Task 1

|  |  |  |
| --- | --- | --- |
| **Storage & Database** | **Limitations** | **Characteristics** |
| Google Cloud Storage | * Does not provide Mobile SDKs * Unstructured data | * Single API across storage classes * Scalable to exabytes of data * Designed for 99.999999999% durability * Very high availability across all storage classes * Time to first byte in milliseconds * Strongly consistent listing * Zero carbon emissions |
| Cloud SQL | * Only for rational structured data * Not for analytical workload * Does not support horizontal scalability | * Scalability * High Performance * Integrated * Simple & Fully Managed * Reliability & Security |
| Cloud Bigtable | * No consistency guarantees for multi-row updates or cross-table updates * Does not support SQL queries or joins * Not a good solution for small amounts of data (< 1 TB) | * High Performance * Security & Permissions * Low Latency * Fully Managed * Redundant Autoscaling * Seamless Cluster Resizing * HBase Compatible * Global Availability * High Durability |
| Cloud Spanner | * Expensive * Higher latencies * Not ideal for high write-throughput use-cases * Write API is not strictly SQL * Vendor lock-in | * Global Scale * Fully Managed * Relational Semantics * Multi-Language Support * Transactional Consistency * Enterprise Grade Security * Highly Available |
| Cloud Datastore | * Expensive * Slower writing data due to synchronous replication | * Rich Admin Dashboard * Multiple Access Methods * Fully Managed * Diverse Data Types * ACID Transactions * Fast & Highly Scalable * Easy to Use Query Language * Simple & Integrated |

Google Cloud Storage can be accessed using gsutil tool, Cloud Storage Client Libraries, XML API, JSON API and Storage Transfer Service. The gsutil is a Python application that lets user access Cloud Storage from the command line, and it supports a wide range of bucket and object management tasks, including Creating and deleting buckets, Uploading, downloading and deleting objects, listing buckets and objects, moving, copying and renaming objects, editing object and bucket ACLs. Cloud Storage Client Libraries provide interface to access the Storage via C#, GO, JAVA, NODE.JS, PHP, PYTHON and RUBY program. XML API is a RESTful interface that lets user manage Cloud Storage data in a programmatic way. Access to Cloud Storage through the XML API is useful when users are using tools and libraries that must work across different storage providers, or when users are migrating from another storage provider to Cloud Storage. In the latter case, users only need to make a few simple changes to their existing tools and libraries to begin sending requests to Cloud Storage. JSON API is a simple, JSON-backed interface for accessing and manipulating Cloud Storage projects in a programmatic way. It is fully compatible with the Cloud Storage Client Libraries and intended for software developers familiar with web programming and be comfortable creating applications that consume web services through HTTP requests. Storage Transfer Service can be used to quickly import online data into Cloud Storage and transfer data within Cloud Storage, from one bucket to another. The service can be used on console, with Google API Client Library in JAVA and PYTHON, and with Storage Transfer Service API. It is more recommended than gsutil when transferring data from another cloud storage provider. All APIs requires authentication before accessing the Storage for most of the operations. Cloud Storage uses OAuth 2.0 for API authentication and authorization. What’s more, Cloud Storage can be integrated with Google Cloud Platform Services and Tools, for example, mounted as a file system on a virtual machine instance in Compute Engine.

We did single transfer benchmark test at 5 different time periods of a day. And for each period, 10 repeats were done. Here is our results.

|  |  |  |  |
| --- | --- | --- | --- |
| single transfer-Upload(MB/S) | MIN | MAX | AVG |
| 20180521 04:10 - 05:00 | 0.945851 | 1.015617 | 0.979699 |
| 20180521 10:50 - 11:30 | 0.917444 | 0.986666 | 0.953065 |
| 20180521 15:20 - 16:00 | 0.889677 | 0.990583 | 0.943145 |
| 20180521 20:40 - 21:20 | 0.856248 | 0.956707 | 0.922195 |
| 20180521 23:10 - 23:50 | 0.841479 | 0.947617 | 0.89935 |

|  |  |  |  |
| --- | --- | --- | --- |
| single transfer-Download(MB/S) | MIN | MAX | AVG |
| 20180521 04:10 - 05:00 | 4.746368 | 6.977806 | 5.741465 |
| 20180521 10:50 - 11:30 | 3.708112 | 6.733701 | 5.307302 |
| 20180521 15:20 - 16:00 | 3.256289 | 5.786416 | 4.436135 |
| 20180521 20:40 - 21:20 | 1.953979 | 3.211833 | 2.559959 |
| 20180521 23:10 - 23:50 | 2.396541 | 5.235984 | 3.340386 |

As we can see that the speed goes down from morning to evening and the rush hours are in the evening.

For parallel transfers, we did 20 repeats in which all 5 processes started uploading/downloading at the same time and another 20 repeats in which there was 5s interval between processes. And here is the average spends of each process.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Average Speed(MB/S) | P1 | P2 | P3 | P4 | P5 |
| parallel transfer-Upload | 0.265227 | 0.27277 | 0.272877 | 0.270459 | 0.269825 |
| parallel transfer(5s)-Upload | 0.293085 | 0.28216 | 0.280574 | 0.279158 | 0.279713 |
| parallel transfer-Download | 4.144846 | 4.025731 | 4.125107 | 4.038982 | 4.046356 |
| parallel transfer(5s)-Download | 4.248248 | 4.200298 | 4.148264 | 4.213221 | 4.204435 |

As we can see that the sum of speeds of parallel transfer processes are higher than the speed of single transfer, we suspect that google has speed limitation on each connection.

We did not notice any newcomer advantage or disadvantage.

Task 3

VM can access Cloud Storage by using all the APIs mentioned in task 1. And by using Cloud Storage URI, VM can export an image to Cloud Storage and use a startup script stored in Cloud Storage. What’s more, VM can mount a bucket on Cloud Storage as a file system with Google Cloud Storage FUSE tool. The mounted bucket behaves similarly to a persistent disk even though Cloud Storage buckets are object storage. The FUSE has been tested successfully with Linux (minimum kernel version 3.10) and OS X (minimum version 10.10.2). Cloud Storage FUSE works by translating object storage names into a file and directory system, interpreting the “/” character in object names as a directory separator so that objects with the same common prefix are treated as files in the same directory. Applications can interact with the mounted bucket like a simple file system, providing virtually limitless file storage running in the cloud. While Cloud Storage FUSE has a file system interface, it is not like an NFS or CIFS file system on the backend. Cloud Storage FUSE retains the same fundamental characteristics of Cloud Storage, preserving the scalability of Cloud Storage in terms of size and aggregate performance while maintaining the same latency and single object performance. As with the other access methods, Cloud Storage does not support concurrency and locking. For example, if multiple Cloud Storage FUSE clients are writing to the same file, the last flush wins. Cloud Storage FUSE has much higher latency than a local file system. As such, throughput may be reduced when reading or writing one small file at a time. Using larger files and/or transferring multiple files at a time will help to increase throughput. Individual I/O streams run approximately as fast as gsutil. Small random reads are slow due to latency to first byte. Random writes are done by reading in the whole blob, editing it locally, and writing the whole modified blob back to Cloud Storage. Small writes to large files work as expected, but are slow and expensive. After installing and setting up credentials for Cloud Storage FUSE, users can mount a bucket to a VM simply by “gcsfuse example-bucket /path/to/mount”.

We ran following benchmark test cases for external storage and local SSD.

|  |  |  |  |
| --- | --- | --- | --- |
|  | ID | External Storage | Local SSD |
| Test Case | 1 | write\_bandwidth\_test | write\_bandwidth\_test |
| 2 | write\_iops\_test | write\_iops\_test |
| 3 | write\_latency\_test | write\_latency\_test |
| 4 | read\_bandwidth\_test | read\_bandwidth\_test |
| 5 | read\_iops\_test | read\_iops\_test |
| 6 | read\_latency\_test | read\_latency\_test |
| 7 | write\_bandwidth\_test\_seq | write\_bandwidth\_test\_seq |
| 8 | write\_iops\_test\_seq | write\_iops\_test\_seq |
| 9 | write\_latency\_test\_seq | write\_latency\_test\_seq |
| 10 | read\_bandwidth\_test\_seq | read\_bandwidth\_test\_seq |
| 11 | read\_iops\_test\_seq | read\_iops\_test\_seq |
| 12 | read\_latency\_test\_seq | read\_latency\_test\_seq |
| 13 | write\_bandwidth\_test\_pt\_seq | write\_bandwidth\_test\_pt |
| 14 | write\_iops\_test\_pt\_seq | write\_iops\_test\_pt |
| 15 | write\_latency\_test\_pt\_seq | write\_latency\_test\_pt |
| 16 | read\_bandwidth\_test\_pt\_seq | read\_bandwidth\_test\_pt |
| 17 | read\_iops\_test\_pt\_seq | read\_iops\_test\_pt |
| 18 | read\_latency\_test\_pt\_seq | read\_latency\_test\_pt |

“pt” means parallel transfer with 4 processes, otherwise it is single transfer. “seq” means it is sequential read/write, otherwise it is random read/write.

For External Storage test cases, we used Ubuntu 16.04 LTS with 4 vCPU, 15 GB memory and 10 G SSD and executed the test cases in 3 regions.

|  |  |  |  |
| --- | --- | --- | --- |
| VM Location | Storage Location | Different Time Periods of a day | Repetitions of each period |
| europe-west2-a | EUROPE-WEST2 | 3 | 9 |
| asia-northeast1-a | ASIA-NORTHEAST1 | 3 | 9 |
| us-central1-a | US-CENTRAL1 | 3 | 9 |

For Local SSD test cases, we executed them with following different configurations of Ubuntu 16.04 LTS.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SSD Size | Number of vCPU | Memory | Different Time Periods of a day | Repetitions of each period |
| 500G | 16 | 60G | 3 | 3 |
| 834G | 32 | 120G | 3 | 3 |
| 2048G | 64 | 240G | 3 | 3 |
| 2048G | 96 | 360G | 3 | 3 |

In total, 2106 cases were executed. Below is our findings from External Storage test.

1. The performance of random write is similar to that of sequential write, but the performance of sequential read is much higher than that of random read.

|  |  |  |
| --- | --- | --- |
|  | Random Average | Sequential Average |
| Write bandwidth (KB/s) | 103204.8889 | 102127.8889 |
| Write iops | 4201.333333 | 4202.222222 |
| Read bandwidth (KB/s) | 2643.4 | 303608.8889 |
| Read iops | 19 | 6063 |

Data from EU at 2018.0525.0940

1. Benchmark of single sequential transfer in EU region at different times of a day

|  |  |  |  |
| --- | --- | --- | --- |
| write\_bandwidth\_test\_seq(KB/S) | MIN | MAX | AVG |
| 2018.0525.0940 | 98490 | 103844 | 102127.8889 |
| 2018.0525.1510 | 99086 | 104331 | 101065.6667 |
| 2018.0525.2000 | 99698 | 103344 | 101740.5556 |

|  |  |  |  |
| --- | --- | --- | --- |
| write\_iops\_test\_seq | MIN | MAX | AVG |
| 2018.0525.0940 | 4042 | 4356 | 4202.222222 |
| 2018.0525.1510 | 4053 | 4360 | 4146 |
| 2018.0525.2000 | 3995 | 4356 | 4113.666667 |

|  |  |  |  |
| --- | --- | --- | --- |
| read\_bandwidth\_test\_seq(KB/s) | MIN | MAX | AVG |
| 2018.0525.0940 | 296671 | 313155 | 303608.8889 |
| 2018.0525.1510 | 293166 | 319347 | 307874.7778 |
| 2018.0525.2000 | 288964 | 318435 | 304650.6667 |

|  |  |  |  |
| --- | --- | --- | --- |
| read\_iops\_test\_seq | MIN | MAX | AVG |
| 2018.0525.0940 | 5955 | 6221 | 6063.555556 |
| 2018.0525.1510 | 6020 | 6242 | 6125.444444 |
| 2018.0525.2000 | 5953 | 6287 | 6107 |

No obvious difference on performance observed at different times of a day. We suspect that since both VM and storage are in the same region, the connections were established in the internal network.

1. Single sequential transfer V.S. parallel sequential transfer

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | ST AVG | PT-P1 AVG | PT-P2 AVG | PT-P3 AVG | PT-P4 AVG |
| Write bandwidth (KB/s) | 102127.8889 | 12303 | 12249 | 12277.33333 | 12122.11111 |
| Write iops | 4202.222222 | 1761.111111 | 1637.333333 | 1696 | 1692.111111 |
| Read bandwidth (KB/s) | 303608.8889 | 72365.22222 | 70629.22222 | 71673.33333 | 72270 |
| Read iops | 6063 | 5527.444444 | 5488.333333 | 5517.222222 | 5525.777778 |

Data from EU at 2018.0525.0940

As we can see that parallel transfer processes share similar performance.

1. EU V.S. Asia V.S. US

|  |  |  |  |
| --- | --- | --- | --- |
| write\_bandwidth\_test\_seq(KB/S) | MIN | MAX | AVG |
| EU | 98490 | 104331 | 101644.7037 |
| Asia | 83632 | 104814 | 96022.44444 |
| US | 13659 | 82156 | 45411.67 |

|  |  |  |  |
| --- | --- | --- | --- |
| write\_iops\_test\_seq | MIN | MAX | AVG |
| EU | 3995 | 4360 | 4153.962963 |
| Asia | 3448 | 4360 | 4090.166667 |
| US | 3567 | 4299 | 4123.704 |

|  |  |  |  |
| --- | --- | --- | --- |
| read\_bandwidth\_test\_seq(KB/s) | MIN | MAX | AVG |
| EU | 288964 | 319347 | 305378.1111 |
| Asia | 242953 | 314806 | 275000.7222 |
| US | 97629 | 276727 | 203676.8 |

|  |  |  |  |
| --- | --- | --- | --- |
| read\_iops\_test\_seq | MIN | MAX | AVG |
| EU | 5953 | 6287 | 6098.666667 |
| Asia | 3649 | 6442 | 5636.666667 |
| US | 5082 | 6044 | 5715.333 |

Data from all 3 different time periods

We can see that EU has the best performance and US has the worst/most unstable performance.

And following is our result from local SSD test.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 500G(16 vCPUs, 60 GB memory) | 834G(32 vCPUs, 120 GB memory) | 2048G(64 vCPUs, 240 GB memory) | 2048G(96 vCPUs, 360 GB memory) |
| Random Write IOPS | 15000 | 25000 | 30000 | 30000 |
| Random Read IOPS | 15000 | 11000 | 13000 | 19000 |
| Seq Write IOPS | 15000 | 25000 | 30000 | 30000 |
| Seq Read IOPS | 15000 | 25000 | 40000 | 40000 |
| Random Write BW (MB/S) | 245 | 410 | 410 | 410 |
| Random Read BW (MB/S) | 245 | 410 | 820 | 820 |
| Seq Write BW (MB/S) | 245 | 410 | 410 | 410 |
| Seq Read BW (MB/S) | 245 | 410 | 820 | 820 |
| Expected Random Write IOPS | 15000 | 25000 | 30000 | 30000 |
| Expected Random Read IOPS | 15000 | 25000 | 40000 | 40000 |
| Expected Random Write BW (MB/S) | 240 | 400 | 400 | 400 |
| Expected Random Read BW (MB/S) | 240 | 400 | 800 | 800 |

Google provided SLA on random IOPS and bandwidth for SSD in following page.

<https://cloud.google.com/compute/docs/disks/performance>

Our result shows that sequential read/write performance is similar to random read/write performance, and matches with Google SLA, except the random read iops of 834G and 2048G size SSD, which is always below the expected IOPS. From the result, it looks like the higher number of vCPU and more memory, the better the performance, so we are not sure if the unexpected low performance was caused by insufficient resources as we can max assign 96 vCPU to a VM.

As mentioned earlier, we also ran tests on latency for both external storage and local SSD, however, due to time limitation and latency is less interesting for this task, we do not provide analysis on the latency test result. But if you are interested, you can find the result in our original result files.