SedonaDB Performance Benchmark

Authors: Wherobots Inc.

Release Date: 10/24/2023

Table of Content

To better showcase SedonaDB's performance, we conducted a comprehensive performance benchmark on some commonly seen spatial data processing tasks.

Settings

All the tests uses open datasets and were performed on Wherobots cloud. We setup 2 Wherobots clusters with different configurations to run the benchmarks (Medium cluster and Large cluster respectively). The following table shows the setups of our clusters. Running the benchmarks on clusters of different sizes allows us to evaluate the scalability of SedonaDB.

Configuration	Medium cluster	Large cluster
Instance Type	General purpose	General purpose
Instance vCPUs	16	32
RAM	64 GB	128 GB
EBS Volume	500 GB	500 GB
EBS Throughput	125 MB/s	125 MB/s
Instances	10 executors + 1 driver	10 executors + 1 driver
AWS Region	us-west-2	us-west-2

For some tasks whose performance were bounded by EBS throughput, we'll run them on the clusters with same configuration, except that the EBS throughput was raised to 500 MB/s. We'll see how EBS throughput affects the performance and scalability of shuffle-heavy workloads.

Wherobots Cloud runs all the clusters in AWS us-west-2 region, so all the EC2 instances and S3 buckets are in us-west-2.

Geometry / Geography Data Processing

Datasets

We are using 2 datasets to benchmark the performance of SedonaDB:

OSM Nodes

This dataset contains all the nodes of the OpenStreetMap Planet dataset. This dataset is preprocessed to parquet files containing WKB representation of nodes. All the geometries in this dataset are POINTs. There are roughly 8 billion points in this dataset.

Overture Buildings

This dataset is the GeoParquet version of Overture Buildings provided by Wherobots, which are essentially parquet files containing WKB representation of geometries. It contains 800 million geometries, which are POLYGONs and MULTIPOLYGONs.

Postal Codes

OSM2015/postal_codes dataset contains boundaries of postal code areas as defined in OpenStreetMap. This dataset is preprocessed to parquet files containing WKB representation of zones. This dataset is filtered to contain only POLYGONs. There are 154,452 polygons in this dataset. This dataset is quite small and was only used in the spatial join performance benchmarks.

We've split the OSM Nodes and Overture Buildings datasets into two non-disjoint parts: a larger part (90%) and a smaller part (10%). The larger part is for running data preprocessing and spatial queries benchmarks. The smaller part is for running the ACID transactions to insert or merge the smaller part into the Havasu table containing the larger part.

The overview of datasets used in this benchmark is shown in the follow table.

Dataset	Size (Parquet files)	Number of Rows	Type of Geometries
OSM Nodes Large (90%)	256 GB	7,456,990,919	POINT
OSM Nodes Small (10%)	30 GB	828,501,977	POINT
Overture Buildings Large (90%)	103 GB	785,524,851	POLYGON and MULTIPOLYGON
Overture Buildings Small (10%)	12 GB	78,557,756	POLYGON and MULTIPOLYGON
Postal Codes	0.86 GB	154,452	POLYGON

Data Pre-processing

We'll run 2 data pre-processing tasks to ingest the datasets into Havasu tables.

1. **Migrate Data Files**: Loading the parquet files directly into Havasu table without copying the data files. This is done using the `[migrate` procedure](https://iceberg.apache.org/docs/latest/spark-procedures/#migrate) of Apache Iceberg, which builds iceberg table's metadata using existing parquet files (without scanning or copying all the data in these parquet files). Please refer to the Wherobots documentation for detailed steps.
2. **Create Spatial Index**: Reorder the records in the Havasu table to sort by the hilbert index of geometries for raster spatial queries. This requires sorting and rewriting the entire Havasu table and needs to run regularly when new data were ingested into the tables.

Performance of Migrate Data Files

	Medium cluster	Large cluster
Loading OSM Nodes Large	33.129 s	33.704 s

Loading Overture Buildings Large	31.807 s	27.673 s
Loading Postal Codes	3.503 s	2.372 s

Data file migration is not a distributed task in Havasu. The metadata of parquet files were all collected by the driver node. So we see constant time spent on this task, regardless of the size of the clusters.

Performance of Create Spatial Index

	Medium cluster	Large cluster
OSM Nodes Large	1195.108 s	642.982 s
Overture Buildings Large	446.820 s	381.735 s
Postal Codes	50.691 s	54.121 s

Creating spatial index is a distributed task requires shuffling. For OSM Nodes and Overture Buildings dataset the performance scales with the size of the cluster. There is no performance improvement observed for creating spatial index for Postal Codes dataset. This dataset is small enough to be handled by a medium-sized cluster so there's no point processing it on a larger cluster.

Spatial Queries

Spatial Transformation Queries

We've run several commonly seen spatial data transformation tasks on OSM Nodes and Overture Buildings dataset. The data transformations were benchmarked on uncached Havasu tables as well as cached Havasu tables. To make the table being fully cached in memory, we allocated 30 GB off-heap memory on each node and cached the tables using OFF_HEAP storage level when running queries on cached tables.

The time spent running these tasks were shown in the table below.

	Medium cluster	Large cluster	Medium cluster cached	Large cluster cached
`ST_AreaSpheroid` - OSM Nodes	55.826 s	31.119 s	27.004 s	11.814 s
`ST_LengthSpheroid` - OSM Nodes	58.987 s	31.213 s	26.368 s	10.908 s
`ST_Envelope_Aggr` - OSM Nodes	71.953 s	36.846 s	41.514 s	16.729 s
`ST_Transform` - OSM Nodes	122.727 s	64.753 s	107.811 s	48.674 s
`ST_Buffer` - OSM Nodes	849.202 s	417.091 s	898.525 s	379.535 s
`ST_Centroid` - OSM Nodes	60.521 s	31.168 s	31.395 s	13.715 s
`ST_AreaSpheroid` - Overture Buildings	79.537 s	41.045 s	75.011 s	33.563 s
`ST_LengthSpheroid` - Overture Buildings	64.836 s	33.093 s	58.010 s	25.970 s
`ST_Envelope_Aggr` - Overture Buildings	17.889 s	10.186 s	8.580 s	4.025 s
`ST_Transform` - Overture Buildings	26.537 s	14.053 s	17.576 s	7.553 s
`ST_Buffer` - Overture Buildings	150.910 s	76.545 s	152.890 s	70.672 s
`ST_Centroid` - Overture Buildings	19.085 s	10.386 s	8.761 s	3.850 s

For simple spatial data transformations, SedonaDB shows almost linear scalability. Readers can estimate the time spent for these data transformations tasks processing their own data in their own Wherobots cluster faithfully based on these data. We've also found that caching the table accelerates IO-bounded queries such as `ST_Envelope_Aggr` and `ST_Centroid` significantly. For CPU-bounded queries such as `ST_Buffer` and `ST_Transform`, the table caching does not show significant improvement.

Spatial Range Queries

We run spatial range queries on both OSM Nodes and Overture Buildings using query windows of different sizes, and observed significant speed up compared to running these queries on plain parquet files. 3 range query windows with different sizes (small, medium, large) are used in the benchmark. The detailed queries can be found in Appendix.

The range queries were benchmarked on uncached Havasu tables as well as cached Havasu tables. To make the table being fully cached in memory, we allocated 30 GB off-heap memory on each node and cached the tables using OFF_HEAP storage level when running queries on cached tables.

The benchmarking result of running spatial range queries using these query windows are shown in the following table. We can see observe that:

1. The performance of spatial range query on Havasu tables is way better than querying the plain parquet files. We observed 10x performance improvement on medium-sized cluster.
2. Medium-sized clusters is enough to achieve good range query performance. It is not necessary to run a big cluster to answer highly selective range queries.
3. Caching does not help accelerating spatial range queries, since it prevents the spatial filter push-down optimization from kicking-in. It is usually better to run spatial range queries on uncached tables.

	Medium cluster Havasu	Medium cluster Parquet	Large cluster Havasu	Large cluster Parquet	Medium cluster Havasu cached	Large cluster Havasu cached
OSM Nodes - small	5.006 s	56.290 s	4.783 s	24.979 s	26.182 s	9.769 s
OSM Nodes - medium	4.937 s	44.122 s	4.515 s	23.017 s	26.114 s	9.983 s
OSM Nodes - large	14.067 s	41.679 s	8.639 s	23.074 s	26.399 s	10.262 s
Overture Buildings - small	5.374 s	20.820 s	3.747 s	10.466 s	6.866 s	3.129 s
Overture Buildings - medium	4.900 s	14.942 s	3.926 s	9.578 s	7.972 s	4.612 s
Overture Buildings - large	6.940 s	15.050 s	4.268 s	9.508 s	6.888 s	3.106 s

Please note that spatial range query performance of Havasu table is largely determined by the distribution of the data. It works best when the data were properly clustered by spatial proximity. Please refer to the Wherobots documentation for details.

KNN Query

We run KNN queries with N = 10 using the following locations as query window:

Name	WKT	Comment
LA	POINT (-118.19854981529845 33.96658592049561)	LA
NY	POINT(-73.98591899205337 40.75149554645705)	NY Manhattan

The KNN queries were benchmarked on uncached Havasu tables as well as cached Havasu tables. To make the table being fully cached in memory, we allocated 30 GB off-heap memory on each node and cached the tables using OFF_HEAP storage level when running queries on cached tables.

The following table shows the benchmarking results. KNN queries in SedonaDB cannot benefit from spatial filter push down and needs to scan the entire dataset, so the performance is irrelevant with the query window. We see linear

scalability when running these queries. KNN queries are IO-bounded, so caching the tables significantly improves the performance.

	Medium cluster	Large cluster	Medium cluster cached	Large cluster cached
OSM Nodes - LA	78.470 s	36.593 s	42.247 s	17.935 s
OSM Nodes - NY	74.715 s	35.631 s	42.502 s	17.816 s
Overture Buildings - LA	19.424 s	11.295 s	11.187 s	5.319 s
Overture Buildings - NY	19.303 s	10.769 s	11.118 s	5.163 s

Spatial Join

We run 3 spatial joins on both clusters. Two of them joins a large dataset (OSM Nodes, Overture Buildings) with a medium sized dataset (Postal Codes), and one joining two large datasets. We see linear scalability when running these queries.

	Medium cluster	Large cluster	Comments
OSM Nodes vs. Postal Codes	820.431 s	440.613 s	
Overture Buildings vs. Postal Codes	189.744 s	126.981 s	
OSM Nodes vs. Overture Buildings	1088.754 s	524.181 s	Using R-Tree for indexing. The default option Quad-Tree is painfully slow in this case.

We've also noticed that running spatial joins on OSM Nodes vs. Overture Buildings using the default spatial quad-tree indexing is very slow, so we switched to use R-tree for spatial indexing and observed a huge performance improvement. According to our experience, R-tree index is usually faster when running polygon vs. polygon spatial joins, while the default quad-tree index works well when one side of the geometries is point.

Data Update Transactions

Insertion

We run `INSERT INTO` using various size of dataset on Havasu tables. The inserted dataset were taken from the smaller (10%) part of the original dataset, so they are completely disjoint with the data in Havasu tables. We can see that both clusters spent almost the same time for inserting 100K ~ 10M rows. If users have regular data ingestion tasks to insert less than 10M rows, a medium sized cluster could be enough.

	Medium cluster	Large cluster
OSM Nodes Large 100K rows	3.212 s	2.507
OSM Nodes Large 1M rows	9.393 s	7.490 s
OSM Nodes Large 10M rows	22.260 s	22.752 s
Overture Buildings Large 100K rows	5.514 s	4.773 s
Overture Buildings Large 1M rows	25.051 s	24.769 s
Overture Buildings Large 10M rows	29.282 s	31.411 s

Merge Into

We run `MERGE INTO` tasks using various size of dataset on Havasu tables. The datasets used in Merge Into tasks are:

- *0% updates*: all records were taken from the 10% smaller split.

- *5% updates*: 95% records were taken from the 10% smaller split, the other 5% were taken from the larger split. The dataset was shuffled to uniformly distribute the inserted records and updated records.

0% Updates:

	Medium cluster	Large cluster
OSM Nodes Large 100K rows	14.183 s	9.818 s
OSM Nodes Large 1M rows	17.344 s	12.483 s
OSM Nodes Large 10M rows	139.103 s	152.072 s
Overture Buildings Large 100K rows	13.698 s	11.325 s
Overture Buildings Large 1M rows	75.208 s	31.688 s
OSM Nodes Large 10M rows	85.540 s	34.619 s

5% Updates:

	Medium cluster	Medium cluster with 500MB/s EBS	Large cluster	Large cluster with 500MB/s EBS
OSM Nodes Large 100K rows	176.457 s	132.806 s	90.812 s	75.126 s
OSM Nodes Large 1M rows	456.985 s	370.200 s	411.686 s	221.397 s
OSM Nodes Large 10M rows	883.080 s	660.144 s	856.020 s	415.859 s
Overture Buildings Large 100K rows	267.536 s	236.750 s	230.177 s	135.999 s
Overture Buildings Large 1M rows	356.306 s	286.469 s	276.332 s	167.203 s
Overture Buildings Large 10M rows	351.929 s	292.439 s	292.787 s	180.121 s

We can see that Merge Into without updates is significantly faster than Merge Into with 5% updates, since it does not need to rewrite existing data files to perform updates. However it is still slower than Insert Into because it needs to scan the entire table to find updated records.

The performance of Merge Into with 5% updates on OSM Nodes table is not linearly scalable. This is because we are hitting the EBS throughput bottleneck (125MB/s). Merge Into involves lots of shuffling, which does lots of disk read and write, so we don't see the scalability of performance when switching to larger EC2 instances. However, we can see the scalability of performance after raising the throughput of EBS to 500 MB/s, since the EBS is not the performance bottleneck and the task became CPU bounded.

Raster Data Benchmark

Datasets

We are using the USGS Landsat dataset to benchmark the performance of SedonaDB. The Landsat dataset used in raster data processing benchmark is Landsat Collection 2 Level-2 UTM Surface Temperature (ST) Product. The GeoTIFF images used in this benchmark are B10 `lwir11` band, which is the surface temperature band. The pixel values indicate the Kelvin temperature of the surface.

The pixel values of B10 band are UInt16 (0 ~ 65535). To obtain surface temperature values, convert the values to Kelvin using `V_kelvin = V_pixel * 0.00341802 + 149.0` (See USGS FAQ). We'll benchmark this conversion in the map algebra benchmarks.

We've taken different size of subsets to test various functionality of Sedona-DB. For benchmarking in-db raster operations we take roughly 1TB of Landsat GeoTIFF images; for benchmarking out-db raster operations we take roughly 10TB of Landsat GeoTIFF images. Please refer to the following table for details of these subsets.

Dataset	Size (storage)	Number of Records
Landsat Out-DB	11483 GB	166912
Landsat In-DB	1025 GB	14890
Postal Codes	0.86 GB parquet files	154,452

The subsamples of the Landsat dataset used in the benchmarks span the entire globe and covers all the lands except the arctic.

Data Pre-processing

This task reads the data from requester-payer buckets owned by USGS and write data into Havasu.

- In-DB: Reads the entire GeoTIFF image data and write into parquet data files
- Out-DB: Only read the metadata of GeoTIFF and write metadata into parquet data files

The data loaded into Havasu were preprocessed to 1024x1024 tiles before writing into Havasu tables, so that the rasters would fit in the RAM of our cluster for further processing (especially map algebra).

The tables were partitioned by (month, SRID) for supporting efficient spatial-temporal query. The reason why we use SRID as partitioned column is because the rasters are in UTM coordinate reference system, so the SRID is a natural spatial partition for this raster dataset.

	Medium cluster	Medium cluster with 500MB/s EBS	Large cluster	Large cluster with 500MB/s EBS
Loading Landsat In-DB	4185.918 s	1726.541 s	4056.993 s	1568.865 s
Loading Landsat Out-DB	327.614 s	243.775	199.664 s	166.966 s

We can see that loading out-db rasters is much faster than loading in-db datasets, even though the number of out-db dataset is 10x larger than the in-db dataset. This is because loading out-db rasters only reads the metadata of GeoTIFF files and extract the image size, pixel type and geo-referencing information, and don't need to read the entire GeoTIFF file from remote storage. Also it is faster to sort the out-db rasters for partitioning than in-db rasters.

We've not observed linear scalability when processing in-db rasters. This is caused by EBS volume throughput (125MB) being hit during the shuffle phases of ingesting data into partitioned tables.

Raster Data Queries

We've run several commonly seen raster data processing tasks on both clusters. We can see that the performance scales linearly for in-db rasters. The reason why the performance does not show linear scalability for out-db rasters can be caused by the network bandwidth bottleneck of EC2 instance or throttling of S3.

Raster Transformation Queries: In-DB

	Medium cluster	Large cluster
`RS_Metadata`	267.831 s	169.888 s
`RS_ConvexHull`	262.358 s	132.325 s
`RS_Count`	290.708 s	155.013 s
`RS_Value` of center point	266.704 s	143.976 s

`Map Algebra`	675.441 s	357.681 s
`RS_Resample`		
ScaleX = 100, ScaleY = -100	374.507 s	187.974 s
`RS_Tile` (256x256)	331.539 s	198.095 s

Raster Transformation Queries: Out-DB

	Medium cluster	Large cluster
`RS_Metadata`	11.122 s	7.112 s
`RS_ConvexHull`	8.164 s	6.299 s
`RS_Count`	9678.901 s	7815.650 s
`RS_Value` of center point	4138.579 s	2139.859 s
`Map Algebra`	14254.806 s	7279.975 s
`RS_Resample`		
ScaleX = 100, ScaleY = -100	10902.768 s	7805.940 s
`RS_Tile` (256x256)	36.715 s	26.374 s

There are several notable results for performance of out-db raster processing:

1. Pure metadata operations are very fast, since they only need to scan the Havasu data files to answer these queries and do not require reading the GeoTIFF files stored in remote storage.
2. `RS_Value` of center point is linearly scalable. This is because this query only need to read a small portion of GeoTIFF files, so no network bottleneck was hit.
3. `RS_Count`, Map Algebra and RS_Resample need to read all the pixel data of the out-db rasters, so the entire GeoTIFF files were read (10TB in total). They do not show linear scalability possibly because of hitting the EC2 network bottleneck or S3 traffic throttling.
4. `RS_Tile` is a pure geo-referencing transformation for out-db rasters, so it is very fast to tiling a large out-db raster into small chips.

Spatial-Temporal Queries

We run spatial range queries on both in-db and out-db datasets using query windows of different sizes. 3 range query windows with different sizes (small, medium, large) are used in the benchmark. The detailed queries can be found in Appendix.

We've also applied a temporal filter: `BEWTEEN TIMESTAMP '2023-08-01' AND TIMESTAMP '2023-09-01'` to query the data for August 2023. Both the temporal filter and the spatial query window can utilize the partitioning of Havasu tables.

	Medium cluster	Large cluster
Landsat In-DB	6.022 s	4.379 s
Landsat Out-DB	0.866 s	0.743 s

We can see that spatial queries can be answered within seconds for both in-db and out-db rasters. Out-db rasters are faster to query because each raster record is smaller in the data files and less data need to be scanned to answer the queries. Spatial query is also a metadata-only operation and don't need to load GeoTIFF files for out-db rasters.

Join Queries

Join queries requires shuffling the datasets. For in-db datasets, the performance is bounded by EBS throughput, since the size of the dataset and per-raster size are very large, the shuffle process for joining in-db rasters with geometry

involves shuffle writing and shuffle reading 2TB of data. That's the reason why the performance does not scale linearly. For out-db rasters, the spatial join queries are CPU-bounded, so raising the throughput of EBS does not help improving the overall performance.

	Medium cluster	Medium cluster with 500MB/s EBS	Large cluster	Large cluster with 500MB/s EBS
Landsat In-DB vs. Postal Codes	2204.226 s	1300.207 s	2195.321 s	1300.230 s
Landsat Out-DB vs. Postal Codes	124.984 s	115.051 s	79.581 s	84.675 s

Data update Transactions

Insertion

We run `INSERT INTO` using various size of dataset on Havasu tables. The inserted dataset were taken from the smaller (10%) part of the original dataset, so they are completely disjoint with the data in Havasu tables. We take 1500 records as insertion set for in-db rasters, this is roughly the number of images collected by Landsat mission daily.

	Medium cluster	Large cluster
Landsat In-DB 1500 rows	420.840 s	393.451 s
Landsat Out-DB 1500 rows	27.794 s	8.002 s
Landsat Out-DB 10000 rows	37.979 s	17.295 s

Merge Into

We run `MERGE INTO` tasks using various size of dataset on Havasu tables. The datasets used in Merge Into tasks are:

- *0% updates*: all records were taken from the 10% smaller split.
- *5% updates*: 95% records were taken from the 10% smaller split, the other 5% were taken from the larger split. The dataset was shuffled to uniformly distribute the inserted records and updated records.

0% Updates:

	Medium cluster	Large cluster
Landsat In-DB 1500 rows	801.595	668.782 s
Landsat Out-DB 1500 rows	59.456 s	24.301 s
Landsat Out-DB 10000 rows	112.163 s	88.459 s

5% Updates:

	Medium cluster	Medium cluster with 500MB/s EBS	Large cluster	Large cluster with 500MB/s EBS
Landsat In-DB 1500 rows	1038.315 s	499.995 s	730.981 s	406.793 s
Landsat Out-DB 1500 rows	58.804 s	52.146 s	25.573 s	24.757 s
Landsat Out-DB 10000 rows	129.194 s	82.568 s	100.846 s	79.127 s

The performance of Merge Into on in-db datasets hits the EBS throughput bottleneck (125 MB/s). Merge Into involves lots of shuffling, which does lots of disk read and write for in-db rasters. We observed significant performance improvement when using EBS volumes with higher throughput (500 MB/s).

Appendix

1. Spatial range queries used in vector data processing:

Name	Selectivity	Comment	WKT
small	0.5%	Los Angeles	POLYGON((-118.58307129967345 34.31439167411405,-118.6132837020172 33.993916507403284,-118.3880639754547 33.708792488814765,-117.64374024498595 33.43188776025067,-117.6135278426422 33.877700857313904,-117.64923340904845 34.19407205090323,-118.14911133873595 34.35748320631873,-118.58307129967345 34.31439167411405))
medium	1.6%	California	POLYGON((-124.4009 41.9983,-123.6237 42.0024,-123.1526 42.0126,-122.0073 42.0075,-121.2369 41.9962,-119.9982 41.9983,-120.0037 39.0021,-117.9575 37.5555,-116.3699 36.3594,-114.6368 35.0075,-114.6382 34.9659,-114.6286 34.9107,-114.6382 34.8758,-114.5970 34.8454,-114.5682 34.7890,-114.4968 34.7269,-114.4501 34.6648,-114.4597 34.6581,-114.4322 34.5869,-114.3787 34.5235,-114.3869 34.4601,-114.3361 34.4500,-114.3031 34.4375,-114.2674 34.4024,-114.1864 34.3559,-114.1383 34.3049,-114.1315 34.2561,-114.1651 34.2595,-114.2249 34.2044,-114.2221 34.1914,-114.2908 34.1720,-114.3237 34.1368,-114.3622 34.1186,-114.4089 34.1118,-114.4363 34.0856,-114.4336 34.0276,-114.4652 34.0117,-114.5119 33.9582,-114.5366 33.9308,-114.5091 33.9058,-114.5256 33.8613,-114.5215 33.8248,-114.5050 33.7597,-114.4940 33.7083,-114.5284 33.6832,-114.5242 33.6363,-114.5393 33.5895,-114.5242 33.5528,-114.5586 33.5311,-114.5778 33.5070,-114.6245 33.4418,-114.6506 33.4142,-114.7055 33.4039,-114.6973 33.3546,-114.7302 33.3041,-114.7206 33.2858,-114.6808 33.2754,-114.6698 33.2582,-114.6904 33.2467,-114.6794 33.1720,-114.7083 33.0904,-114.6918 33.0858,-114.6629 33.0328,-114.6451 33.0501,-114.6286 33.0305,-114.5888 33.0282,-114.5750 33.0351,-114.5174 33.0328,-114.4913 32.9718,-114.4775 32.9764,-114.4844 32.9372,-114.4679 32.8427,-114.5091 32.8161,-114.5311

			32.7850,-114.5284 32.7573,-114.5641 32.7503,-114.6162 32.7353,-114.6986 32.7480,-114.7220 32.7191,-115.1944 32.6868,-117.3395 32.5121,-117.4823 32.7838,-117.5977 33.0501,-117.6814 33.2341,-118.0591 33.4578,-118.6290 33.5403,-118.7073 33.7928,-119.3706 33.9582,-120.0050 34.1925,-120.7164 34.2561,-120.9128 34.5360,-120.8427 34.9749,-121.1325 35.2131,-121.3220 35.5255,-121.8013 35.9691,-122.1446 36.2808,-122.1721 36.7268,-122.6871 37.2227,-122.8903 37.7783,-123.2378 37.8965,-123.3202 38.3449,-123.8338 38.7423,-123.9793 38.9946,-124.0329 39.3088,-124.0823 39.7642,-124.5314 40.1663,-124.6509 40.4658,-124.3144 41.0110,-124.3419 41.2386,-124.4545 41.7170,-124.4009 41.9983))
large	20%	The US Continent	POLYGON ((-179.99989999978519 16.152429930674884, -179.99989999978519 71.86717445333835, -66.01355466931244 71.86717445333835, -66.01355466931244 16.152429930674884, -179.99989999978519 16.152429930674884))

2. Spatial range queries used in raster data processing.

Name	Selectivity	Comment	WKT
small	0.1%	City in China	POLYGON ((117.72773847796182 37.09045838397545, 117.25060575416128 35.36928107311448, 119.24046766589856 34.983667323083765, 119.76218887831448 36.7041948977658, 117.72773847796182 37.09045838397545))
medium	0.8%	Japan	POLYGON((128.55792392457062 33.87173430238668,136.95147861207062 38.239427626907464,140.3792129870706 45.43041336933314,153.1672989245706 47.84372363138709,152.2005020495706 45.58440134772845,143.1038223620706 41.08054839577082,141.8733536120706 34.19948676248202,133.65558017457062 32.21414487336577,131.01886142457062 28.923021479173585,127.1956192370706 31.654733407521242,128.55792392457062 33.87173430238668))
large	17%	US	POLYGON ((-179.99989999978519 16.152429930674884, -179.99989999978519 71.86717445333835, -66.01355466931244 71.86717445333835, -66.01355466931244 16.152429930674884, -179.99989999978519 16.152429930674884))