Disaggregated Architecture for at Scale Computing

Chung-Sheng Li[1], Hubertus Franke[1], Colin Parris[2] and Victor Chang[3]
[1]IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598, U.S.A.
[2]GE Global Research Center, One Research Circle, Niskayuna, NY 12309, U.S.A.
[3]School of Computing, Creative Technologies and Engineering, Leeds Beckett University, Leeds LS6 3QS, U.K.

Keywords: Disaggregated Datacenter Architecture, Software Defined Environments, Software Defined Networking.

Abstract: The rapid growth of cloud computing both in terms of the spectrum and volume of cloud workloads brought many challenges to the traditional data center design - fast changing system configuration requirements due to workload constraints, varying innovation cycles of system components, and maximal sharing of systems and subsystems. In this paper, we developed a qualitative assessment of the opportunites and challenges for leveraging disaggregated datacenter architecture to address these challenges. In particular, we compare and contrast the programming models that can be used to access the disaggregated resources, and developed the implications for the network and resource provisioning and management.

1 INTRODUCTION

Cloud computing is quickly becoming the fastest growing platform for deploying enterprise, social, mobile, and analytic workloads. As many of these workloads grew to internet scale, they have ushered a new era of datacenter scale computing on top of the previous centralized and distributed computing era (Barroso 2013). During the *centralized computing era*, the computing resources are fully shared (share everything) and centralized managed. The subsequent *distributed computing era* allows decentralized management of distributed resources interconnected by networks. The "*at scale" computing era*, in contrast, involves *de facto* centralized management of massive amount of distributed and often virtualized resources that are locally concentrated within mega-datacenters and often spread across multiple datacenters. Recently, the need for increased agility and flexibility has accelerated the introduction of software defined environments (which include software defined networking, storage, and compute) where the control and management planes of these resources are decoupled from the data planes so that they are no longer vertically integrated as in traditional compute, storage or switch systems and can be deployed anywhere within a datacenter (Li et al., 2014).

The emerging at scale cloud data centers are facing the following challenges: fast changing system configuration requirements due to highly dynamic workload constraints, varying innovation cycles of system components, and the need for maximal sharing of systems and subsystems resources in order to minimize the Capital Expenditure (CAPEX) and Operational Expenditure (OPEX) for efficient at scale operation. These challenges are further elaborated below. Enabling software and platform as a service with optimal stack level cost performance has become a major differentiator in the marketplace, especially for those services involving massive scale out environment such as Hadoop and Spark.

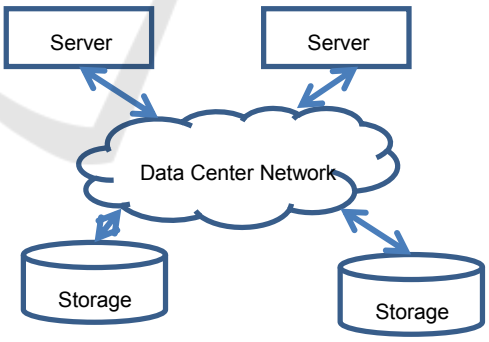

Figure 1: Traditional datacenter with servers and storage interconnected by datacenter networks.

Systems in a cloud computing environment often have to be configured differently in response to different workload requirements. A traditional datacenter, as shown in Fig. 1, includes servers and storage interconnected by datacenter networks. A

Li C., Franke H., Parris C. and Chang V..
Disaggregated Architecture for at Scale Computing.
DOI: 10.5220/0005551300450052
In *Proceedings of the 2nd International Workshop on Emerging Software as a Service and Analytics* (ESaaSA-2015), pages 45-52
ISBN: 978-989-758-110-6

typical server system configured with only CPU and memory while keeping the storage subsystem (which also includes the storage controller and storage cache) remote is likely to be applicable to workloads which do not require large I/O bandwidth and will only need to use the storage occasionally. This configuration is usually inexpensive and versatile - but unlikely to perform well when large I/O bandwidth or small latency become pertinent. Alternatively, the server can be configured with large amount of local memory, SSD, and storage. This configuration, however, is likely to become expensive. Some of the system resources such as the SSDs configured within the server could be potentially underutilized at various times as workloads may not always be able to fully utilize them.

Traditional systems also impose identical lifecycle for every component inside the system. As a result, all of the components within a system (whether it is a server, storage, or switches) are replaced or upgraded at the same time. The "synchronous" nature of replacing the whole system at the same time prevents earlier adoption of newer technology at the component level, whether it is memory, SSD, GPU, or FPGA. The average refresh cycle of CPUs is approximately 3-4 years, HDDs and fans are around 5 years, battery backup (i.e. UPS), RAM, and power supply are around 6 years. Other components in a data center typically have a lifetime of 10 years. Consequently, an integrated system with CPU, memory, GPU, power supply, fan, RAM, HDD, SSD likely has the same lifecycle for everything within the system as replacing these components individually will be uneconomical.

Traditional system resources (memory, storage, and accelerators) configured for high throughput or low latency usually could not share these resources across the data center as they are only accessible within the "system" they are in. As a result, the utilization of the resources could be low. Those resources configured as remote (or network attached) allow maximal shareability but the performance in terms of throughput and latency are usually poor. As an example, challenges in terms of operational efficiency and security in the cloud based financial service domain were reported in Chang (2014), where a pipelined cloud service architecture is implemented on top of a traditional datacenter architecture.

In this paper, we developed a qualitative assessment of the approaches and challenges for leveraging the disaggregated architecture for at scale cloud datacenters. We compare and contrast the programming models that can be used to access the disaggregated resources. We also developed the implications for the network and resource provisioning and management. Based on this qualitative assessment and early experimental results, we concluded that disaggregated architecture with appropriate programming models and resource provisioning is likely to achieve improved datacenter operating efficiency. This architecture is particularly suitable for heterogeneous workload environments as these environments often have dynamic resource requirements and can benefit from the improved elasticity of the physical resource pooling offered by the disaggregated datacentre architecture.

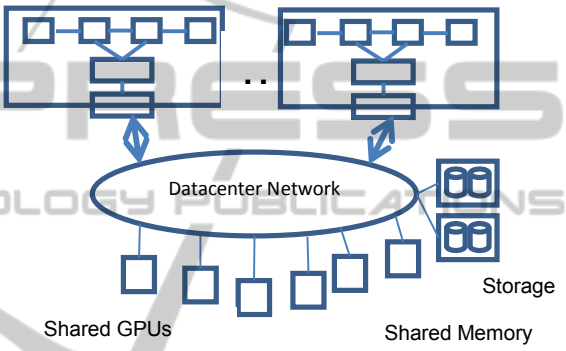

Figure 2: Partially disaggregated datacentre with both fully integrated server /storage as well as disaggregated GPU, SSD, and memory pools.

2 COMPOSABLE SYSTEMS ARCHITECTURE

The emerging disaggregated datacenter architecture or datacenter as a computer (Lim et al, 2009), as shown in Fig. 2, leverages the fast progress of the networking capabilities, software defined environments, and the increasing demand on high utilization of computing resources in order to achieve maximal efficiency.

On the networking front, the emerging trend is to utilize a high throughput low latency network as the “backplane” of the system. Such a system can vary from rack, cluster of racks, PoDs, domains, availability zones, regions, and multiple datacenters. During the past 3 decades, the gap between the backplane technologies (as represented by PCIe) and network technologies (as represented by Ethernet) is quickly shrinking. During the next 5 years, the bandwidth gap between PCIe gen 4 (~250 Gb/s) and 100/400 GbE is becoming much less significant.

When the backplane speed is no longer much faster than the network speed, many interesting opportunities arise for refactoring systems and subsystems as these system components are no longer required to be in the same "box" in order to maintain high system throughput. As the network speeds become comparable to the backplane speeds, SSD and storage which are locally connected through a PCIe bus can now be connected through a high speed network. This configuration allows maximal amount of sharing and maximal amount of flexibility to address the complete spectrum of potential workloads. The broad deployment of Software Defined Environments (SDE) within cloud datacenters is facilitating the disaggregation among the management planes, control planes, and data planes within servers, switches and storage (Li et al, 2014).

Systems and subsystems within a composable disaggregated data center are refactored so that these subsystems can use the network "backplane" to communicate with each other as a single system. Composable system concept has already been successfully applied to the network, storage and server areas. In the networking area, physical switches, routing tables, controllers, operating systems, system management, and applications in traditional switching systems are vertically integrated within the same "box". Increasingly, the newer generation switches logically and physically separate the data planes (hardware switches and routing tables) from the control planes (controller and OS and applications) and management planes (system and network management) and allow the disaggregation of switching systems into these three subsystems where the control and management planes can reside anywhere within a data center, while the data planes serve as the traditional role for switching data. Similar to the networking area, storage systems are taking a similar path. Those monolithically integrated storage systems that include HDDs, controllers, caches (including SSDs), special function accelerators for compression and encryption are transitioning into logically and physically distinct data planes - JBOD (just a bunch of drives), control planes (controllers, caches, SSDs) and management planes.

A partially disaggregated memory architecture was proposed by Lim et al (2009, 2012) in which each disaggregated compute node retains a smaller size of memory while the rest of the memory is aggregated and shared remotely. When a compute node requires more memory to perform a task, the hypervisor integrates the local memory and the remote shared memory to form a flat memory address space for the task. During the run time, accesses to remote addresses result in a hypervisor trap and initiate the transfer of the entire page through RDMA (Remote Direct Memory Access) mechanism to the local memory. Their experimental results show an average of ten times performance benefit in a memory-constrained environment. A detailed study of the impacts of network bandwidth and latency of a disaggregated datacenter for executing in memory workloads such as GraphLab, MemcacheD and Pig was reported in Rumble et al. (2011). When the remote memory is configured to contain 75% of the working set, it was found that the application level degradation was minimal (less than 10%) when network bandwidth is 40 Gb/s and the latency is less than $10\mu s$ (Han et al, 2013). Server products based on a disaggregated architecture have already appeared in the marketplace. These include the Cisco UCS M-Series Modular Server, AMD SeaMicro disaggregated architecture, and Intel Rack Scale Architecture as part of the Open Compute Project.

3 SOFTWARE STACK

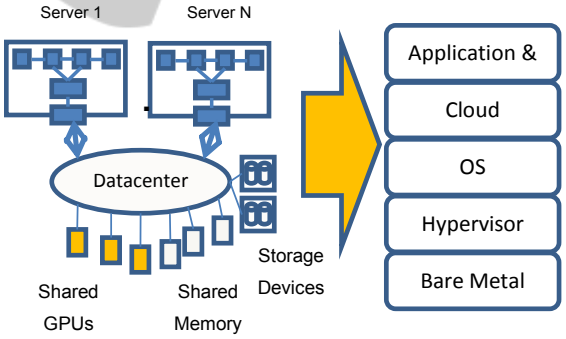

Figure 3: Software stack for accessing disaggregated resources.

Disaggregated datacenter resources can be accessed by application programming models through different means and methods. We consider three fundamental approaches here: (i) hardware based, (ii) hypervisor/operating system based, and (iii) middleware/application based.

The hardware based approach for accessing disaggregated resources is transparent to applications and the OS/hypervisor. Disaggregated memory is mapped to the physical address space of a compute node and is byte addressable across the network. In this case, disaggregated memory is

entirely transparent to the applications. While such transparency is desirable, it forces a tight integration at the memory subsystem either at the physical level or the hypervisor level. At the physical level the memory controller needs to be able to handle remote memory accesses. To avoid the impact of long memory access latencies, we expect that a large cache system is required. Disaggregated GPU and FPGA can be accessed as an I/O device based on direct integration through PCIe over Ethernet. Similar to disaggregated memory, the programming models remain unchanged once the mapping of the disaggregated resource to the I/O address space of the local compute node.

In the second approach, the access of disaggregated resources can be exposed at the hypervisor/container/operating system levels. New hypervisor level primitives - such as getMemory, getGPU, getFPGA, etc. - need to be defined to allow applications to explicitly request the provisioning and management of these resources in a manner similar to *malloc*. It is also possible to modify the paging mechanism within the hypervisor/operatoring systems so that the paging to HDD is now going through a new memory hierarchy including disaggregated memory, SSD, and HDD. In this case, the application does not need to be modified at all. Accessing remote Nvdia GPU through rCUDA (Duato 2010) has been demonstrated, and has been shown to actually outperform locally connected GPU when there is appropriate network connectivity.

Disaggregation details and resource remoteness can also be directly exposed to applications. Disaggregated resources can be exposed via high-level APIs (e.g. put/get for memory). As an example, it is possible to define *GetMemory* in the form of Memory as a Service as one of the Openstack service. The Openstack service sets up a channel between the host and the memory pool service through RDMA. Through *GetMemory* service, the application can now explicitly control which part of its address space is deemed remote and therefore controls or is at least cognizant which memory and application objects will be placed remotely. In the case of GPU as a service, a new service primitive *GetGPU* can be defined to locate an available GPU from a GPU resource pool and host from the host resource pool. The system establishes the channel between the host and the GPU through RDMA/PCIe and exposes the GPU access to applications via a library or a virtual device.

4 NETWORK CONSIDERATIONS

One of the primary challenges for a disaggregated datacenter architecture is the latency incurred by the network when accessing memory, SSD, GPU, and FPGA from remote resource pools. The latency sensitivity depends on how the disaggregated resources are exposed to the programming models in terms of direct hardware, hypervisor, or resource as a service.

The most stringent requirement on the network arises when disaggregated memory is mapped to the address space of the compute node and is accessed through the byte addressable approach. The total access latency across the network cannot be significantly larger than the typical access time of DRAM – which is on the order of 75 ns. As a result, silicon photonics and optical circuit switches (OCS) are likely to be the only options to enable memory disaggregation beyond a rack. Large caches can reduce the impact of remote access. When the block sizes are aligned with the page sizes of the system, the remote memory can be managed as extension of the virtual memory system of the local hosts through the hypervisor and OS management. In this configuration, local DRAM is used as a cache for the remote memory, which is managed in page-size blocks and can be moved via RDMA operations.

Disaggregating GPU and FPGA are much less demanding as each GPU and FPGA are likely to have its local memory, and will often engage in computations that last many microseconds or milliseconds. So the predominant communication mode between a compute node and disaggregated GPU and FPGA resources is likely through bulk data transfer. It has been shown by Reano et al. (2013) that adequate bandwidth such as those offered by RDMA at FDR data rate (56 Gb/s) already demonstrated superior performance than a locally connected GPU.

Current SSD technologies has a spectrum of 100K IOPS (or more) and ~100 us access latency. Consequently, the access latency for non-buffered SSD should be on the order of 10 us. This latency may be achievable using conventional Top-of-the-Rack (TOR) switch technologies if the communication is limited to within a rack. A flat network across a PoD or a datacenter with a two-tier spine-leaf model or a single tier spline model is required in order achieve less than 10 us latency if the communication between the local hosts and the disaggregated SSD resource pools are across a PoD or a datacenter.

5 DISTRIBUTED RESOURCE PROVISIONING

In a disaggregated datacenter with physical resource pooling, it is essential that the physical resources are requested and provisioned with minimum latency so that the use of remote resources will not create a serious performance bottleneck. In this section, we will describe an approach based on distributed scheduling with global shared state in conjunction with predictive resource provisioning.

Resource provisioning and scheduling can be carried out through a centralized, hierarchical, or fully distributed approach. The centralized approach is likely to achieve the optimal resource utilization, but may result in a single point of failure and a severe performance bottleneck. The hierarchical approach, such as the one used in Mesos (Hindman, 2011), allows flexible addition of heterogeneous schedulers for different classes of workloads to a centralized scheduler. The centralized scheduler allocates chunks of resources to the workload specific scheduler, which in turn allocates resources to individual tasks. However, this approach often results in sub-optimal utilization. A fully distributed approach with global shared state, such as the Google Omega (Schwarzkopf, 2013) project, utilizes an optimistic approach for resource scheduling. This approach is likely to perform better as compared to other approaches.

The mechanism for scheduling and provisioning resources from disaggregated physical resource pools starts with the requesting node establishing the type and amount of resource required. As discussed in the previous section, the amount of resource required can be established explicitly by the workload or implicitly as the current requesting node runs out of resource locally. Once the request is received, the resource provisioning engine will identify one or more of the resource pools with available resources, potentially based on the global shared state, for provisioning resources. It will then communicate with the resource manager of the corresponding resource pool to reserve the actual resource. The resource manager for each resource pool commits the resource to the incoming request and resolves the potential conflicts if multiple requests for the same resource occur simultaneously. Once the resource is reserved, the communication between the requesting node and the resource can then commence.

Due to the low latency requirement for provisioning physical resources in a disaggregated datacenter, it is likely that the resources will need to be provisioned and reserved before the actual needs from the workload arise rather than on demand. This may require the resource scheduler to monitor the history of the resource usage so that an accurate workload dependent projection of the resource usage can always be maintained.

6 EXPERIMENTAL RESULTS

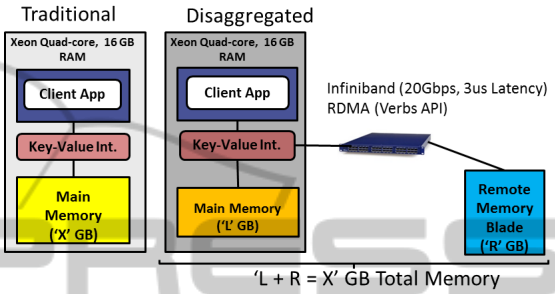

Figure 4: Experimental setup for performance measurement in a disaggregated environment for MemcacheD.

In this section, we describe experiments that demonstrate the workload behavior when a cloud centric application such as MemcacheD is deployed in a disaggregated system environment. In this case, part of client app data is in local DRAM, while the rest is located in the memory of a remote node accessed through an RDMA capable fabric via Verbs API.

The disaggregated infrastructure, as shown in Fig. 4, is entirely transparent to the MemcacheD client. The server side is modified so that the data accessed via key-value interface will be automatically retrieved from either local or remote memory.

The modification is as follows: A small program on another machine allocates a specified amount of memory and registers the allocation with the Infiniband HCA. MemcacheD handshakes with the remote server and obtains the pertinent information such as remote buffer address and access_key. After an initial handshake, it can now perform RDMA reads and writes directly to the remote buffer. The remote buffer is treated as a "victim cache" and is maintained as an append-only log. When MemcacheD runs out of local memory, instead of evicting a key/value pair in the local memory, it now does an RDMA write to the remote memory. When looking up a particular key, it first checks with the local memory (via a hash table). If the key does not exist locally, MemcacheD checks

the remote memory via a locally maintained hash table. If key/value is in the remote memory, it reads in this value through RDMA to a temporary local buffer and sends it to the client. A particular key/value is always either in local memory or remote memory and can never resides in both locations.

The experiments consist of 100,000 operations (95% reads, 5% updates) with uniform random accesses (i.e. no notion of working set as this represents the most challenging situation) running in a single thread.

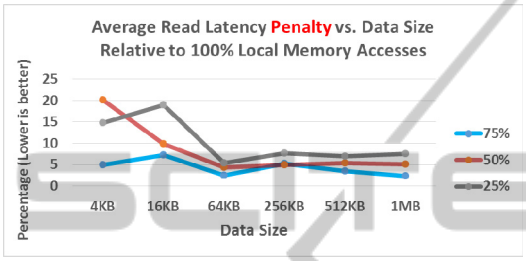

Figure 5: Average read latency penalty vs. data size with respect to 100% local access when the local portion of data varies from 75% to 25%

As shown in Fig. 5, higher percentage of local data always introduces fewer penalties. However, the difference begins to diminish among different ratio of local vs. remote data when the data block size is larger than 64 KB, as larger block size reduces the overhead in the data transfer.

The second set of experiments consist of 100,000 read and update operations (95% reads, 5% updates) with uniform random accesses (i.e. no notion of working set as this represents the most challenging situation) evenly split among 10 threads.

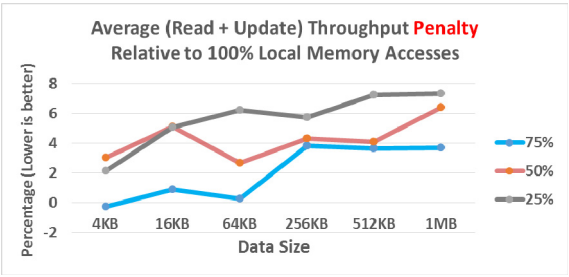

Figure 6: Average read/update throughput penalty vs. data size with respect to 100% local access when the local portion of data varies from 75% to 25%.

As shown in Fig. 6, the throughput penalty is nearly nonexistent when 75% of the access is local and the data size is 4KB. The penalty increases to 2% when only 25% of the access is local. As the data sizes increases, the transfer time of the entire page between the local and the remote node increases, resulting in higher penalty at 4% and 6%, respectively, for 75% and 25% local access.

We can conclude from these experiments that negligible latency and throughput penalty are incurred for the read/update operations if these operations are 75% local and the data size is 64 KB. Smaller data size results in larger latency penalty while larger data size results in larger throughput penalty when the ratio of nonlocal operations is increased to 50% and 75%.

In a second experiment we examine the popular graph analytics platform Giraph, that enables implementation of distributed graph algorithms. In this particular case we populated a 50 node virtual compute cluster with a randomly generated graph of 100 million vertices. The graph is partitioned into 50^2 partitions which are distributed across the compute nodes. We then compute the TopKPagerank properties of the graph. As the computation progresses, messages need to be exchanged to traverse the graph as it crosses node boundaries. Dependent on the connectivity of the graph, the variance in the message creation can result in substantial different memory consumptions per node. Under memory pressure, Giraph will swap entire partitions and messages per vertex to disk using LRU. We examine the memory utilization across the nodes as computations progresses. While cpu utilization is very uniform across all nodes and across the execution of the program, memory utilization varies considerable, which is shown as a heatmap in Fig. 7.

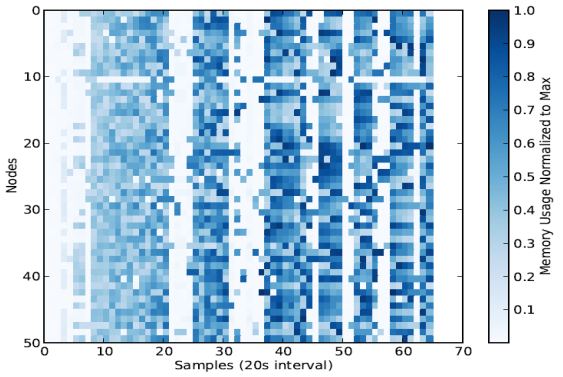

Figure 7: Memory Consumption of Distributed Giraph TopKPagerank application over time.

Analysis of this data reveals that the peak per node memory usage versus the average per node memory has a 2.78:1 ratio, where the aggregate memory usage has a 1.68:1 ratio. We then reduce the per

node memory by a factor of 3 to explore the impact of memory pressure, while the average per node memory is maintained. This increases the overall runtime of the experiment by a factor of 13.8x highlighting that planning resource consumption for best performance requires a memory overprovisioning of a factor of three or alternatively to pay a substantial performance penalty. When the swap disk is on each node is configured to a RamDisk, the overhead reduces to a factor of 6.14x - which is still too high. Having observed the low overheads of RDMA in the MemCacheD example, we stipulate that sharing unused memory across the entire compute cluster instead of through a swap device to a remote memory location can further reduce the overhead. However the rapid allocation and deallocation of remote memory is imperative to be effective.

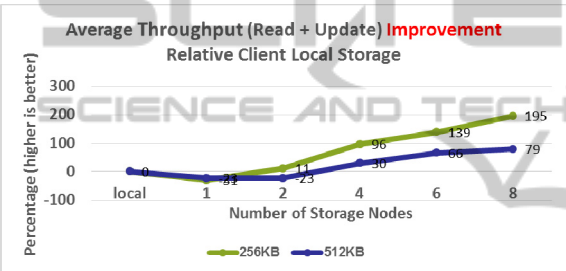

Figure 8: Throughput improvement of disaggregated storage for Cassandra workload.

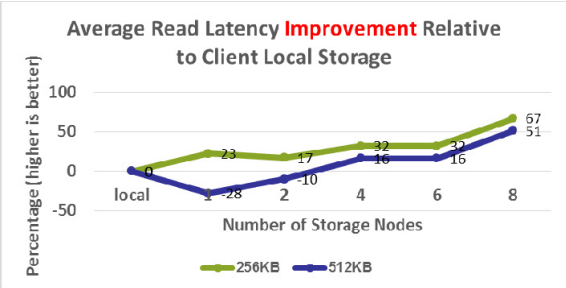

Figure 9: Latency improvement of disaggregated storage for Cassandra workload.

In our final experiment we examine the impact of disaggregated storage. We utilized Cassandra, a popular persistent, i.e disk backed, key value store. In the traditional setup a single server is populated with eight SATA disks that together form the block storage for a ZFS filesystem on which the key value pair storage resides. Ultimately the number of disks in the server is limited to the order of 10s and the SATA v3 bandwidth is limited to 6Gbps. In the disaggregated setup we utilize 4 storage nodes with eight disks each and access to Cassandra was over a 10Gbps Ethernet network. The ZFS cache was limited and data was flushed out of the page cache to ensure that almost all accesses go to disk. A client consisting of 20 threads issued 10K operations (95% read) uniformly accessing the data domain. The bandwidth and latency improvement are shown in Figures 8 and 9. Accessing blocks size of 256KB and 512KB, we observed throughput improvements of up to 195% and 79 % respectively for the disaggregated system case. And latency improvement was 67% and 51%. This experiment substantiates our thesis that accessing data from across multiple disks connected via Ethernet poses less of a bandwidth restriction than SATA and thus improves throughput and latency of data access and obviates the need for data locality. Overall disaggregated storage systems are cheaper to build, manage and incrementally scale and offer higher performance than traditional setups.

As more operations are moved to the shared physical resource pools, it is conceivable that the utilization of shared physical resources will improve, resulting in reduced Capex and/or Opex.

7 CONCLUSIONS

The rapid growth of cloud computing workloads both in terms of the spectrum and volume brought many challenges to the traditional data center design: (1) Fast changing system configuration requirements due to workload constraints; (2) Varying innovation cycles of system components; (3) Maximal sharing of systems and subsystems in order achieve optimal efficiency. The disaggregated architecture provides a promising approach to address these simultaneous challenges. Datacenters based on this architecture allows the refactoring of the datacenter for improved operating efficiency and decoupled innovation cycles among components while the datacenter network becomes the "backplane"of the datacenter.

In this paper, we developed a qualitative assessment of the approaches and challenges for leveraging disaggregated architecture for at scale cloud datacenters. In particular, we compare and contrast the programming models that can be used to access the disaggregated resources, the implications for the network and resource provisioning and management.. Based on this qualitative assessment and early experimental results, we concluded that disaggregated architecture with appropriate programming models and resource provisioning is likely to achieve improved datacentre operating

efficiency for heterogeneous workload environments that can benefit from the improved elasticity of physical resources.

ACKNOWLEDGEMENTS

The authors are grateful for the Dr. Mukil Kesavan for performing the experiments described in Section 6.

REFERENCES

Barroso, Luiz André, Jimmy Clidaras and Urs Hölzle. (2013) The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second edition.

Chang, V. (2014). The business intelligence as a service in the cloud. Future Generation Computer Systems, 37, 512-534.

Duato, J., Pena, A. J., Silla, F., Mayo, R., & Quintana-Ortí, E. S. (2010, June). rCUDA: Reducing the number of GPU-based accelerators in high performance clusters. In High Performance Computing and Simulation (HPCS), 2010 International Conference on (pp. 224-231). IEEE.

Han, S., N. Egi, A. Panda, S. Ratnasamy, G. Shi, and S. Shenker, (2013) Network support for resource disaggregation in next-generation datacenters. In Proc. HotNets.

Herodotou, H., F. Dong and S. Babu. No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics. In Proc. of the 2nd ACM Symposium on Cloud Computing, 2011.

Hindman B., A. Konwinski, M. Zaharia, A. Ghodsi, A. D Joseph, R. H Katz, S. Shenker, I. StoicaMesos (2011): Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. Proc. ACM USENIX Symposium on Networked Systems Design & Implementation (NSDI), 2011.

Krug, Perry, How Many Nodes? Part 1: An introduction to sizing a Couchbase Server 2.0 cluster. http:// blog.couchbase.com/how-many-nodes-part-1-introduction-sizing-couchbase-server-20-cluster.

Li, C.-S, B. L. Brech, S. Crowder, D. M. Dias, H. Franke, M. Hogstrom, D. Lindquist, G. Pacifici, S. Pappe, B. Rajaraman, J. Rao, R. P. Ratnaparkhi, R. A. Smith and M. D. Williams. (2014) Software defined environments: An introduction. In IBM Journal of Research and Development Vol. 58 No. 2/3 pp. 1-11, March/May.

Lim, K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt and T. F. Wenisch. (2009) Disaggregated Memory for Expansion and Sharing in Blade Servers. In Proc. ISCA.

Lim, K., Y. Turner, J. R. Santos, A. AuYoung, J. Chang, P. Ranganathan and T. F. Wenisch. (2012) System-level implications of disaggregated memory. In Proc. HPCA.

Reano, C., R. May, E. S. Quintana-Orti, F. Silla, J. Duato, A. J. Pena (2013), Influence of InfiniBand FDR on the Performance of Remote GPU Virtualization, IEEE International Conference on Cluster Computing (CLUSTER), pp. 1-8.

Rumble, S. Rumble, D. Ongaro, R. Stutsman, M. Rosenblum, and J. Ousterhout. (2011) It's time for low latency. In Proc. HotOS.

Schwarzkopf, M., Konwinski, A., Abd-El-Malek, M., & Wilkes, J. (2013, April). Omega: flexible, scalable schedulers for large compute clusters. In Proceedings of the 8th ACM European Conference on Computer Systems (pp. 351-364). ACM.

GraphLab. http://graphlab.com/

Memcached - a distributed memory object caching system. http://memcached.org/

PigMix benchmark tool.http://cwiki.apache.org/ confluence/display/PIG/PigMix.

Cisco UCS M-Series Modular Servers. http://www.cisco.com/c/en/us/products/servers-unified-computing/ucs-m-series-modular-servers/index.html.

AMD Disaggregates the Server, Defines New Hyperscale Building Block. http://www.seamicro.com/sites/ default/files/MoorInsights.pdf.

SeaMicro Technology Overview. http://seamicro.com/ sites/default/files/SM_TO01_64_v2.5.pdf.

Intel, Facebook Collaborate on Future Datacenter Rack Technologies, http://newsroom.intel.com/community/ intel_newsroom/blog/2013/01/16/intel-facebook-collaborate-on-future-data-center-rack-technologies, Jan. 2013.

Open Compute Project. http://www.opencompute.org.