Accelerating Virtualized & Distributed Cassandra databases with FPGAs

Introduction

Field Programmable Gate Arrays (FPGAs) as accelerators for data center workloads are beginning to cross the chasm of broader adoption. FPGAs have been around for more than twenty-five years and have successfully accelerated IO-centric applications such as network routers and storage controllers. FPGAs are reconfigurable hardware that offer software-like flexibility while delivering hardware-like performance using spatial computing techniques, leveraging parallel computational units with custom interconnections. Their use as accelerators for data center workloads continues to gain momentum. This trend is primarily driven by a recurring necessity of energy-efficient data center infrastructure and IO-centric workloads such as databases and compute intensive workloads such as inference for AI. Now, solutions that combine hardware like FPGAs with advanced software can provide extreme performance improvements for existing software like open source databases.

Cassandra Database:

Apache Cassandra is a free and open-source, distributed, wide column store, NoSQL database management system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure. Cassandra offers robust support for clusters spanning multiple datacenters, with asynchronous master-less replication allowing low latency operations for all clients. (Source: Wikipedia)

Cassandra Read performance

Cassandra’s read performance can benefit from built-in caching. Read performance when using Cassandra gets decreased due to certain operations like compaction runs, consistency level, read repair chance, etc. Read performance can be improved by increasing the replication factor, but it can make the cache less efficient. If you want to decrease read latency, you can use a lower consistency level but it does that at the cost of consistency.
Due to the challenges relating to read performance in Cassandra, there is a need to improve caching and potentially reduces the number of nodes. rENIAC's Data Engine (rDE) is an FPGA-based accelerator for Cassandra that acts as a proxy and cache for existing Apache Cassandra or DataStax DB nodes. rENIAC's Data Engine does all the heavy lifting required to add a flash-based cache acceleration to your Cassandra DB, without requiring application developers to modify applications (e.g. continue using standard Cassandra query language API), manage cache invalidation, data population, or cluster management. Thus rDE

frees up cycles for application business logic (i.e. the revenue generating applications) while enabling significantly higher performance.

rENIAC Data Engine

rENIAC Data Engine (rDE) is an FPGA-based database accelerator that acts as a transparent proxy or a caching tier. It sits between a database client and database node, caching the data in (Flash) storage that is accessible by the FPGA. It responds to queries by serving data either from its local storage or fetching it from the backend database when the data does not exist in the local storage. This ensures that read requests are satisfied with predictably low latency and allows rDE to achieve throughputs much higher than a standard database cluster.

rENIAC Data Engine has been designed to work without requiring any changes to the client code or the database, and with minimal configuration.

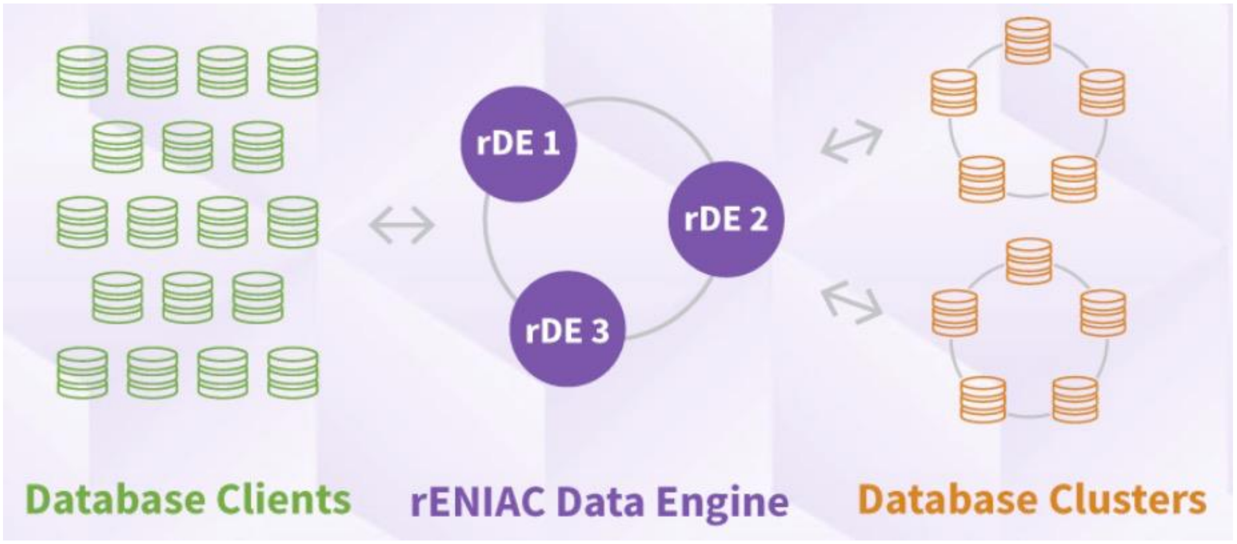


Figure 1: RDE Deployed as Data Proxy for Cassandra Databases - Conceptual Architecture

The rDE nodes listen for incoming queries on the configured port. For read queries, the rDE parses the query and looks for the data in the local storage. If found, it returns the result to the client. If not found, it obtains the data from the database cluster, stores a copy in the local storage and returns the result to the client.

In the current version of the product, for insert, update and delete operations, the proxy forwards the query to the database cluster, invalidating the data stored in its own cache. When the database has successfully processed the query, the proxy forwards the response to the client.

Challenges with read performance with Cassandra Databases:

Modern databases are built to use commodity hardware efficiently, specifically Input/Output (I/O) and processing components. They are designed to rely on horizontal scaling to scale (query) throughput while ensuring commonality in all of the nodes in the cluster. One of the real core functions that a database is intended to execute is read servicing - but - not all databases are optimized to do so. One such example is Apache Cassandra, one of the most popular open source databases which is used for its ability to scale, consistency and availability. Cassandra does a great job of handling write queries and is widely used at leading organizations like Netflix, Apple and Walmart.

Large scale data-centric environments often have applications with read/write ratios ranging anywhere from 5:1 to 500:1, exacerbating the read inefficiency, and often, creating the need for a cache layer to service reads at a latency that fits within service level agreement (SLA) windows. An in-memory or cache tier is a good solution for many organizations, but it can be difficult to maintain scale and cost when architecting for applications like AI or ML.

In an architecture where the database is accessing Flash memory and database logic is a crucial operation, significant CPU resources are required just to keep up with a fast Flash storage drive - together, in a read-heavy workload, Storage and Network IO can consume 60% of CPU usage.

CPU cycles are also spent on “plumbing” operations in open source databases for typical write-heavy workloads. Modern databases rely on computationally complex functions such as compression, encryption, and compaction which add to computational load. It has been well established that both compression and encryption are computationally expensive and even prohibitive for CPUs to carry out these functions. For a write heavy workload, compaction and compression/decompression together can account for almost 65% of the total CPU cycles.

These inherent inefficiencies mean a lot of CPU cores are required to handle some basic functions, particularly as data scales. Data-centric database workloads, no matter how efficiently coded, perform badly on traditional compute-optimized environments.

Virtualizing rENIAC and Cassandra:

In this solution all components of the infrastructure are virtualized on vSphere.   rENIAC was deployed on a vSphere host with a physical Intel FPGA card installed. The rENIAC virtual machine is configured with direct passthrough access to the FPGA. The database and the client virtual machines are deployed as standard virtual machines with appropriate sizing for CPU, Memory and disk.

Configuration

The schematic shows the architecture that includes a three node Cassandra Database cluster, a three node rENIAC cluster and two database clients. The configuration of the solution is shown below. All components are virtualized and rENIAC is deployed as a proxy between the Cassandra databases and its clients.

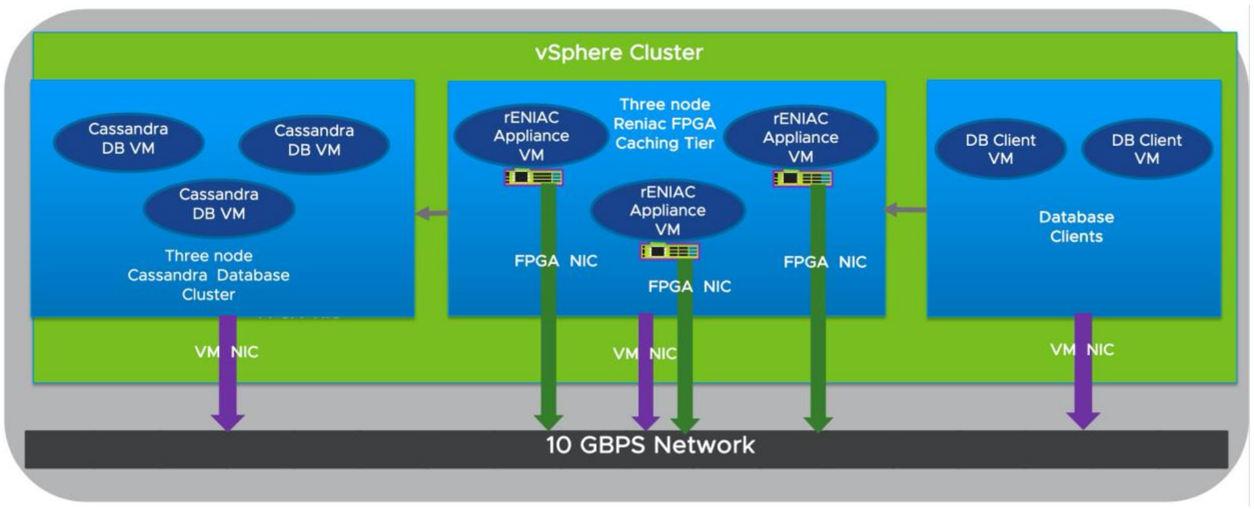


Figure 2: Logical Schematic of Virtualized rDS & Cassandra

OS and Software Requirements for FPGA Virtual Machine

rENIAC Data Engine has been tested to work with CentOS 7.5 or later, with a minimum Linux kernel version of 3.10.

OS	CentOS 7.5 or later Linux kernel 3.10 or later
vSphere	7.0
FPGA	Intel PAC with Intel Arria 10 GX FPGA
Python	v2.7
Apache Cassandra	v3.11.0 or later CQL v3.4 or later
CPU	16 cores
RAM	64 GB
Storage	1 TB Pass Through NVMe storage
Networking	10 Gbps

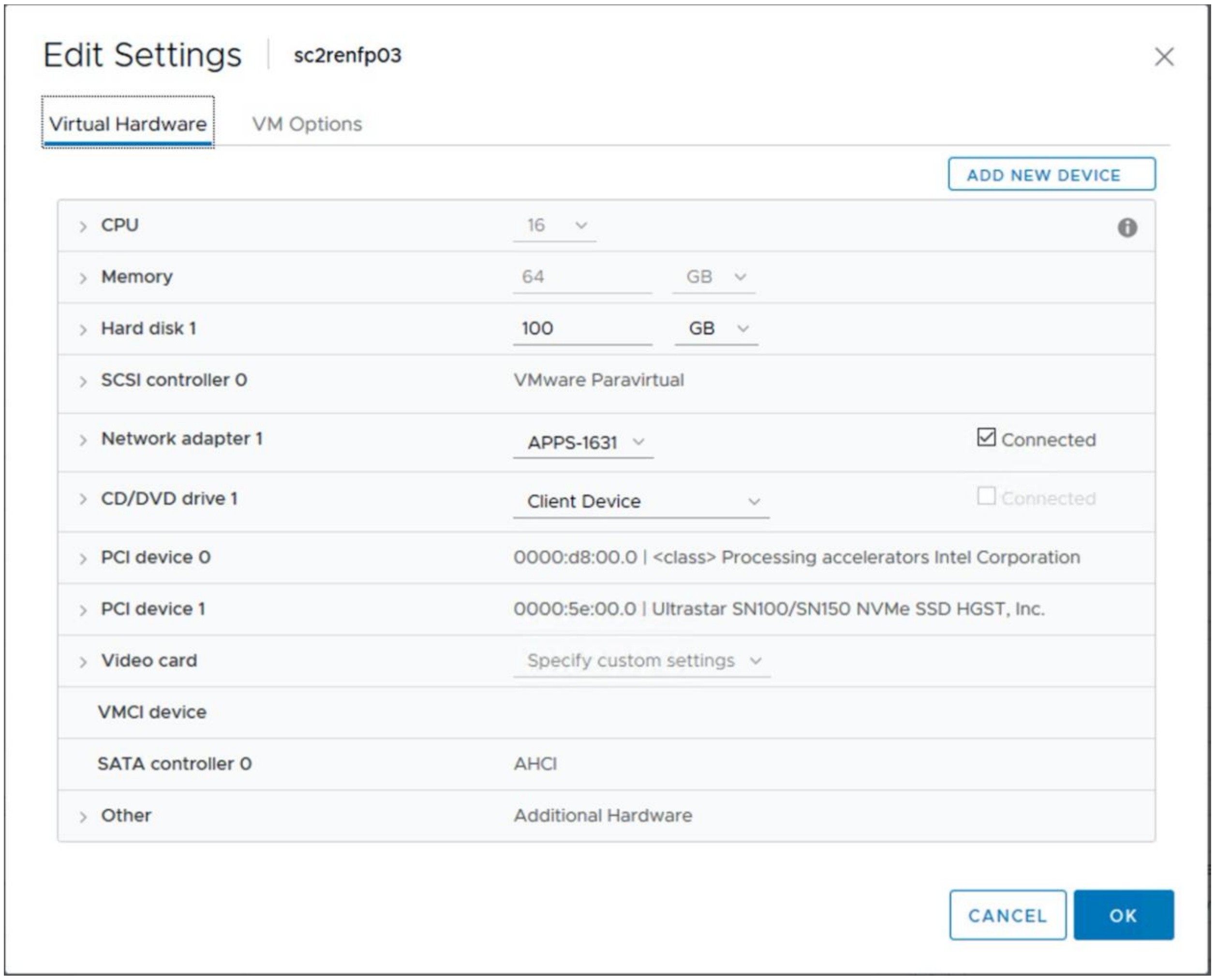
Table 1: FPGA Virtual Machine Specifications

Deployment:

A Cassandra database server cluster was set up with three database servers running on Centos 7.x Linux based virtual machines. Two database clients with Cassandra stress test utilities were also set up.

The following steps were used to deploy the rENIAC Data Engine.

1. The components required to run rDE are installed.
2. The virtual machines have the Intel Arria 10 FPGA card setup in passthrough mode along with a PCIe NVMe device for storage. Figure 3 shows the configuration of one of the rDE devices with the two passthrough devices.



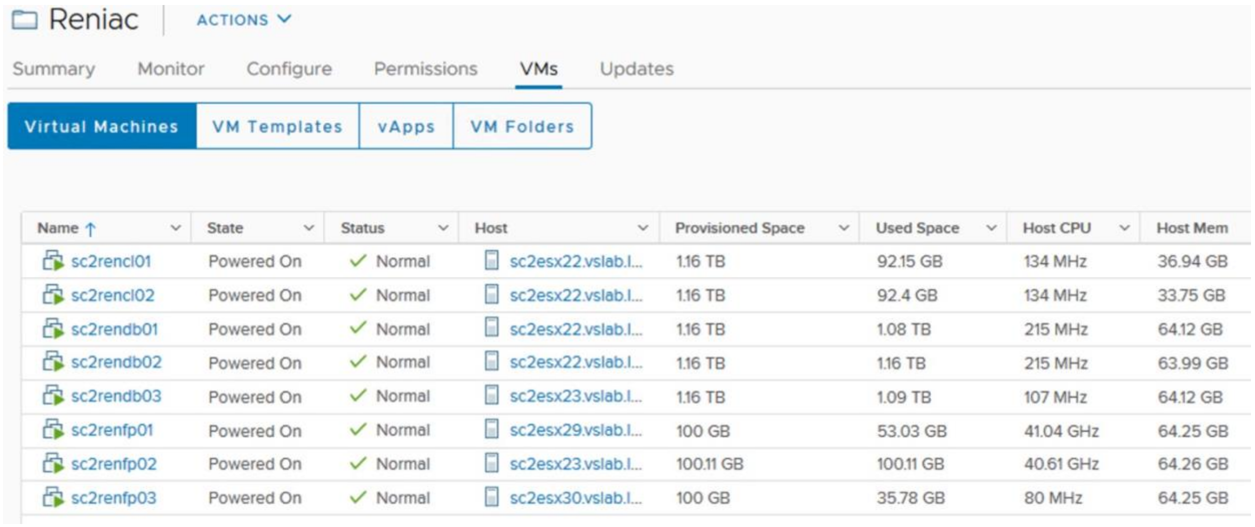
CPU	16		
Memory	64	GB	
Hard disk 1	100	GB	
SCSI controller 0	VMware Paravirtual		
Network adapter 1	APPS-1631		✓ Connected
CD/DVD drive 1	Client Device		☐ Connected
PCI device 0	0000:d8:00.0 | <class> Processing accelerators Intel Corporation		
PCI device 1	0000:5e:00.0 | Ultrastar SN100/SN150 NVMe SSD HGST, Inc.		
Video card	Specify custom settings		
VMCI device			
SATA controller 0	AHCI		
Other	Additional Hardware		

Figure 3: rDE virtual machine settings with passthrough devices

3. The Cassandra database cluster IP is used in the setup of the rDE.
4. rDE is started by running the setup script. This script will flash the FPGA card and start the required software services on each of the three rDE nodes

5. The network address used by the clients for the database cluster are changed to point to the cluster IP of the rDE cluster instead of the Cassandra cluster IP address.

All components of this solution were setup in a vSphere 7 environment and used for the testing. The components of the solution are shown in Figure 4.



Name ↑	State	Status	Host	Provisioned Space	Used Space	Host CPU	Host Mem
sc2rencl01	Powered On	✓ Normal	sc2esx22.vslab.l...	1.16 TB	92.15 GB	134 MHz	36.94 GB
sc2rencl02	Powered On	✓ Normal	sc2esx22.vslab.l...	1.16 TB	92.4 GB	134 MHz	33.75 GB
sc2rendb01	Powered On	✓ Normal	sc2esx22.vslab.l...	1.16 TB	1.08 TB	215 MHz	64.12 GB
sc2rendb02	Powered On	✓ Normal	sc2esx22.vslab.l...	1.16 TB	1.16 TB	215 MHz	63.99 GB
sc2rendb03	Powered On	✓ Normal	sc2esx23.vslab.l...	1.16 TB	1.09 TB	107 MHz	64.12 GB
sc2renfp01	Powered On	✓ Normal	sc2esx29.vslab.l...	100 GB	53.03 GB	41.04 GHz	64.25 GB
sc2renfp02	Powered On	✓ Normal	sc2esx23.vslab.l...	100.11 GB	100.11 GB	40.61 GHz	64.26 GB
sc2renfp03	Powered On	✓ Normal	sc2esx30.vslab.l...	100 GB	35.78 GB	80 MHz	64.25 GB

Figure 4: Virtual Machines representing the compute nodes in the rENIAC acceleration solution

Testing:

The testing run for the benchmark data was done to demonstrate how a solution like rENIAC Data Engine can be dropped into a virtualized environment and accelerate read processing by up to 20x. rENIAC Data Engine encapsulates software-hardware optimization and is deployed on standard CPU based servers with no required changes to application software architecture.

rENIAC’s Approach to Solve the Read Problem:

If the performance of databases is being negatively affected by the traditional CPU ineffectiveness in dealing with the most common and the most user-perceptible task (read); why not relegate that task to something much efficient at executing it?

rENIAC Data Engine tunes the read inefficiency inherent to open source databases to increase overall throughput and significantly reduce latency. One method of doing so is through the use of commercially available FPGAs, like those available from Intel, and standard servers, like those available from Dell, that are programmed to provide hardware assist for many of the functions associated with servicing read requests. It turns out that such an optimized platform can result in performance 20x times more efficient than on virtualized machines alone. A small number of rENIAC nodes can handle a volume of requests that may have required hundreds of standard database nodes running on VMs to handle; all while delivering dramatically lower and more deterministic latency.

Results:

The performance benefits offered by using rDE are measured by comparing the performance of running queries directly on the Cassandra database and running queries through rDE. Using the cassandra-stress tool, we see that the query throughput with rDE is approximately three times higher than the baseline. We also see that the latency values are in a much narrower range.

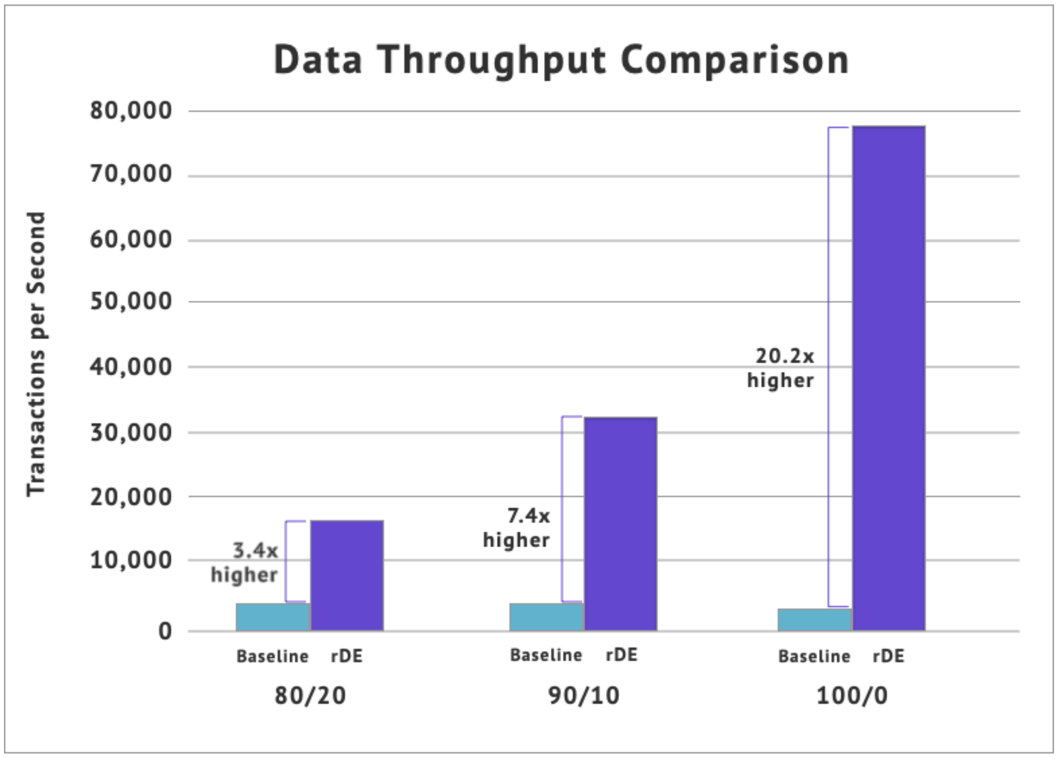


Figure 5: Transactions per second for rDE compared to direct access to Cassandra

Tests were executed with the cassandra-stress utility. The baseline tests were performed directly against the Cassandra database and the performance were measured. The tests were then repeated with rENIAC Data Engine as a proxy layer.

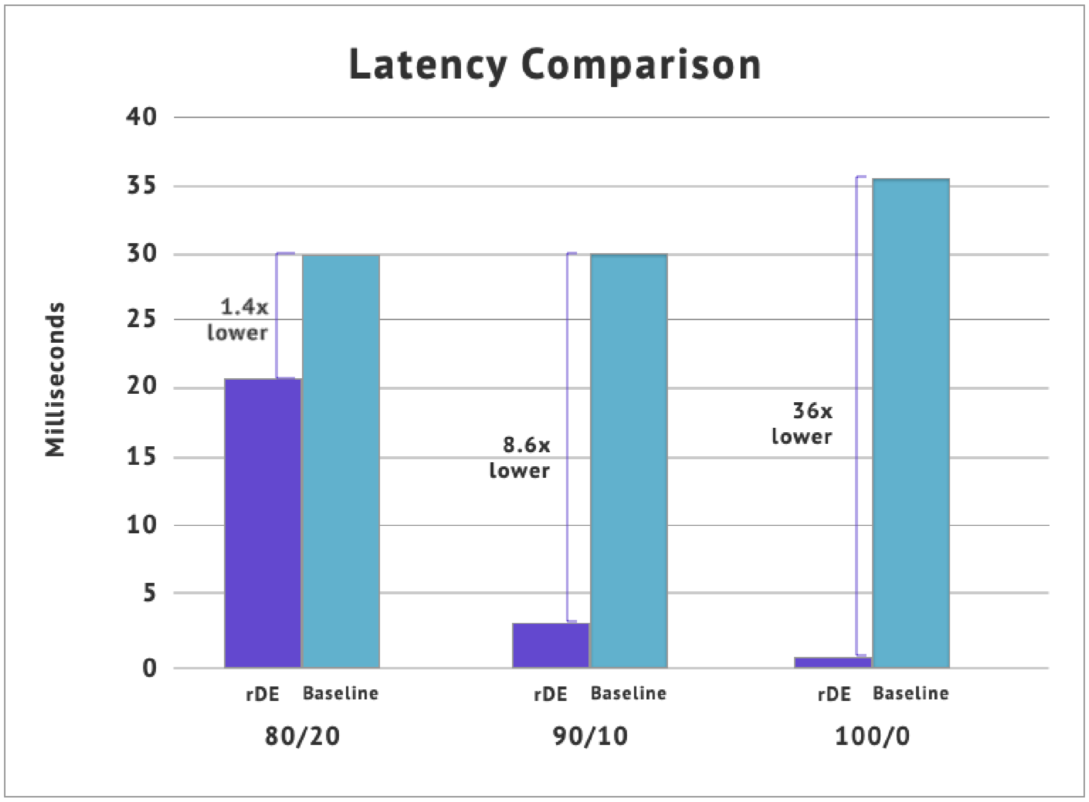


Figure 6: Millisecond Latency at p95 for rDE compared to direct access to Cassandra

We see that the performance is greatly enhanced through the rDE solution . We see that the transactions per second are 20x faster for 100% reads, 7.4x faster for 90% reads and 3.4x times faster for 80% reads.

Similarly, latency is greatly reduced through the rDE solution. Latency is 36x lower for 100% reads, 8.6x lower for 90% reads and 1.4x faster for 80% reads. These results show that massive performance for both throughput and latency reduction can be realized by leveraging FPGAs to proxy access to Cassandra databases.

Conclusion:

Cassandra databases natively need major changes in infrastructure to improve read performance. A caching tier or proxy-based acceleration layer like that provided by rENIAC Data Engine (rDE) can help improve read performance drastically without any changes to the underlying Cassandra database infrastructure. In this solution, we leveraged vSphere infrastructure and its support for FPGAs to host rDE and all Cassandra components. Our tests have shown that rDE accelerates performance 3.4-20x by leveraging rENIAC and FPGAs. vSphere combined with rENIAC allows for a flexible deployment for small to large implementations that will perform at scale. Running rDE on vSphere adds all the flexibility, agility and enterprise capabilities of the platform and can help massively improve read performance of Cassandra databases.