ScaleArc Performance Evaluation with SQL Server 2014

Performance Evaluation: Executive Summary

- Testing goals:
  - To evaluate and visualize the performance of both ScaleArc and SQL Server 2014 for OLTP workloads
  - To show ScaleArc's impact on database workload latency and performance

- Benchmark testing measured:
  - Baseline performance with SQL Server
    - Directly to database – on disk and in memory
    - Via ScaleArc – with no caching
    - Via ScaleArc – with SQL query caching
  - Latency added by ScaleArc's load balancing software
  - Performance improvements by caching SQL queries in ScaleArc

ScaleArc

Benchmark Setup

Client (1)

CPU: 2 x Intel Xeon E5-2620 v2 @ 2.10GHz (6 cores per socket, multi-threading off)

Memory: 64GB

Disk: PCIe NVMe 300K IOPS

ScaleArc (1)

CPU: 2 x Intel Xeon E5-2620 v2 @ 2.10GHz (6 cores per socket, multi-threading off)

Memory: 64GB

Disk: PCIe NVMe 300K IOPS

OS: ScaleArc for SQL Server, v3.10

DB Server (1)

CPU: 2 x Intel Xeon E5-2620 v2 @ 2.10GHz (6 cores per socket, multi-threading off)

Memory: 64GB

Disk: PCIe NVMe 300K IOPS

OS: Windows Server 2012 Enterprise

Database: SQL Server 2014 Enterprise

ScaleArc

Workload for Benchmark : Creation

- All the measurements were performed with a very small database that fits entirely in memory
- The following command was used to create the database:

```
CREATE DATABASE [oltp_performance]
CONTAINMENT = NONE
ON  PRIMARY
( NAME = N'oltp', FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL12.MSSQLSERVER\MSSQL\DATA\oltp_performance.mdf' , SIZE = 178176KB , MAXSIZE = UNLIMITED, FILEGROWTH = 1024KB ),
FILEGROUP [imoltp_performance_mod] CONTAINS MEMORY_OPTIMIZED_DATA DEFAULT
( NAME = N'imoltp_performance_mod1', FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL12.MSSQLSERVER\MSSQL\DATA\imoltp_performance_mod1' , MAXSIZE = UNLIMITED)
LOG ON
( NAME = N'oltp_performance_log', FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL12.MSSQLSERVER\MSSQL\DATA\oltp_performance_log.ldf' , SIZE = 1475904KB , MAXSIZE = 2048GB , FILEGROWTH = 10%)
```

- The following commands were used to create 64 tables each for disk-based and in-memory tables. Each table was then populated with 10,000 rows.

```
CREATE TABLE [dbo].[oltp1](
        [id] [int] NOT NULL,
        [k] [int] NOT NULL CONSTRAINT [DF_oltp1_k]  DEFAULT ((0)),
        [c] [varchar](120) NOT NULL,
        [pad] [varchar](60) NOT NULL,
 CONSTRAINT [PK_oltp1] PRIMARY KEY CLUSTERED
(       [id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

CREATE TABLE [dbo].[oltpmem1]
(       [id] [int] NOT NULL,
        [k] [int] NOT NULL,
        [c] [varchar](120) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
        [pad] [varchar](60) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
 PRIMARY KEY NONCLUSTERED
(
        [id] ASC
)
)WITH ( MEMORY_OPTIMIZED = ON , DURABILITY = SCHEMA_AND_DATA )
```

ScaleArc

Workload for Benchmark: Benchmarking

SELECT c FROM <table_name> WHERE id=<random_id>

SELECT c FROM <table_name> WHERE id=<random_id>

SELECT c FROM <table_name> WHERE id=<random_id>

SELECT c FROM <table_name> WHERE id=<random_id>

SELECT c FROM <table_name> WHERE id=<random_id>

SELECT c FROM <table_name> WHERE id=<random_id>

SELECT c FROM <table_name> WHERE id=<random_id>

SELECT c FROM <table_name> WHERE id=<random_id>

SELECT c FROM <table_name> WHERE id=<random_id>

SELECT c FROM <table_name> WHERE id=<random_id>

SELECT c FROM <table_name> WHERE id=<random_id>

SELECT c FROM <table_name> WHERE id BETWEEN <random_id_1> AND (<random_id_1> +10 - 1)

SELECT SUM(K) FROM <table_name> WHERE id BETWEEN <random_id_2> AND (<random_id_2> + 10 - 1)

SELECT c FROM <table_name> WHERE id BETWEEN <random_id_3> AND (<random_id_3> + 10 - 1) ORDER BY c

SELECT DISTINCT c FROM <table_name> WHERE id BETWEEN <random_id_4> AND (<random_id_4> + 10 - 1) ORDER BY c

SELECT DISTINCT TOP 10 c FROM <table_name> WHERE id > <random_id> ORDER BY c

UPDATE <table_name> SET k=k+1 WHERE id=<random_id>

UPDATE <table_name> SET c='<random_string>' WHERE id=<random_id>

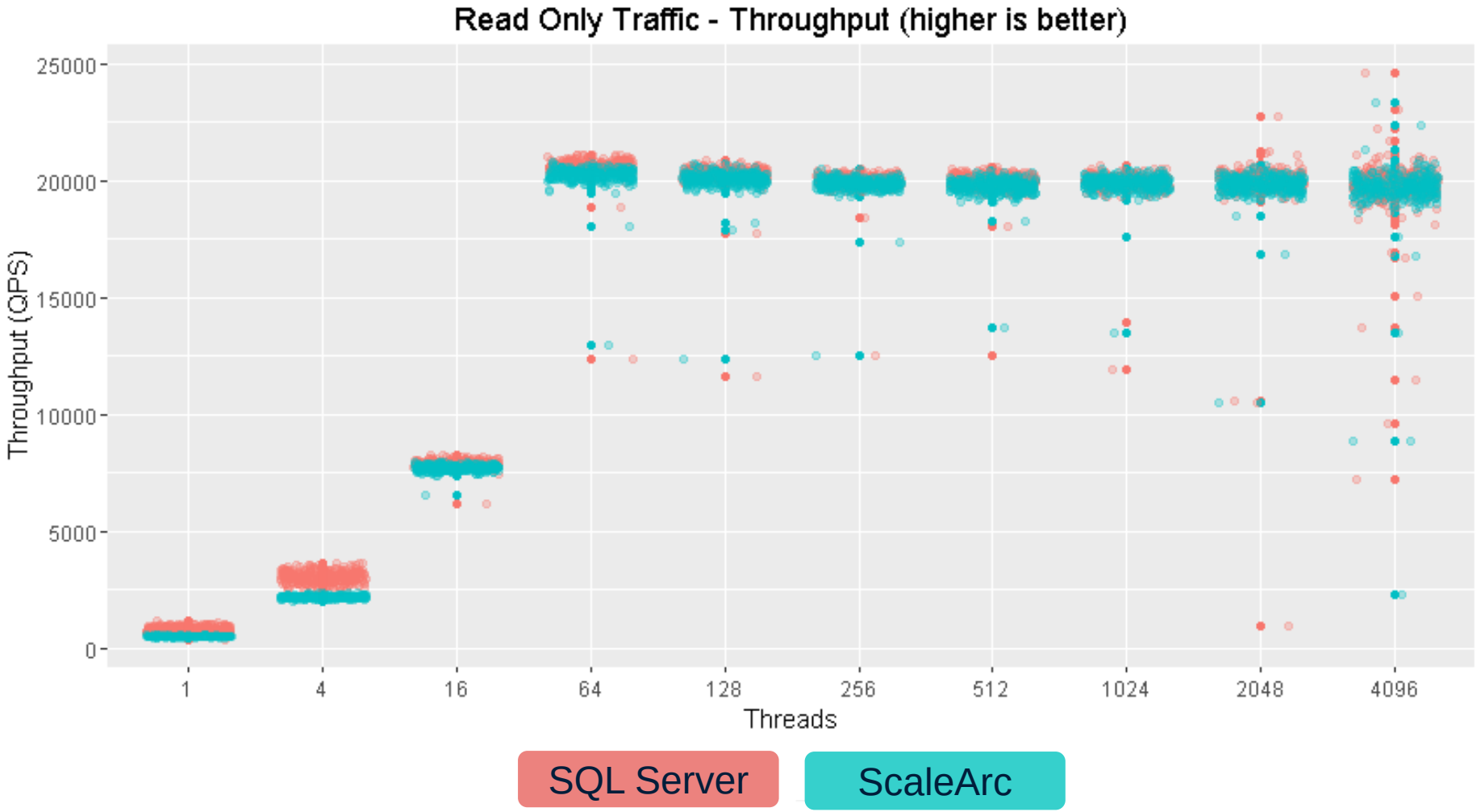
DELETE FROM <table_name> WHERE id=<random_id_5>; INSERT INTO <table_name>(id, k, c, pad) VALUES(<random_id_5>, <random_number>, '<random_string_1>', '<random_string_2>')

Note: For read-only workload testing, only SELECT queries were executed from the batch.

ScaleArc

Throughput in Queries Per Second – Read Only

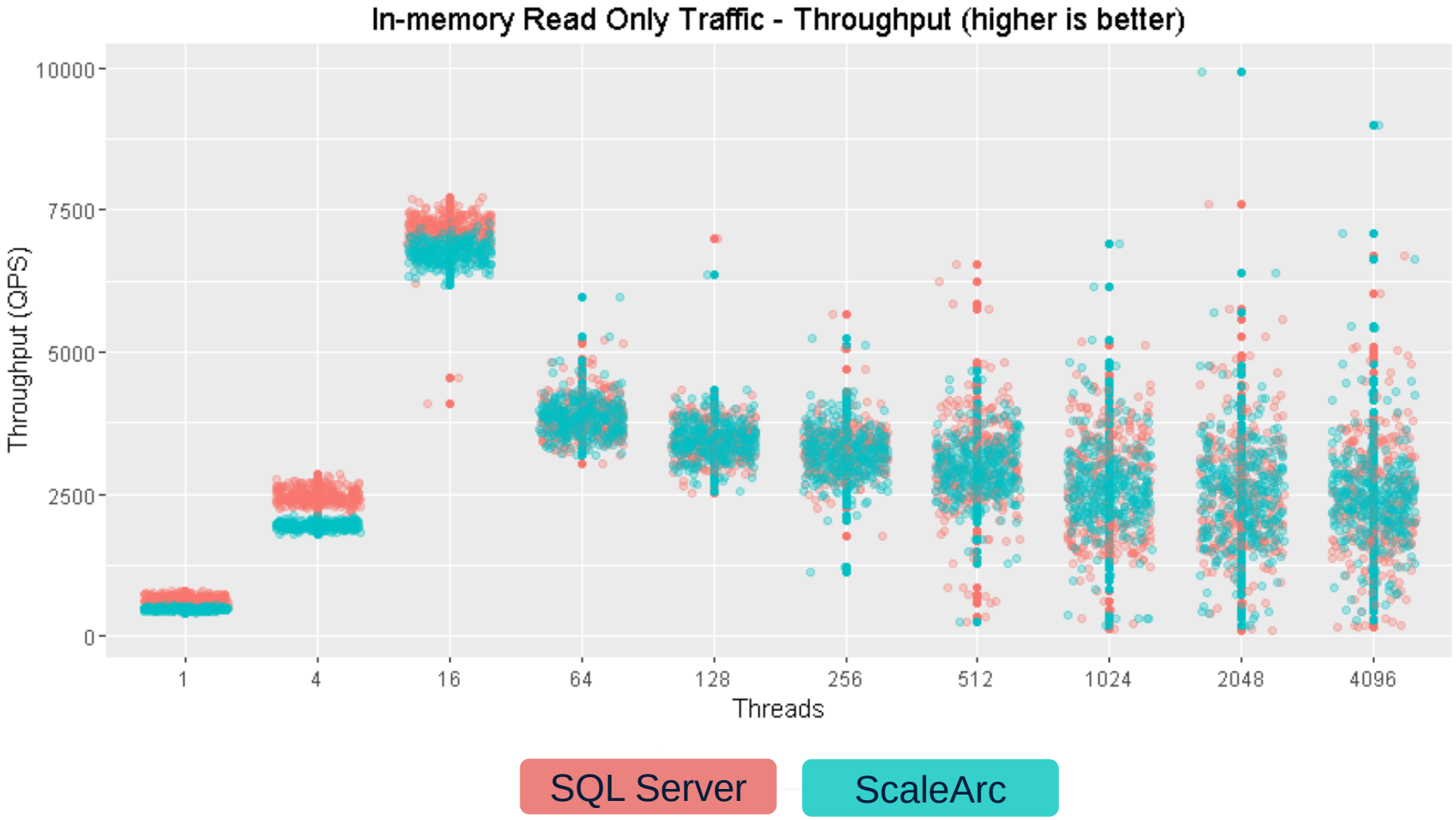
- Key findings:
  - Beyond 16 connections, throughput in queries per second (QPS) for most traffic is equivalent going through ScaleArc vs. going directly to the database
  - At less than 16 connections (atypical for OLTP applications), performance going through ScaleArc is 80% to 90% of the performance of traffic going directly to the database



ScaleArc

Throughput in Queries Per Second – In Memory, Read Only

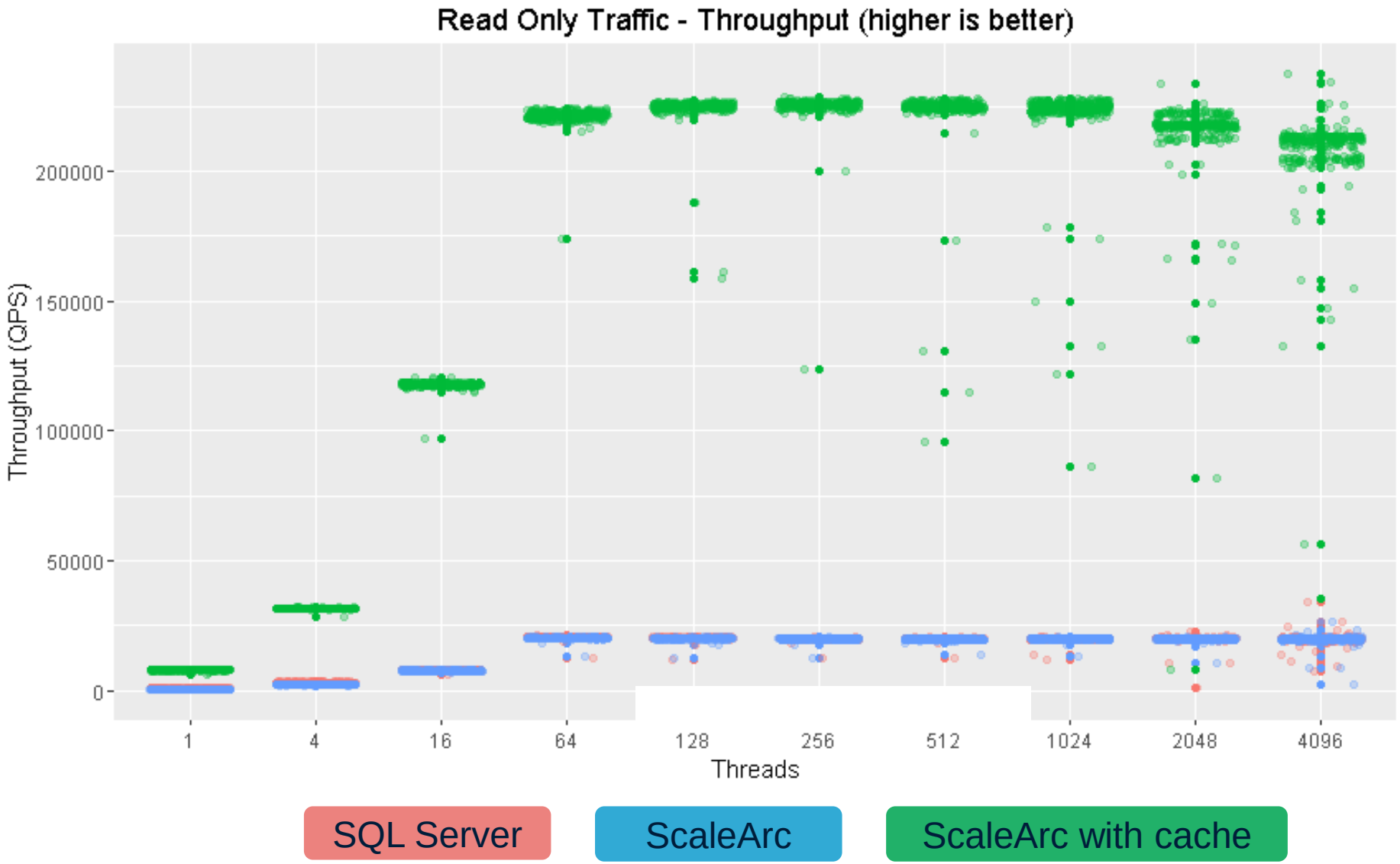
- Key findings:
  - Beyond 16 connections, throughput in queries per second (QPS) is equivalent between traffic going through ScaleArc vs. directly to the database
  - At less than 16 connections (atypical for OLTP applications), performance going through ScaleArc is 80% to 90% of the performance of traffic going directly to the database



ScaleArc

Effects of Caching – Throughput in Queries Per Second, Read Only

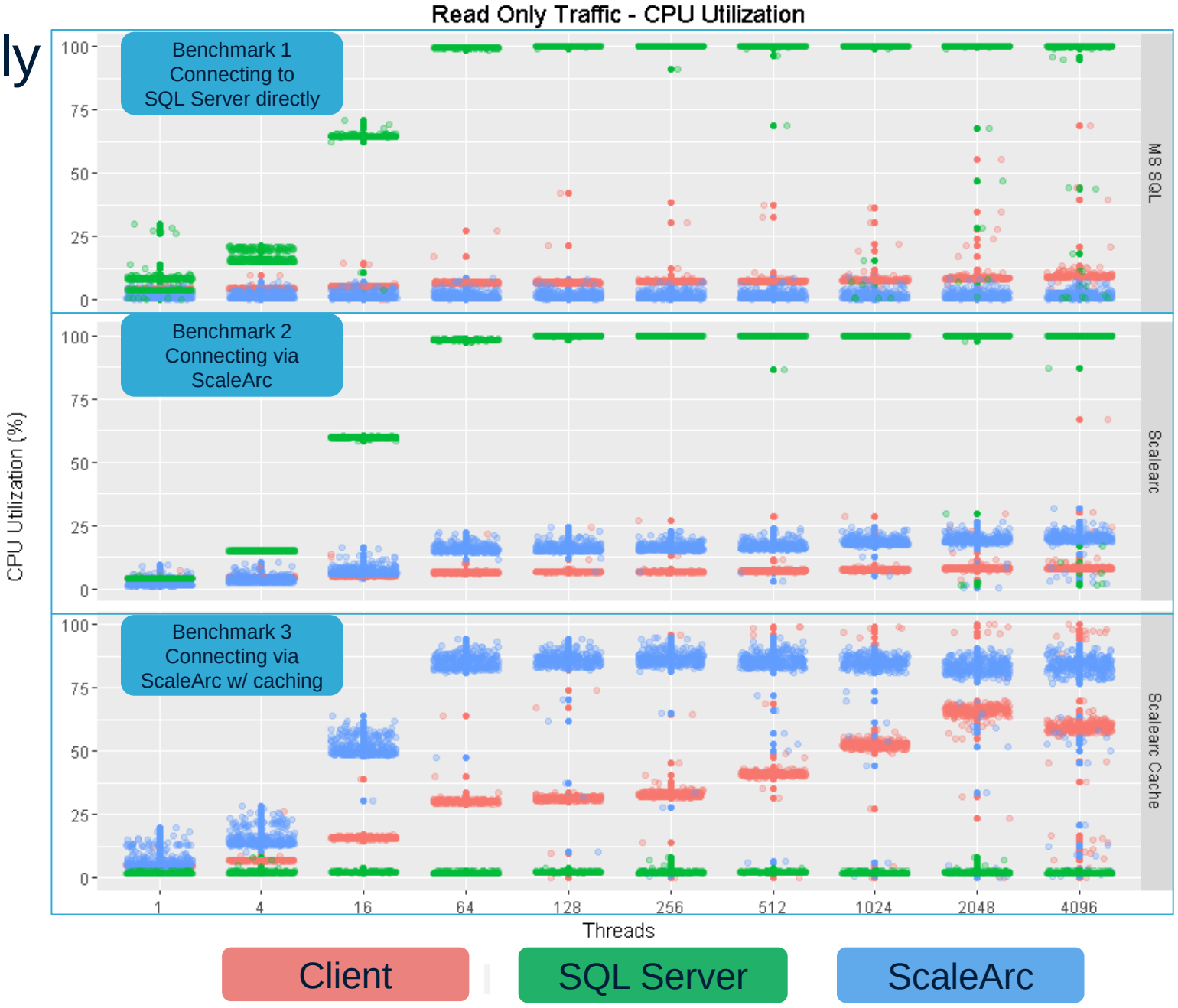
- Key findings:
  - Beyond 4 connections, caching delivers a substantial improvement in throughput in queries per second (QPS)
  - ScaleArc with caching enabled is nearly 8x faster than SQL Server 2014, for read-only traffic



ScaleArc

CPU Utilization – Read Only

- Key findings:
  - In Benchmarks 1 and 2, the database server's CPU is the bottleneck – the client node's CPU remains more than 75% idle
  - In Benchmark 2, ScaleArc's CPU remains 75% idle – we would have needed to run this load to 3 additional database servers to fully tap the ScaleArc CPU
  - In Benchmark 3 (caching), the database server's CPU is barely used, because when ScaleArc serves a query from cache, the query never hits the database, lowering the database's CPU utilization



Throughput in Queries Per Second – Read+Write

- Key findings
  - Beyond 16 connections, throughput in queries per second (QPS) is nearly equivalent
  - At less than 16 connections (atypical for OLTP applications), performance going through ScaleArc is 80% to 90% of the performance of traffic going directly to the database
  - Read+write traffic shows more scattered results than read-only traffic

ScaleArc

Effects of Caching – Throughput in Queries Per Second, Read+Write

- Testing methodology:
  - Read:Write ratio was 5:1,

  [18] threads:64, tps:0, reads/s:93434, writes/s:18710, avg_responsetime:0.008917106 ms
- Key findings:
  - Beyond 4 connections, caching delivers a substantial improvement in throughput in queries per second (QPS)
  - ScaleArc with caching enabled is 3.5x to 4x faster than SQL Server 2014, for read + write traffic

ScaleArc

CPU Utilization – Read+Write

- Key findings:
  - In Benchmarks 1 and 2, the database server's CPU is the bottleneck – the client node's CPU remains more than 75% idle
  - In Benchmark 2, ScaleArc's CPU remains 75% idle – we would have needed to run this load to 3 additional database servers to fully tap the ScaleArc CPU
  - In Benchmark 3 (caching), the database server's CPU is 50% barely used, for serving write queries