of DCA being dependant on the kind of software stacks used, developing benchmarks with least overheads is essential. As DCA is closely tied to cache hierarchy, dynamically enabling and disabling it can have implications on the system energy-delay product. The results of our evaluation serve as good starting point for studying the trade-offs in terms of power consumption.

The other facet is to explore the support in OS runtime and other management frameworks for a dynamic model of configuring architectural features. We briefly describe the architecture of YARN, and how it could be used but it needs more in depth analysis. The changes to be made in the hardware design also needs to be explored.

The observations pave way for exploring other micro-architectural features which can be used more effectively based on the workloads running on the infrastructure.

Bibliography

[1] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O 'malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. Apache Hadoop YARN: Yet Another Resource Negotiator. *SOCC '13 Proceedings of the 4th annual Symposium on Cloud Computing*, 13:1–3, 2013.

[2] Divakar Mysore, Shweta Jain, and Shrikant Khupat. Big Data Architecture and Patterns. `https://www.ibm.com/developerworks/library/bd-archpatterns1/index.html`, 2013 (accessed April 28, 2017).

[3] Peter ffoulkes. InsideBIGDATA Guide to The Intelligent Use of Big Data on an Industrial Scale. `https://insidebigdata.com/white-paper/guide-big-data-industrial-scale/`, 2017 (accessed April 25, 2017).

[4] Rob Kitchin. Top 10 Big Data Trends for 2017. `https://www.systemonline.cz/clanky/big-data.html`, 2017 (accessed April 25, 2017).

[5] Krushnaraj Kamtekar. Performance Modeling of Big Data. `https://www.cse.wustl.edu/~jain/cse567-15/ftp/bigdata.pdf`, 2015 (accessed April 24, 2017).

[6] Martin Dimitrov, Karthik Kumar, Patrick Lu, Vish Viswanathan, and Thomas Willhalm. Memory system characterization of big data workloads. *Proceedings - 2013 IEEE International Conference on Big Data, Big Data 2013*, (1):15–22, 2013.

[7] Kapil Bakshi. Considerations for big data: Architecture and approach. *IEEE Aerospace Conference Proceedings*, pages 1–7, 2012.

[8] Wen Xiong, Zhibin Yu, Zhendong Bei, Juanjuan Zhao, Fan Zhang, Yubin Zou, Xue Bai, Ye Li, and Chengzhong Xu. A characterization of big data benchmarks. In *2013 IEEE International Conference on Big Data*, pages 118–125. IEEE, oct 2013.

[9] Lei Wang, Jianfeng Zhan, Chunjie Luo, Yuqing Zhu, Qiang Yang, Yongqiang He, Wanling Gao, Zhen Jia, Yingjie Shi, Shujie Zhang, Chen Zheng, Gang Lu, Kent

Zhan, Xiaona Li, and Bizhu Qiu. BigDataBench: A big data benchmark suite from internet services. *Proceedings - International Symposium on High-Performance Computer Architecture*, pages 488–499, 2014.

[10] Li Zhao, Laxmi N. Bhuyan, Ravi Iyer, Srihari Makineni, and Donald Newell. Hardware support for accelerating data movement in server platform. *IEEE Transactions on Computers*, 56(6):740–753, 2007.

[11] Steen Larsen, Parthasarathy Sarangam, Ram Huggahalli, and Siddharth Kulkarni. Architectural breakdown of end-to-end latency in a TCP/IP network. *International Journal of Parallel Programming*, 37(6):556–571, 2009.

[12] Amit Kumar, Ram Huggahalli, and Srihari Makineni. Characterization of direct cache access on multi-core systems and 10gbe. In *15th International Conference on High-Performance Computer Architecture (HPCA-15 2009), 14-18 February 2009, Raleigh, North Carolina, USA*, pages 341–352, 2009.

[13] Michaela Blott, Kimon Karras, Ling Liu, Kees Vissers, Jeremia Bär, and Zsolt István. Achieving 10Gbps line-rate key-value stores with FPGAs. *HotCloud (USENIX Workshop on Hot Topics in Cloud Computing)*, 2013.

[14] Rodrigo A. Melo, David M. Caruso, and Salvador E. Tropea. Memory-mapped I / O over Dual Port BRAM on. *III Southern Conference on Programmable Logic (SPL)*, pages 0–5, 2012.

[15] Quentin Hardy. Intel Betting on (Customized) Commodity Chips for Cloud Computing. `https://bits.blogs.nytimes.com/2014/12/19/intel-betting-on-customized-commodity-chips-for-cloud-computing/?_r=0`, 2014 (accessed April 25, 2017).

[16] R Huggahalli, R Iyer, and S Tetrick. Direct cache access for high bandwidth network I/O. *32nd International Symposium on Computer Architecture ISCA05*, 33(2):50–59, 2005.

[17] Intel Corporation. Intel ® Data Direct I / O Technology (Intel ® DDIO): A Primer. (February), 2012.

[18] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. Clearing the Clouds: A Study of Emerging Scale-out Workloads on Modern Hardware. *Asplos'12*, 40(Asplos):37–48, 2012.

[19] Luiz Barroso and Urs Hoelzle. *The Datacenter as a Computer:An Introduction to the Design of Warehouse-Scale Machines*. Morgan Claypool, 2009.

[20] Dennis Abts and John Kim. *High Performance Datacenter Networks:Architectures, Algorithms, and Opportunities*. Morgan Claypool, 2011.

[21] Yuan Xie. *Emerging memory technologies: Design, architecture, and applications*, volume 9781441995. 2014.

[22] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 365–376, New York, NY, USA, 2011. ACM.

[23] Rodrigo A. Melo, David M. Caruso, and Salvador E. Tropea. Memory-mapped I / O over Dual Port BRAM on. *III Southern Conference on Programmable Logic (SPL)*, pages 0–5, 2012.

[24] Sheng Li, Pradeep Dubey, Hyeontaek Lim, Victor W. Lee, Jung Ho Ahn, Anuj Kalia, Michael Kaminsky, David G. Andersen, O. Seongil, and Sukhan Lee. Architecting to achieve a billion requests per second throughput on a single key-value store server platform. *Proceedings of the 42nd Annual International Symposium on Computer Architecture - ISCA '15*, pages 476–488, 2015.

[25] Lei Wang, Rui Ren, Jianfeng Zhan, and Zhen Jia. Characterization and architectural implications of big data workloads. *ISPASS 2016 - International Symposium on Performance Analysis of Systems and Software*, pages 145–146, 2016.

[26] Ilias Marinos, Robert N.M. Watson, Mark Handley, and Randall R. Stewart. Disk, crypt, net: Rethinking the stack for high-performance video streaming. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 211–224, New York, NY, USA, 2017. ACM.

[27] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang. The hibench benchmark suite: Characterization of the mapreduce-based data analysis. In *2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010)*, pages 41–51, March 2010.

[28] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi. A case for specialized processors for scale-out workloads. *IEEE Micro*, 34(3):31–42, May 2014.

[29] John Ousterhout, Arjun Gopalan, Ashish Gupta, Ankita Kejriwal, Collin Lee, Behnam Montazeri, Diego Ongaro, Seo Jin Park, Henry Qin, Mendel Rosenblum, Stephen Rumble, Ryan Stutsman, and Stephen Yang. The ramcloud storage system. *ACM Trans. Comput. Syst.*, 33(3):7:1–7:55, August 2015.

[30] Myoungsoo Jung, Wonil Choi, John Shalf, and Mahmut Taylan Kandemir. Triple-a: A non-ssd based autonomic all-flash array for high performance storage systems. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, pages 441–454, New York, NY, USA, 2014. ACM.

[31] Sangwhan Moon, Jaehwan Lee, Xiling Sun, and Yang-Suk Kee. Optimizing the hadoop mapreduce framework with high-performance storage devices. *J. Supercomput.*, 71(9):3525–3548, September 2015.

[32] Sang-Woo Jun, Ming Liu, Sungjin Lee, Jamey Hicks, John Ankcorn, Myron King, Shuotao Xu, and Arvind. Bluedbm: An appliance for big data analytics. In *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ISCA '15, pages 1–13, New York, NY, USA, 2015. ACM.

[33] Sai Rahul Chalamalasetti, Kevin Lim, Mitch Wright, Alvin AuYoung, Parthasarathy Ranganathan, and Martin Margala. An FPGA memcached appliance. *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays - FPGA '13*, page 245, 2013.

[34] Maysam Lavasani, Hari Angepat, and Derek Chiou. An FPGA-based In-Line Accelerator for Memcached. *IEEE Computer Architecture Letters*, 13(2):57–60, 2014.

[35] Tayler H. Hetherington, Mike O'Connor, and Tor M. Aamodt. Memcachedgpu: Scaling-up scale-out key-value stores. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, SoCC '15, pages 43–57, New York, NY, USA, 2015. ACM.

[36] Kai Zhang, Kaibo Wang, Yuan Yuan, Lei Guo, Rubao Lee, and Xiaodong Zhang. Mega-kv: A case for gpus to maximize the throughput of in-memory key-value stores. *Proc. VLDB Endow.*, 8(11):1226–1237, July 2015.

[37] Po-An Tsai, Nathan Beckmann, and Daniel Sanchez. Jenga: Software-defined cache hierarchies. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, pages 652–665, New York, NY, USA, 2017. ACM.

[38] Stanko Novakovic, Alexandros Daglis, Edouard Bugnion, Babak Falsafi, and Boris Grot. Scale-out numa. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, pages 3–18, New York, NY, USA, 2014. ACM.

[39] Hyeontaek Lim, Dongsu Han, David G. Andersen, and Michael Kaminsky. Mica: A holistic approach to fast in-memory key-value storage. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 429–444, Seattle, WA, 2014. USENIX Association.

[40] Nathan Beckmann and Daniel Sanchez. Jigsaw: Scalable software-defined caches. *Parallel Architectures and Compilation Techniques - Conference Proceedings, PACT*, pages 213–224, 2013.

[41] Intel Corporation. Intel ® Data Plane Development Kit(Intel ® DPDK): API Reference . 2014.

[42] Intel Corporation. Intel ® VTune Amplifier 2016 : User guide. 2016.