REPLY
DATA

REPORT BY DATA REPLY

BENCHMARKING OF BIG DATA TECHNOLOGIES FOR INGESTING AND QUERYING GEOSPATIAL DATASETS

ABSTRACT

Features inherent in geospatial data cause large scale processing to be problematic. For example, searching for matching records (such as checking if a point lies within a polygon) is often computationally expensive.

We benchmark 6[1] of the more prominent Big Data technologies with Geospatial features / add-ons / libraries to assist the interested reader in selecting the right technology for the right workload, along with tips on tuning for performance.

[1] The original scope included a seventh technology (GeoWave). Following some investigation this technology was de-scoped, because - at the time of writing - it does not comply with the GeoJSON data format, and hence could not satisfy the technical requirements set by Dstl for this study.

CONTENT

Introduction	6
Approach	7
Data Generation	9
General Steps	11
GeoSpark	12
Implementation	13
Configuration	15
Results	16
Hive	18
Implementation	19
Configuration	21
Results	22
MongoDB	23
Implementation	24
Configuration	26
Results	27
GeoMesa	28
Implementation	29
Configuration	30
Results	31
Elasticsearch	33
Implementation	34
Configuration	35

CONTENT

Results	36
Postgres-XL	38
Implementation	39
Configuration	41
Results	42
Technology Comparison	45
Simple Queries	46
String Queries	51
Complex Queries	53
Join Queries	56
Recommendations	57
Appendices	58
Appendix 1: Cluster Specification	59
Appendix 2: Hadoop Configuration	60
Appendix 3: HDFS Ingestion	61
Appendix 4: Type handling issue	63
Appendix 5: Queries	64
Appendix 6: Additional Datasets	65
Appendix 7: GeoSpark Vs Magellan	67
Appendix 8: GeoWave Exclusion	67
Appendix 9: GNU Free Documentation License	69

Produced by Data Reply

Commissioned by the Defence Science and Technology Laboratory

Copyright (c) 2017 Reply Ltd.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

REPLY
DATA

INTRODUCTION

INTRODUCTION

Processing geospatial data involves nuances and complications that often don't arise in other domains: geospatial objects are irregular and hence it can be difficult to succinctly describe the corresponding data structures (e.g. points in 2d space representing an island); query operations, such as identifying whether two geospatial objects overlap, are often expensive. As a result, there are many software packages and libraries for the niche data modelling and processing needs of this domain - including in a Big Data context.

We form an objective, evidence-based evaluation of each technology which serves to help guide the reader towards making a more informed decision for their own geospatial technology stack.

How does one effectively select which Big Data technology is most appropriate for the workload in question?

This report tackles this question by benchmarking six different technologies that can be used to work with big geospatial datasets. We have benchmarked under broadly equivalent hardware topology and configuration constraints, with latency being the primary objective metric under study.

1. GEOSPARK
2. HIVE
3. MONGODB
4. GEOMESA
5. ELASTICSEARCH
6. POSTGRES-XL

APPROACH

HIGH LEVEL DATA DESCRIPTION

3 Datasets[2] are in scope:

- **Dataset 1:** A collection of single lat/long points and 7 non-geospatial fields (10 bn records).
- **Dataset 2:** A collection of single lat/long points and 1 text field (10 m records).
- **Dataset 3:** Ellipses and timestamps (10 bn records).

BENCHMARK RULES

A number of rules for this benchmarking study were agreed with DSTL in order to achieve useful results within the project timescales.

DATASETS

In early testing, it became apparent that some technologies in scope would significantly exceed practical time limits for ingestion / indexation / query execution. As a result, we agreed to limit the datasets in scope to two: Dataset 1 and Dataset 3. The rationale for choosing these two datasets is that they respectively represent the simplest and most complex data structures[3]: Dataset 1 is points only; Dataset 3 is 16-point ellipses. In addition, Dataset 2 (required for the Join queries) was also retained in scope.

[2] The initial scope included 5 datasets. 2 datasets were subsequently removed from scope for the reasons mentioned in the BENCHMARK RULES section. Hence from 5 we reduced it to 3: Dataset 1 & Dataset 3 each 10 bn points; and Dataset 2 with 10 m points.

[3] Complexity is based on the number of points defining each structure type.

TIME-OUT PERIOD

Execution times over the following thresholds were agreed to be classed as 'TIMED-OUT'. In those cases in which we could extrapolate based on partial completion time, or based on a previous run with a smaller dataset, we also able to class the execution as TIMED-OUT. This would mean that if a 6 bn run fails, we will not attempt the corresponding 10 bn run. Similarly, if a Dataset 1 query fails, we will not attempt the corresponding Dataset 3 query (because the higher complexity implies it too will fail).

- Query execution: 12 Hours
- Index creation: 24 Hours

It is an important principle of the benchmarking study that we do not rely on a single execution time, as this might be unrepresentative. We agreed that two runs of each query would be executed initially. If the execution time for the 2nd run is not within +/- 33% of run 1, then a third run will also be executed.

GENERATION

In order to facilitate the public distribution of the report DSTL specified pseudo-randomly generated geospatial datasets on which to run the benchmarks. Given the specification outlined by DSTL, we developed bespoke software which would efficiently generate the necessary data in the WGS-84 / GeoJSON format. Items such as lines, polygons, ellipses, points were all generated concurrently across a number of workers with each worker operating its own independent thread pool. We used 20 n1-standard-16 GCE nodes[4] to generate the data in parallel. Given the size of the datasets, each instance of the generation software was set to generate 1 bn elements, with two instances running per VM to ensure full utilization of system resources. Each dataset in the raw data was spread across 5 disks in total except for Dataset 2 which easily fits on to a single disk. Spreading the data across disks allows the ingestion process for components that do not use HDFS to read in parallel without the disk becoming a bottleneck.

TESTING

The code was broken down into FeatureGeneration and EntityGeneration. Features represent fields within a GeoJSON object and an entity represents the full JSON object (i.e. a single row in the file). Two test suites were created to test the feature generation functions in isolation with Property Based Testing as well as the entity generation functions. In addition to this, we performed extensive integration testing on reduced datasets prior to executing each benchmark on the full dataset.

INGESTION / QUERYING

In order to measure the performance of queries[5] with respect to dataset size, we ran them in two batches. For a single dataset at a time, 6 bn elements were ingested and the queries were benchmarked with up to three runs per query. A further 4 bn elements were then ingested and the same queries were executed up to three times per query on the total dataset size. This process was repeated for datasets: 1 & 3.

HDFS INGESTION & BALANCING

Three of the technologies in scope are Hadoop-based, running off data stored in HDFS. Before executing any of the benchmarks on these technologies, we ingested all of the raw data into HDFS. To speed up this process, we ingested across 5 nodes in parallel using the HDFS PUT client. Once HDFS had ingested and replicated all of the data, we ran the HDFS balancer with the default settings. This original run took too long to complete, so we amended the settings as listed below. All of the configurations and timings are listed in Appendix 3 under the HDFS Ingest and HDFS Balancer section.

YARN CONFIGURATION

Please see the ‘YARN Configuration’ section in Appendix 2 for a full detailed overview of all YARN configuration changes. In short, all configuration options were tailored according to the recommended settings from HortonWorks.

INGESTION FOR NON-HDFS TECH

For NON-HDFS technologies namely MongoDB, Elasticsearch & Postgres-XL, we attached the persistent disks containing the data on 6 data nodes. We ingested data across these nodes in parallel to speed up the process and optimally utilize the cluster resources.

[4] See Appendix 1 for cluster specification.
[5] When referring to a specific query for a dataset we use integers 1 to 11. Table 1 in Appendix 5 lists all queries and corresponding IDs. Different runs of the same query are indicated by appending a letter to the query ID (e.g. 3 different runs of query 7 on Dataset 1a are denoted by 7a, 7b, and 7c)

DATA GENERATION

Five datasets were generated to test a variety of types of geospatial data. All five datasets contain fields populated with homogeneous geospatial object types (either points, polygons, or lines), which consist of a set of geospatial points described by numeric longitude and latitude values. In cases where these points are sampled randomly (fields "Latitude" and "Longitude" in datasets 1 and 2, ellipse centre points in dataset 3, polygon location in dataset 4, and starting points of lines in dataset 5), the sampling procedure is adjusted so that resulting points are uniformly distributed on the WGS 84 globe.

In addition to that, the geospatial objects are generated to be consistent with the requirements of the GeoJSON RFC 7946 format (link: https://tools.ietf.org/html/rfc7946, reference: Gillies, Sean, et al. "The GeoJSON Format"). Note that this format requires geospatial objects crossing the anti-meridian to be split into two parts, which individually are on either side of the anti-meridian. Therefore, in order to achieve computational efficiency when generating the data, none of the generated geospatial objects cross the anti-meridian as well as the 0° meridian. This does not affect query performance in any of the technologies.

All geospatial objects are generated on an earth-size spheroid as described by the WGS 84 system. Vincenty's formulae were used for placing geospatial points in order to satisfy the requirements of WGS84 format.

Dataset 1 & 2

Datasets 1 & 2 consisted of simple points (long, lat) pairs with associated metadata and thus did not require any special treatment to conform to WGS 84.

Dataset 1

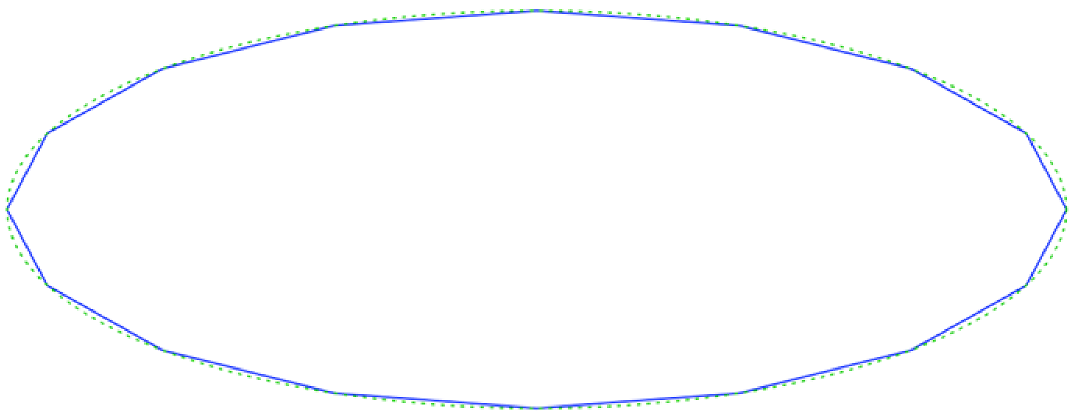
FIELD	COMMENT
Location	Random Point, valid Lat Long e.g. "54.22313 12.234234"
Short_text_field	Single Random Word + optional numbers e.g. "Dog456", "3Cat", "Cow"
Long_text_field_1	Multiple, varying random words (10-200 words) and punctuation e.g. "Dog Cat Fish Cow Horse, Pig...#"
Long_text_field_2	Multiple, varying random words (10-200 words) and punctuation in a random character set e.g. "狗; 猫"
Security_Tag	Randomly picked from "high", "medium" & "low"
Numerical_field_1	Random Integer, e.g. "45"
Numerical_field_2	Random Float, e.g. "4.45646"
Timestamp	Random in last 10 years, e.g. "2007-04-05T12:00:01"

Dataset 2

FIELD	COMMENT
Location	Random Point, valid Lat Long e.g. "54.22313 12.234234"
Short_text_field	Single Random Word + optional numbers e.g. "Dog456", "3Cat", "Cow"

DATASET 3

Since none of the technologies have native support for ellipses, we agreed with DSTL to approximate them with polygons consisting of 16 vertices allocated along the ellipse line, which has an accuracy of ~97.4% (with respect to area). The choice of 16 points was made based on the tradeoff between best fit to ellipse and query efficiency.



The 16 points are placed at specific angles (in degrees) relative to the major axis to ensure that, for a 16-point polygon, the difference between ellipse area and polygon area is minimised.

Based on the requirements specified by DSTL, the random ellipses were generated to have a random centre point, a random major axis of 0.1-10km, a random minor axis of 0.1-2km, and a random orientation.

DATASET 3

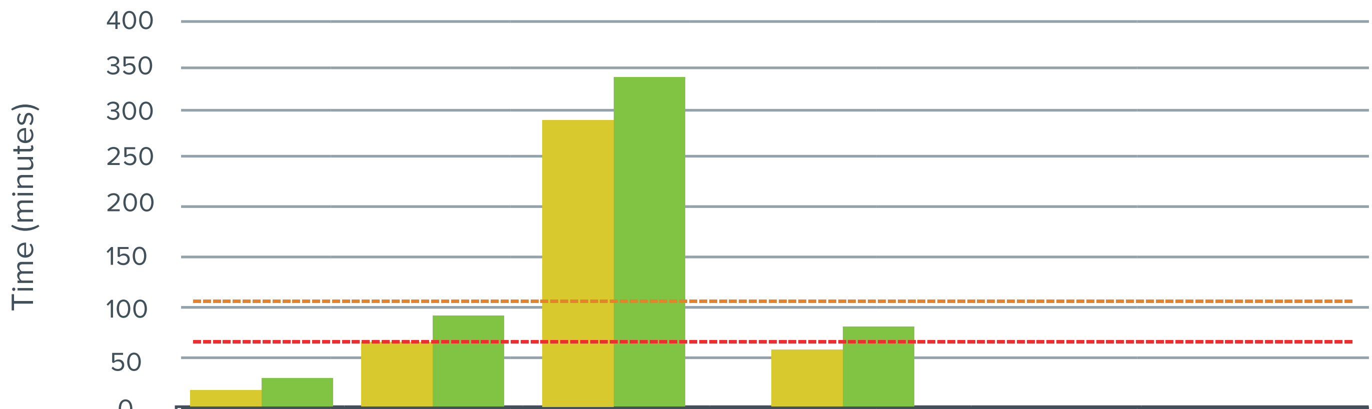
FIELD	COMMENT
Location	A randomly generated ellipse with a random major axis of 0.1-10 km, a random minor axis of 0.1-2 km and a random orientation.
Timestamp	Random in last 10 years, e.g. "2007-04-05T12:00:01"

GENERAL STEPS

The general approach to testing each technology broadly followed the steps below. In subsequent sections we describe the technology-specific implementation and results in more detail.

1. Select and refine cluster topology e.g. deciding on the number of shards and replicas for MongoDB & Elasticsearch, or the number of GTMs, Coordinators and data nodes to be used for Postgres-XL.
2. Setup the infrastructure (installing the technology and configuring it with optimal settings).
3. Ingest Dataset 1 (6-billion-points) with optimal number of threads and batch size.
4. Create indexes for the above ingested points
5. Execute queries corresponding to the ingested dataset.
6. Drop indexes created previously and start ingesting the next 4 billion points for Dataset 1. Once the ingestion completes, Dataset 1 should have 10 billion points in total.
7. Create indexes for 10 billion points.
8. Query execution for the above ingested points.
9. Drop database1 to clear disk space.
10. Steps 3 to 9 are repeated for Dataset 3.
11. Log results and produce cross-technology comparison chart (see below for an example).

Dataset 1 – QUERY 1



	HIVE	GEOSPARK	MONGODB	ELASTICSEARCH	GEOMESA	POSTGRES-XL
6 bn	16.33	60.88	282.50	55.69	0.77	0.77
10 bn	27.85	89.63	338.00	78.83	1.07	-
-- 6 bn Avg	69.49	69.49	69.49	69.49	69.49	69.49
-- 10 bn Avg	107.08	107.08	107.08	107.08	107.08	107.08

BIG DATA TECHNOLOGIES

GEOSPARK

GeoSpark is an open-source geospatial data processing library implemented in JAVA and taking advantage of Apache Spark and JTS Topology Suite. All our work is based on version 0.3.2 of GeoSpark, which at time of writing is freely available at https://github.com/DataSystemsLab/GeoSpark/tree/0.3.2.

Due to the fact that version 0.3.2 of GeoSpark does not give the user control of the storage level (which defaults to memory only), a custom build (as agreed with DSTL) is used with storage level changed to memory and disk (serialised). The source code, which was used for benchmarking, is freely available at https://github.com/DataReplyUK/GeoSpark. Note that the changes made to the original GeoSpark source code are minimal (only hard-coded storage level values were changed), do not affect geospatial implementation behind the queries, and only improve query performance. It is also worth pointing out that without these changes, ingestion in GeoSpark would not be comparable to ingestion in other technologies, due to the fact that generated datasets are too large to fit in memory (except for Dataset 2); the original storage level would result in ingested data being dropped as soon as memory fills up and those data points (including indices build on top of them) would have to be recomputed when a query needs them.

IMPLEMENTATION

This section explains the implementation of GeoSpark benchmark. All parts were implemented using Scala due to its compatibility with Apache Spark.

INGESTION

■ Reported ingestion times consist of the time that it takes to read the GeoJSON files from HDFS, map them into spatial RDDs as used in GeoSpark, and build indices on top of them, while persisting all data structures in memory and disk in a serialised format.

■ A custom GeoJSON parser was written because GeoSpark natively supports only a single GeoJSON format that is different from what we are using.

■ The default R-tree index type was used as it supports the within queries.

■ By default, GeoSpark repartitions the data. However, this causes a significant portion of the data to be sent to the driver (in the case of 10 bn data points of Dataset 1 this would be >60GB), therefore, we opted out of this repartitioning.

■ Since GeoSpark v0.3.2 does not support LineString objects, custom functionality was implemented using Spark RDDs and JAVA Topology Suite (the approach followed by GeoSpark). However, no indices were used in this case because otherwise our implementation would likely not be directly comparable with the native GeoSpark JAVA implementation.

QUERIES

■ For geospatial bounding box queries, native GeoSpark functionality was used.

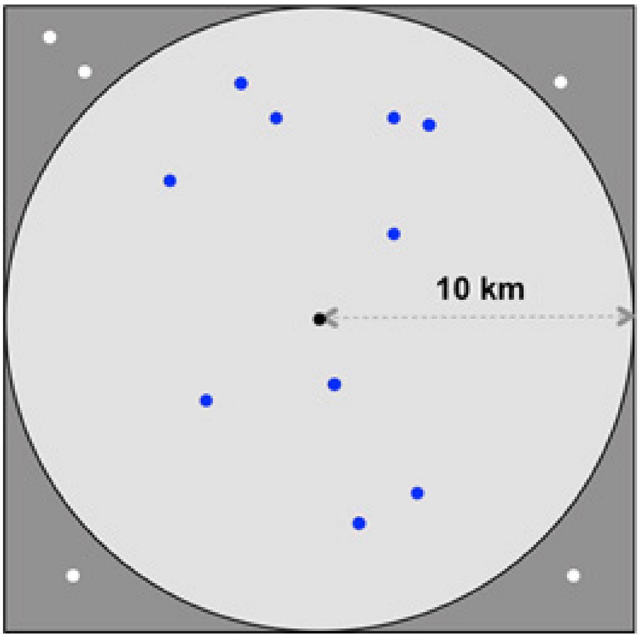
■ In the case of queries involving non-geospatial fields (e.g. query 4), the geospatial part of the query was executed first (using GeoSpark functionality or, if not available, custom implementations), and after that filtering and sorting of results was done by accessing underlying Spark RDDs and using their functionality.

■ Where needed, point-to-point and point-to-polygon distances were calculated using GeographicLib (http://geographiclib.sf.net).

■ The following queries involved implementing custom geospatial functionality, which is not available in GeoSpark 0.3.2:

- **Queries of type:** "within distance from a point" (for example Query 9 & 10 from Dataset 1) are not supported by GeoSpark 0.3.2, therefore, an efficient two-stage approach was taken to execute the geospatial part of these queries:

Stage 1: Execute a bounding box query using a box which bounds a circle of 10 km radius around point1. This stage returns slightly more returns all data points that are within the desired distance and is efficient because the number of results returned from Stage 1 is negligible compared to the whole dataset. Results of this stage are then ordered as desired to produce the final output of these queries.



In the figure above points in the light shaded area are within 10km of point 1 (points coloured blue). All points outside this area, but within the bounding box (i.e. in the dark grey area) are more than 10km from point 1 (points coloured white).

Note that even though writing custom code was required to implement the queries mentioned above, this was done taking into account how missing functionality is likely to be implemented in future releases of this technology.

- **Join Queries:** GeoSpark 0.3.2 does not support this type of query, which requires making a join between two datasets based on distance. A custom implementation was used. All these queries involve Dataset 2, which is quite small compared to others, therefore, this query is implemented as a mapping operation. First of all, the whole Dataset 2 is collected to the driver process, an R-tree index is built on top of it (using the JTS Topology Suite), and then it is broadcast to the worker nodes (this computation and broadcasting is included when measuring query time). Then for each data point from datasets 1 and 3 we search for a point in the broadcast Dataset 2 until the first point satisfying the distance condition is found. All data points in Dataset 1 and 3 for which the distance condition is never satisfied are filtered out and returned.

CONFIGURATION

All jobs were run in client mode, i.e. the driver process was running on one of the master nodes. This choice was made due to resource availability on the master nodes, efficient use of resources on the worker nodes, and fair comparison with other technologies, such as Hive. The table below gives detailed Spark configurations used throughout the benchmark for all queries except the joins.

PARAMETER	VALUE	COMMENT
spark.executor.instances	36	See spark.executor.cores.
spark.executor.memory	17666m	Set to a value which fully utilizes available memory on the worker nodes.
spark.yarn.executor.memoryOverhead	2000m	Set by doing test runs and tracking executor JVM metrics.
spark.executor.cores	4	In combination with spark.executor.instances this value results in 3 executor instances on every worker node and 4 tasks running in parallel within each executor. This value was chosen to maximize IO operations and achieve efficiency of memory usage, and was arrived at by doing a set of tests runs and monitoring their performance.
spark.driver.memory	30g	Set to a high value based on resource availability on the master node.
spark.yarn.driver.memoryOverhead	10000m	Set to a high value based on resource availability on the master node.
spark.driver.cores	5	Set to a high value based on resource availability on the master node.

The table below shows Spark configurations used for the join queries.

PARAMETER	VALUE	COMMENT
spark.executor.instances	12	See spark.executor.cores.
spark.executor.memory	54000m	Set to a value which fully utilizes available memory on the worker nodes.
spark.yarn.executor.memoryOverhead	2000m	Set by doing test runs and tracking executor JVM metrics.
spark.executor.cores	3	In combination with spark.executor.instances this value results in a single executor on every worker node and 3 tasks running in parallel within each executor. We chose to have a single executor per worker node because when broadcasting Dataset 2 it has to be sent to every executor; such configuration results in efficient memory use. A low value of 3 executor cores was chosen because the join queries are memory-intensive and the amount of memory available on the worker nodes is not sufficient to have more tasks running in parallel.
spark.driver.memory	30g	Set to a high value based on resource availability on the master node.
spark.yarn.driver.memoryOverhead	10000m	Set to a high value based on resource availability on the master node.
spark.driver.cores	5	Set to a high value based on resource availability on the master node.

REPLY DATA

RESULTS

INGESTION

■ Ingestion of Dataset 1 and dataset 3 took about the same time in both 6 bn and 10 bn runs, which would imply that complexity of geospatial procedures is the same in both point and polygon (with 16 vertices) cases. However, one should not forget that even though Dataset 1 contains points, it also contains a much larger amount of data (more number of fields) compared to dataset 3. Hence, we conclude that ingest time is about the same due to an unequal amount of data, however ingestion of points is actually more efficient than ingestion of polygons (with 16 vertices) if the other fields are not ignored.

■ Ingestion time increases sublinearly with respect to the number of data points in both datasets 1 and 3.

■ The bottleneck of the ingestion stage is write speed. More than twice as much data has to be written compared to the amount of data read because of caching behaviour (two RDDs are persisted as mentioned in the ingestion implementation section).

DATASET	BATCH SIZE	DURATION
1	6 bn	3 h 45 min 47 sec
1	10 bn	4 h 59 min 24 sec
2	10 m	52 sec
3	6 bn	3 h 36 min 33 sec
3	10 bn	5 h 9 min 41 sec

QUERIES

■ Multiple runs of the same query show very similar performance.

■ Queries that do not involve GeoSpark computations (queries 3, 6, 7, and 8 on Dataset 1) are the most efficient (in terms of running time), and have similar run times within 6 bn and 10 bn runs. However, differences between GeoSpark and Spark-only queries are not so significant because in both cases all data has to be read from disk. The performance of Spark-only queries is also close to being linear with respect to the number of data points due to the fact that non-geospatial data was not indexed.

■ Bounding box queries run substantially faster in the case of Dataset 1 compared to dataset 3, which is the result of the much higher number of points in the polygons. In the case of both datasets these queries scale better than linearly due to indexing. The size of the bounding box does not affect the query time.

■ Similar trends are observed in the "within" queries: they run faster for Dataset 1 due to lower complexity, and they scale better than linearly due to indexing. Also note that sorting does not have a big impact on the run time: in all cases the number of data points, which have to be sorted, is negligible compared to the size of the whole dataset because it is performed after having filtered.

■ The join query was tested only on 6 bn data points of Dataset 1 because it exceeded the 12 h limit (side note: GeoSpark managed to finish the query in 15.4 h).

NOTES

■ GeoSpark required a large amount of storage because for each dataset two RDDs are persisted (in memory and disk). The total amount of disk space required is more than 3 times greater than the size of raw data. The first RDD contains original data after parsing it and transforming into geospatial format, while the second one stores the original data and an index structure built on top of it.

■ The project is still very young and lacks functionality, however custom functionality can be easily implemented using the same programming model (Spark's RDDs + JTS Topology Suite).

■ Query performance is substantially reduced if no geospatial repartitioning is done. Version 0.3.2 is not suited to repartitioning of large geospatial datasets because too much data (1% of the whole dataset) is sent to the driver process.

■ Query performance of GeoSpark is limited by the fact that it does not provide a way to efficiently access individual partitions of a dataset based on index, i.e. the whole dataset has to be pulled out from disk into memory even if it is indexed. This limitation is inherent in Apache Spark, the underlying processing engine, and is unavoidable in technologies – such as GeoSpark – that are based on it.

■ A disadvantage of GeoSpark is that it does not provide a way to index any other data types, except for geospatial data. Hence the queries involving other data types can take longer execution times.

■ Analysts working with GeoSpark would have to be proficient in Scala or JAVA because the technology does not provide a simplified query language or graphical interface.

■ A limitation of GeoSpark is that data has to be ingested and indices rebuilt every time before running queries, and these are lost after the GeoSpark job finishes (unless all data structures are written to disk after initial ingestion and index building, however, this is not yet supported by GeoSpark).

■ Performance is highly sensitive to Apache Spark configuration especially the parameters given in the table in the above section on Configuration.

BIG DATA TECHNOLOGIES

HIVE

Apache Hive is an open-source data warehouse software project built on top of Apache Hadoop for providing data summarization, query, and analysis. Hive gives a SQL-like interface to query data stored in various databases and file systems that integrate with Hadoop.

For all Hive related ingestion and queries, we used the default version (Hive 1.2.1) which ships with Hortonworks HDP 2.5.0. We re-configured the Hive execution engine to use Tez as opposed to MapReduce to improve performance through in-memory processing. ORC file compression was also used to minimize execution time.

REPLY DATA

IMPLEMENTATION

All queries and ingestion steps were performed using the out-of-the-box tools provided by Hive; namely through the Hive shell. A few additional jar files had to be used to support geospatial processing with Hive: Esri-geometry-api, Spatial-sdk-hadoop & JSON-udf-1.3.8-jar-with-dependencies. This is unlike many of the other technologies such as GeoSpark, Elasticsearch and MongoDB where custom code and/or builds were required to ingest/query the data correctly.

Once these jars were added to the classpath, ingestion and queries could be performed with all the standard 'ST functions' provided by ESRI.

INGESTION

■ Reported ingestion times consist of the time that it takes to read the GeoJSON files from HDFS into a temp table (with all fields in a single column) and then select this into a final table with each field correctly separated with ORC compression. This was done to parse the JSON fields.

■ No indexing was performed as it does not function correctly with ORC compression and Hive. The reason for this is ORC. ORC has built in Indexes which allow the format to skip blocks of data during read.

■ Data was balanced on HDFS prior to ingestion.

■ All fields used in the Hive ingestion were typed except for the location field which was initially ingested as a string and transformed to a GeoJSON binary type at query time. This is due to an incompatibility between ORC file formats and ESRI library which was uncovered during integration testing.

QUERIES

■ For the most part the queries could be performed using the native ESRI ST functions.

■ Native support was lacking for queries that need to return geometries 'within x km of a point'. To resolve this, we generated a circle centered at the desired point of radius 10km and used the ESRI functions ST_WITHIN & ST_INTERSECTS. In cases where the result set needed to be ordered by distance, we generated a line from the origin to point1 and calculated the distance in kilometers with ST_GeodesicLengthWGS84. Also, the chines character was not directly supported so we had to use its Unicode character to run query as on the next page.

```
Select * from db1 where long_text_field_2 rlike "\u8FCE";

set point1poly = ST_GeomFromGeoJSON(,{"type":"Polygon","-
coordinates":[[[- 0.14613299999999266,51.58658341116211],[-
0.09090106796740506,51.57973407661179],[-0.04416955758275508,
51.560180075159934],[-0.012996560674274378, 51.53098946442925],
[-0.0021356893840764517,51.4965806115976],[-
0.013196820449947563, 51.46219744386785],[-0.044452769165083575,
51.43306884291964],[-0.09110178009907081,51.41357722154654],[-
0.14613299999999266,51.406753664628894],[-0.2011642199009145,
51.41357722154654],[-0.24781323083490173,51.43306884291964],[-
0.27906917955003774,51.46219744386785],[-0.2901303106159088,
51.4965806115976],[-0.2792694393257109,51.53098946442925],[
0.24809644241723022,51.560180075159934],[-0.20136493203268202,
51.57973407661179],[-0.14613299999999266,51.58658341116211]]]}‘);

Select * from db1
where ST_Within(ST_SetSRID(ST_GeomFromGeoJSON(location),4326),${hivecon-
f:point1poly})
Order BY ST_GeodesicLengthWGS84(ST_SetSRID(ST_LineString(array(ST_Geom-
FromGeoJSON(location), ${hiveconf:point1})), 4326))/1000 ASC;

Select * from db3 where ST_Intersects(ST_SetSRID(ST_GeomFromGeoJSON(loca-
tion),4326), ${hiveconf:point1poly}) OR ST_Within(${hiveconf:point1poly},
ST_SetSRID(ST_GeomFromGeoJSON(location),4326)) OR ST_Within(ST_SetSRID(ST_
GeomFromGeoJSON(location),4326),${hiveconf:point1poly})
ORDER BY ts ASC;
```

Queries that spanned multiple datasets were implemented using a full Cartesian join since no other option is provided in Hive SQL – all of these cases exceeded the agreed timeout period of 12 hours for Hive.

REPLY DATA

CONFIGURATION

HIVE CONFIGURATION

NAME	OLD VALUE	NEW VALUE	COMMENT
Execution engine	Map Reduce	Tez	Switched to Tez execution engine to leverage the large memory available on the cluster.
File Format	Raw / textual	ORC format	Changed to use ORC file format for the reasons outlined at: http://bit.ly/2h2GLax
Reduce vectorization	TRUE	FALSE	Disabled as it is incompatible with ORC file formats: http://bit.ly/2hQHyYS
Tez container size	8gb	19968mb	Increased due to recommendation from Hortonworks documentation
hive.auto.convert.join.noconditionaltask.size	2290649224	5583457484	Increased due to recommendation from Hortonworks documentation

RESULTS

INGESTION

The table below details the times taken to ingest the data into Hive. Note that this does not include the time taken to ingest into HDFS or to rebalance HDFS data.

DATASET	BATCH SIZE	DURATION
1	6 bn	1 h 41 min 56 sec
1	4 bn	1 h 9 min 45 sec
2	10 m	15 sec
3	6 bn	2 h 38 min 37 sec
3	4 bn	1 h 44 min 16 sec

QUERIES

- The results for the basic geo-spatial queries (queries 1-5 of Dataset 1 and 1-4 of datasets 3) show a wide array of results depending on two key factors: dataset size and type of data (point, polygon, ellipse etc.).
- In summary, we see query times for Dataset 1 being the lowest on average and in the best/ worst case. This is likely attributed to the simplicity of the underlying types (points).
- The simplest queries (1 & 2) are both the fastest to execute with an average time of 16.3 and 16.2 minutes respectively. This is only marginally different to queries 4 & 5.
- String queries include any 'string contains' query and are only relevant for Dataset 1. As one might expect, the contains query on the 'short_text_field' performed the best with the remaining two queries ('long_text_field_1' & 'long_text_field_2') taking far longer due to the length of the field being searched. We can also see the re-occurring effect of dataset size on the query runtimes with the average increase being 64.2% across each of the three queries.
- What is clear is the complexity of the object (Point vs. Polygon vs. LineString) and the amount of data present has a direct impact on the query times. This was found to be the case for both simple geo queries as well as complex ones.
- Queries involving joins timed out (exceeding the agreed period of 12 hours). If the magnitude of the data were smaller this might help, but it is ultimately a complexity problem that arises from the fact that Hive only allows for a full Cartesian join (unless you create a new table with both datasets pre-joined).

NOTES

- Other than the lack of native support for certain queries (e.g. within 10 km of a given point), we found Hive to be relatively easy to setup and use. Other than adding the required jars, no additional steps were required.
- It is a mature project with lots of community support. So, in case of any issues, the errors would be quickly resolved.

BIG DATA TECHNOLOGIES

MONGODB

MongoDB is a free and open-source cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schemas.

All work was carried out using version 3.2.11. Queries were for the most part written using the standard DSL for MongoDB – except for special cases such as any queries that required a join. Since queries were written in Javascript and executed via the shell, we were able to add additional logic to avoid a full Cartesian join (unlike Hive) – though we were still unable to get a result within the allotted 12-hour time limit.

REPLY DATA

IMPLEMENTATION

The structure of the MongoDB cluster was setup to include:

- 3 Config Servers (each on a different node).
- 6 Routers/Shards (on 6 of the worker nodes).
- 6 Replica Sets.

Note that shards and replica sets are provided with a full dedicated node rather than being shared. The MongoDB documentation states that shards/replica sets should not be co-located on a single node due to resource contention. We did attempt to do so in order to maximize resource utilization on the cluster, however the shard/replica sets ended up consuming all the memory on each shared node which killed the mongod daemon. Three nodes were also designated as config servers that do not process any data but merely store metadata about the cluster setup. Three config servers were selected to make it comparable with HDFS where we had 3 master nodes.

INGESTION

TYPE HANDLING ISSUE

Refer to 'Type handling issue' in Appendix 4.

The tool was developed in Scala and used Rapture JSON to parse the GeoJSON file along with casbah 3.1.1 to interface with MongoDB.

Ingestion performance was maximized by:

1. Using multiple threads.
2. Using multiple clients.
3. Using Scala futures for asynchronous processing.
4. Processing data in batches rather than using a single request per document.
5. Delaying any indexing until after data has been ingested.

Ingestion was performed by mounting each disk to a given node, and executing the instance with a batch size of 1000 and 12 concurrent threads.

SPLIT CHUNK ISSUE

Once the initial issue with types was solved, MongoDB proved very slow when the size of the datasets exceeded 6 bn elements. We established that once the number of points being ingested reached a sufficiently large number (in our case approximately 4 bn+ elements) performance began to degrade significantly. Upon closer inspection, we found that MongoDB was spending much of the time trying to perform splitChunk() and splitVector() operations which are blocking operations. We mitigated this issue by overriding the default values for chunk size and the initial number of chunks. After further research we also decided to pre-split the data and disable the MongoDB balancer for an added performance boost.

INDEXING

Once the data was ingested, it was indexed using both compound and individual indices where appropriate. Location fields were indexed using the 2dsphere index. Once the first batch of 6 bn elements was ingested and indexed, and the queries had completed, we had to remove the existing indices before ingesting the remaining 4 bn elements for performance reasons[6]. Once all 10 bn elements were ingested, we then re-ran indexing for the whole set.

The table below details the times taken to index the data in MongoDB:

DATASET	BATCH SIZE	INDEX FIELD	DURATION
1	6 bn	Location & Time (Compound Index)	4 h 20 min 6 sec
1	6 bn	Time	2 h 27 min 39 sec
1	4 m	Location & Time (Compound Index)	5 h 48 min 2 sec
2	10 m	Location	14 sec
3	6 bn	Location & Time (Compound Index)	Stopped after 30 h (during which time it had built just 13% of the index).

The index build had to be stopped for Dataset 3; if left to run it would have taken approximately 10 days to complete, well outside our acceptable threshold of 24 Hours.

QUERIES

- Unlike ingestion, the MongoDB queries did not require custom workarounds except for join queries where we needed to write javascript code to join documents from 2 collections.

- Queries were written in .js files and piped to the Mongo Shell for execution.

- We were able to complete the queries using the MongoDB functions: geoNear, geoWithin, nearSphere, geoIntersects, maxDistance & regex.

[6] If the data to be inserted is large it is recommended that you drop the existing index, insert the data and then rebuild indexes. This is done to reduce the overhead of updating indexes on inserting each record which has significant impact on insertion speed.

CONFIGURATION

We made a number of changes to the default MongoDB configurations to adhere to best practices for the size and scope of the cluster. The table below details all the changes that were made to the default configurations.

NAME	OLD VALUE	NEW VALUE	COMMENT
Chunk Size	64mb	1024mb	Required to mitigate blocking calls to splitChunk() / splitVector()
Initial Chunks (per shard)	2	8192	Required to mitigate blocking calls to splitChunk() / splitVector()
Balancer	Enabled	Disabled	Pre-splitting data improved ingestion performance (which was naturally the case for us since it was generated in equally sized partitions)

We also made changes to our default Linux configuration based on recommendations listed in the MongoDB documentation:

NAME	OLD VALUE	NEW VALUE	COMMENT
nproc (processes/threads)	1024	64000	Increases max number of processes in Linux
nofile (open files)	1024	64000	Increases max number of open files in Linux
as (virtual memory size)	-	Unlimited	Increases virtual memory size (MongoDB uses memory mapped files behind the scenes) http://bit.ly/2t1MUFd

RESULTS

INGESTION

■ Ingestion time for Dataset 3 is lower than Dataset 1 as Dataset 3 (2 fields) contains comparatively less fields than Dataset 1 (8 fields).

■ As the MongoDB router has to route the incoming documents to the required data node based on the hash of a unique document_id, the total ingestion time is high compared to other technologies.

The table below details the times taken to ingest the data into MongoDB. Note it doesn't include the indexing time:

DATASET	BATCH SIZE	DURATION
1	6 bn	1 d 21 h 32 min
1	4 bn	2 d 13 h 42 min
2	10 m	9 min
3	6 bn	1 d 16 h 54 min

QUERIES

■ Query execution time was mostly dependent on the result set (documents returned) & the geo-spatial complexity in the dataset (point, polygon, line).

■ For queries using regular expressions the timings were quite high as MongoDB did a COLSCAN instead of IXSCAN (index scan) i.e scanned all the documents for the matching expression.

■ Query 3 for Dataset 1 ran for more than 12 hours and hence was classed as 'TIMED OUT'.

■ Geo-spatial queries with comparatively few results returned (Queries 2, 5, 9 & 10) completed in milliseconds as 2d-sphere indexes were built on the collection. Queries returning significantly more results took more time as MongoDB needs to iterate over each batch of results sequentially as the default batch size is 20.

■ The Join query (query 11 for Dataset 1) was stopped after 5 hours 20 minutes as in that time it had only scanned 2 million records of a total of 6 bn * 10 m records.

NOTES

■ Indexing takes a long time but it has a huge impact for geo-spatial queries. Indexing geo-spatial points took around 5 hours for 10 billion points but for more complex geometries like polygons, the indexes are much slower & can easily take a week depending upon the size of data.

■ When wishing to add data to an existing collection it would be preferable to drop the indexes first and then insert the new data and re-build indexes if the volume of data to be inserted is comparable to the volume of data already present, the reason for this is that MongoDB has to create the index while you insert each document which increases the ingest time significantly.

■ MongoDB is very memory hungry. It often consumes all available memory and can crash the Mongod daemon while running some of the queries as it tries to cache. Also, architectural care needs to be taken with respect to setting up all shards & replicas.

■ The query syntax is rigid, limiting the geo-spatial operations that are possible.

REPLY DATA

BIG DATA TECHNOLOGIES

GEOMESA

GeoMesa is an open-source, distributed, spatio-temporal index built on top of Bigtable-style databases using an implementation of the Geohash algorithm. We use Accumulo as data store as per DSTL's requirement. Leveraging a highly-parallelized indexing strategy, GeoMesa aims to provide as much of the spatial querying and data manipulation to Accumulo as PostGIS does to Postgres.

GeoMesa execution was carried out using version 1.3.1 on top of Accumulo 1.7.0. Queries were for the most part written using the standard CQL. Ingestion was performed using the built-in tools that ship with GeoMesa.

REPLY DATA

IMPLEMENTATION

The structure of the Accumulo cluster was setup to include:

- 12 TServers ("Tablet" servers).
- 3 Master nodes.

INGESTION

- Ingestion was carried out using the ingest tool ('geomesa ingest') that ships with GeoMesa. There were 2 options: local ingestion or MapReduce ingestion. We opted for MapReduce ingestion as recommended by the GeoMesa committer community. The MapReduce ingestion has significant ingestion speed as compared to local ingest as the data is loaded from HDFS.
- Custom converters and schemas were written to parse the GeoJSON file as appropriate. These were added to a single application.conf file.
- The 'GeoMesa Accumulo Distributed Runtime' JAR file that contains server-side code for Accumulo was made available on each of the Accumulo tablet servers in the cluster. These JARs contain GeoMesa code and the Accumulo iterators required for querying GeoMesa data.
- GeoMesa creates 3 separate tables for ingestion of each dataset.

1. The main records table indexed by the feature id.
2. The spatial-only index table. This index will be created if the feature type has the geometry type Point. This is used to efficiently answer queries of features with point geometry with a spatial component but no temporal component.
3. The spatio-temporal index table. This index will be created if the feature type has the geometry type Point and has a time attribute. This is used to efficiently answer queries on data with point geometry with both spatial and temporal components.

QUERIES

- Queries were executed using the out-of-the-box export tools that ship with GeoMesa for all the queries, except those that required sorting which were written in standard CQL format as the documentation suggests.
- Shell scripts were written for executing queries and the results of the query were piped out to a csv file.
- Spark-SQL was used for the queries that required sorting, the join queries & one of the regular expression queries that returned 90% of the data (query 7, 9, 10 & 11 for Dataset 1 & query 5, 6 for Dataset 3). This is because there is no built-in functionality in GeoMesa to carry out sorting. Use of Spark-SQL was recommended by the GeoMesa committer community and also DSTL to carry out sorting and improve performance.
- The 'geomesa-accumulo-spark-runtime' jar file was passed to spark-submit while executing these queries.

REPLY
DATA

CONFIGURATION

GEOMESA/ACCUMULO CONFIGURATION

The table below details the Geomesa/Accumulo configuration used throughout the benchmark.

PARAMETER	VALUE	COMMENT
tserver.scan.files.open.max	500	Maximum total files that all tablets in a tablet server can open for scans. Set to a high value based on the available resources.
tserver.readahead.concurrent.max	32	The maximum number of concurrent read aheads that will execute. Set to a value which fully utilizes available resources on the worker nodes.
tserver.metadata.readahead.concurrent.max	32	The maximum number of concurrent metadata read ahead that will execute. Set to a value, which fully utilizes available resources on the worker nodes.
table.split.threshold	5G	Set this to a higher value so that splits do not occur frequently during ingest
table.scan.max.memory	2G	Set to a higher value to increase amount of memory to be used to cache query results.

SPARK CONFIGURATION

All jobs were run in client mode, i.e. the driver process was running on one of the master nodes. This choice was made due to resource availability on the master nodes, efficient use of resources on the worker nodes, and fair comparison with other technologies, such as Hive.

PARAMETER	VALUE	COMMENT
spark.executor.instances	36	See spark.executor.cores.
spark.executor.memory	17666m	Set to a value which fully utilizes available memory on the worker nodes.
spark.yarn.executor.memoryOverhead	2000m	Set by doing test runs and tracking executor JVM metrics.
spark.executor.cores	4	In combination with spark.executor.instances this value results in 3 executor instances on every worker node and 4 tasks running in parallel within each executor. This value was chosen to maximize IO operations, achieve efficiency of memory usage, and was arrived at by doing a set of tests runs and monitoring their performance.
spark.driver.memory	30g	Set to a high value based on resource availability on the master node.
spark.yarn.driver.memoryOverhead	10000m	Set to a high value based on resource availability on the master node.
spark.driver.cores	5	Set to a high value based on resource availability on the master node.

REPLY
DATA

RESULTS

INGESTION

■ More datapoints were ingested than anticipated because MapReduce retries ingestion of datapoints that fail. A few of the mappers failed but as GeoMesa / Accumulo don't have ACID properties, points are ingested prior to failing. The mapper then tries to reingest, causing some records to be duplicated, and hence more points were ingested than intended.

■ Ingestion time for GeoMesa is higher than technologies like GeoSpark, even though it ingests data from HDFS, because it needs to create 3 separate tables. Hence write speed is the bottleneck of the ingestion stage.

■ Ingestion of Dataset 1 and Dataset 3 took about the same time in both 6 bn and 10 bn runs, which would imply that complexity of geospatial procedures is the same in both point and polygon (with 16 vertices) cases. However, one should not forget that even though Dataset 1 contains points, it also contains a much larger amount of data (more number of fields) compared to Dataset 3. Hence, we conclude that ingest time is about the same due to an unequal amount of data, however ingestion of points is actually more efficient than ingestion of polygons (with 16 vertices) if the other fields are not ignored.

DATASET	BATCH SIZE	DURATION
1	6 bn	27 h 19 min 6 sec
1	10 bn	21 h 50 min 42 sec
2	10 m	8 min 44 sec
3	6 bn	21 h 2 min 58 sec
3	10 bn	23 h 26 min 37 sec

QUERIES

■ Multiple runs of the same query show very similar performance.

■ The query run time was directly proportional to the result set being written to file; hence the bottleneck is writing speed.

■ The performance was linear when moving from a 6 bn run to a 10 bn run of the same dataset.

■ Query 7 was first executed using GeoMesa's export tool which timed out as the result set was approximately 90% of the data (something we knew from the results of the other technologies). The query was then executed using Spark-SQL which performed better compared to the other regular expression queries (query 5 & 6) that were executed using GeoMesa export tool and had comparatively less results. This was because of the in-memory computations done by Spark.

■ Bounding box queries ran substantially faster in the case of Dataset 1 compared to Dataset 3, which is the result of the much higher number of points in the polygons.

■ The join query was tested only on 6 bn data points of Dataset 1 as it could not even complete 1% of MapReduce task in 45 minutes and was termed TIMED OUT as it would not have been able to complete more than 15% in the 12 hours limit.

■ Interestingly, "within" queries took slightly longer for 6 bn rows than 10 bn rows for Dataset 3. Considering the fact that the result set is quite small and the time difference between them is negligible we conclude that the dataset size didn't really affect the performance as it used indexes.

NOTES

■ GeoMesa / Accumulo requires a large amount of storage in order to create 3 separate tables. However, as the tables serve as indexes and you can control which table you require before you ingest, this is comparable to other technologies which require index creation.

■ Also, Accumulo runs bulk insert by writing into small files and then merging these files into bigger files (process called 'Compaction'). Due to Compaction, a huge amount of free HDFS space is needed temporarily during merge. The space is released once the compaction process completes.

■ If Spark SQL is used with GeoMesa libraries, it is directly comparable with Hive ESRI. We would recommend using Spark SQL with GeoMesa based on the results of the queries we ran.

■ The project is under active development and the committers provide good support on gitter's GeoMesa channel. There is a learning curve (e.g. understanding how to write converters & schema for ingestion) but once understood the documentation is clear and helpful.

■ Accumulo's Monitor UI contains a wealth of information about the state of an instance. The Monitor shows graphs and tables which contain information about read/write rates, cache hit/miss rates, and Accumulo table information such as scan rate and active/ queued compactions. The Monitor should always be the first point of entry when attempting to debug an Accumulo problem as it will show high-level problems in addition to aggregated errors from all nodes in the cluster.

BIG DATA TECHNOLOGIES

ELASTICSEARCH

Elasticsearch (ES) is a search engine based on the open source Lucene project. It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents. ES is developed in Java and is released as open source under the terms of the Apache License.

Our ES work was carried out using version 5.1.2. Queries were for the most part written using the standard DSL for ES via REST.

IMPLEMENTATION

The structure of the Elasticsearch cluster was setup to include:

- 24 shards per index.
- 1 replica per shard.
- 3 master nodes.

Note that shards and replica sets were not assigned dedicated nodes as they were with MongoDB. This is because Elasticsearch does not appear to suffer from the same resource contention issues in our experience. The benefit of this is that we are able to achieve full use of the 12 compute nodes.

INGESTION

- Elasticsearch ingestion was similar to that of MongoDB and suffered from the same issues, namely Type Handling Issue and the need to write a custom bulk ingestion tool.
- Before bulk ingestion was started, the replicas were set to 0 so that it does not interfere i.e. consume resources while the ingestion is running & 'merge throttle' was set to none to improve performance as per the documentation.
- Once the bulk ingestion was completed, the replica was set to 1.

TYPE HANDLING ISSUE

Refer to 'Appendix 4: Type Handling issue'.

Ingestion performance was maximized by:

1. Using multiple threads.
2. Using multiple clients.
3. Using Scala futures for asynchronous processing.
4. Processing data in batches rather than using a single request per document.

Actual ingestion was performed by mounting each disk to a given node, and executing the instance with a batch size of 2000 and 4 concurrent threads.

QUERIES

- Unlike ingestion, the ES queries did not require custom workarounds. The queries were written using Query DSL based on JSON and then a GET request was sent to the ES server.
- A Bash script was written that uses ES's Scroll Api in combination with a while loop to get all the results and save them to a file.
- Queries that required sorting by distance in kms were executed without sort as there is no current functionality in ES for this. There is an open issue for this missing feature: http://bit.ly/2sbcH1J.
- The geo_shape data type was used for ingestion & query of geo-spatial fields. In-built functions like 'within' were used for queries.
- Elasticsearch has no functionality to execute cross joins across different indices.

REPLY DATA

CONFIGURATION

NAME	OLD VALUE	NEW VALUE	COMMENT
indices.store.throttle.type	merge	none	While we are doing a bulk import, and don't care about search, we can disable merge throttling entirely. This allows indexing to run as fast as the disks will allow. Setting the throttle type to none disables merge throttling. When we are done importing, we can then set it back to merge to re-enable throttling. http://bit.ly/2rdhUkp
index.refresh_interval	1s	-1	While doing a large import, we can disable refreshes by setting this value to -1 for the duration of the import. Don't forget to re-enable it when you are finished! Done to optimize bulk ingestion. http://bit.ly/2saTs8c
index.number_of_ replicas	2	0	By indexing with zero replicas and then enabling replicas when ingestion is finished, the recovery process is essentially a byte-for-byte network transfer. This is much more efficient than duplicating the indexing process. Once the ingestion is completed this value is set to 1 to have a replica. http://bit.ly/2alQeXb
scroll size	100	2million	Set after trying different values. This parameter allows you to configure the maximum number of hits to be returned with each batch of results.

We also made changes to our default Linux configuration based on recommendations listed in the ES documentation:

NAME	OLD VALUE	NEW VALUE	COMMENT
nproc (processes/ threads)	1024	65536	Increases max number of processes in Linux
nofile (open files)	1024	65536	Increases max number of processes in Linux
vm.max_map_count (virtual memory)	-	262144	Elasticsearch uses a hybrid mmapfs / niofs directory by default to store its indices. The default operating system limits on mmap counts is likely to be too low, which may result in out of memory exceptions. http://bit.ly/2r2OVRh

RESULTS

INGESTION

■ Ingestion of data in temporary tables took more time for dataset 3 as compared to Dataset 1 as there are 16 points in ellipses as compared to just a single point in Dataset 1. Although Dataset 1 has more data fields, the full JSON is inserted as it is, in a single column without parsing.

■ Similar trend is observed in final table where again dataset 3 takes more time than Dataset 1. This might be because of how postgres parses JSON and stories ellipse containing 16-points.

DATASET	BATCH	DURATION
1	6 bn Temp	3 h 50 min 57 sec
1	6 bn Final	10 h 30 min 38 sec
1	10 bn Temp	3 h 57 min 22 sec
1	10 bn Final	2 h 42 min 58 sec
2	10 m Temp	46 sec
2	10 m Final	96 sec
3	6 bn Temp	5 h 3 min 9 sec
3	6 bn Final	15 h 33 min 30 sec

QUERIES

■ Multiple runs of the same query show very similar performance.

■ The query run time was directly proportional to the result set being written to file. Hence the bottleneck is writing speed.

■ The execution of all the queries involving just index-scan on the location field completed in less than a second.

■ Execution of all the other queries (e.g. queries involving just timestamps or regular expressions) took a much longer time to complete.

■ The join query was executed for the 6 bn run of Dataset 1 but exceeded the TIME-OUT period of 12 hours and hence no further join queries were run for other datasets.

■ The query execution time for Dataset 1 & Dataset 3 were very similar because they have similarly sized result sets & GIST indexes are leveraged in both cases.

NOTES

■ The project is still under development and is relatively inactive. Support is almost non-existent. And as there are not many users you might not find many recommendations or tutorials online.

■ As the project has no up-to date binary packages, it required lots of painstaking work to install from source. This involved installing all the dependencies, which was not straightforward as some dependency versions were not directly compatible with the current version of Postgres-XL. For example, we had to manually exclude armadillo package from EPEL release to install the correct version of GDAL. There was a similar issue with the latest stable version of Postgis (2.3.2) being incompatible with the Postrges-XL version (we installed Postgis 2.3.1 to resolve this).

■ There are lots of issues that may make it difficult to use Postgres-XL in production. One such issue is that while changing a table from 'UNLOGGED' to 'LOGGED' the Postgres-XL daemon keeps on waiting as it is trying to acquire a lock on the table (which it isn't able to). A similar issue was encountered when we tried to use 'DROP TABLE' & 'DROP INDEX'.

■ There is no information on the progress of INDEX creation or the INSERT or COPY process, which makes it difficult to anticipate the completion time of the process.

■ The next version of the project (9.6) should bring better query execution performance as it will take advantage of multi-core processing.

BIG DATA TECHNOLOGIES

POSTGRES-XL

Postgres-XL is an all-purpose fully ACID open source scale-out SQL database solution. It aims to provide feature parity with PostgreSQL while distributing the workload over a cluster. The Postgis extension was used to add support for geographic objects allowing location queries to be run in SQL.

Execution was carried out using Postgres-XL 9.5r1.5 (based on PostgreSQL 9.5.6) with Postgis 2.3.1 extension. The Postgres-XL project currently does not provide any up-to date binary packages (rpm, deb, etc.) so it had to be installed from source.

IMPLEMENTATION

The Postgres-XL cluster was setup as follows:

- 3 GTM Servers (each on a different node).
- 6 GTM Proxy, Coordinator & Data Nodes (on 6 of the worker nodes).
- 6 Data Node Slaves.

Three nodes were used as GTM servers to provide consistent transaction management and tuple visibility control. The Coordinator is an interface to the database for applications and is functionally similar to a router in MongoDB. The Coordinator is where the queries are run by the user. The GTM proxy provides proxy features from Postgres-XL Coordinator and Datanode to GTM. GTM proxy groups connections and interactions between GTM and other Postgres-XL components to reduce both the number of interactions and the size of messages.

The Postgres-XL Cluster Control utility, pgxc_ctl, was used to setup the environment and manage the cluster i.e configuration, initialization, starting, stopping, monitoring and failover of Postgres-XL components.

INGESTION

- The ingestion was carried out in 2 stages (similar to Hive). In the first stage, we uploaded the GeoJSON files into a temporary table (staging area) using the inbuilt COPY function.

```
CREATE TABLE IF NOT EXISTS dataset3_temp_w2 (raw_JSON JSONb) TO NODE
(dn1);
ls /mnt/hadoop-w-1/data/D3S4/*.JSON | time xargs -n1 -P10 sh -c "psql -p
30001 -d dstl -c \"\\COPY dataset3 _temp_w2 FROM '\$0' \""
CREATE TABLE IF NOT EXISTS dataset3 (
  geom geometry,
  timestmp timestamp
   ) DISTRIBUTE BY HASH(timestmp);
 INSERT into dataset3  (
  SELECT
   ST_SetSRID(ST_GeomFromGeoJSON(raw_JSON->>'location'), 4326) AS geom,
   to_timestamp(raw_JSON->'prop'->>'timestamp', 'YYYY-MM-DDTHH:MI:SSZ')
AS timestmp
 FROM dataset3 _temp_w2
 );
```

■ Each of the 6 nodes were used for ingestion. The data was divided into 6 folders and each node ingested one folder. 6 temporary tables were created for each dataset for the first stage. The temporary tables were not distributed and each table was stored locally on that node.

■ Once the data is ingested in the temporary tables, the final stage consists of copying the records from the temporary table, parsing the JSON format and extracting the fields into separate columns on the final distributed table.

INDEXING

■ Postgis GIST indexes were created on the location field for all the datasets.

■ To avoid breaching the index time threshold, once the 6 bn run was complete (elements ingested and indexed, and queries completed) we removed the existing indices before ingesting the remaining 4bn elements. Once all 10 bn elements were ingested, we then re-ran indexing for the whole set.

■ The index build had to be stopped for the 10 bn run of Dataset 1 as it exceeded the agreed 24-hour TIME-OUT period. Similarly, as the 6 bn run of Dataset 3 took more than 22 hours, the corresponding 10 bn index job was not attempted.

QUERIES

■ For the most part, queries could be performed using the native Postgis ST functions.

■ In some cases, native support was lacking (e.g. 'within x km of a point'). To resolve this, we generated a circle centered at the desired point of radius 10 km and used the postgis functions ST_WITHIN & ST_INTERSECTS.

CONFIGURATION

POSTGRES-XL CONFIGURATION

NAME	OLD VALUE	NEW VALUE	COMMENT
maintenance_work_mem	16MB	15GB	Increased to speed up Indexing & VACUUM processes.
shared_buffers	32MB	10GB	Determines how much memory is dedicated to PostgreSQL to use for caching data. Tuning this configuration parameter can have a great impact on query times; the default setting is far too low given the amount of memory available on modern servers.
effective_cache_size	4GB	15GB	Increased so that indexes can be saved in memory and for queries an index scan can occur instead of sequential scan.
Tez container size	8GB	19968MB	Increased due to recommendation from Hortonworks documentation

RESULTS

INGESTION

■ Ingestion of data in temporary tables took more time for Dataset 3 as compared to Dataset 1 as there are 16 points in ellipses as compared to just a single point in Dataset 1. Although Dataset 1 has more data fields, the full JSON is inserted as it is, in a single column without parsing.

■ Similar trend is observed in final table where again Dataset 3 takes more time than Dataset 1. This might be because of how postgres parses JSON and stories ellipse containing 16-points.

DATASET	BATCH SIZE	DURATION
1	6 bn Temp	3 h 50 min 57 sec
1	6 bn Final	10 h 30 min 38 sec
1	10 bn Temp	3 h 57 min 22 sec
1	10 bn Final	2 h 42 min 58 sec
2	10 m Temp	46 sec
2	10 m Final	96 sec
3	6 bn Temp	5 h 3 min 9 sec
3	6 bn Final	15 h 33 min 30 sec

QUERIES

■ Multiple runs of the same query show very similar performance.

■ The query run time was directly proportional to the result set being written to file. Hence the bottleneck is writing speed.

■ The execution of all the queries involving just index-scan on the location field completed in less than a second.

■ Execution of all the other queries (e.g. queries involving just timestamps or regular expressions) took a much longer time to complete.

■ The join query was executed for the 6 bn run of Dataset 1 but exceeded the TIME-OUT period of 12 hours and hence no further join queries were run for other datasets.

■ The query execution time for Dataset 1 & Dataset 3 were very similar because they have similarly sized result sets & GIST indexes are leveraged in both cases.

NOTES

- The project is still under development and is relatively inactive. Support is almost non-existent. And as there are not many users you might not find many recommendations or tutorials online.

- As the project has no up-to date binary packages, it required lots of painstaking work to install from source. This involved installing all the dependencies, which was not straightforward as some dependency versions were not directly compatible with the current version of Postgres-XL. For example, we had to manually exclude armadillo package from EPEL release to install the correct version of GDAL. There was a similar issue with the latest stable version of Postgis (2.3.2) being incompatible with the Postrges-XL version (we installed Postgis 2.3.1 to resolve this).

- There are lots of issues that may make it difficult to use Postgres-XL in production. One such issue is that while changing a table from 'UNLOGGED' to 'LOGGED' the Postgres-XL daemon keeps on waiting as it is trying to acquire a lock on the table (which it isn't able to). A similar issue was encountered when we tried to use 'DROP TABLE' & 'DROP INDEX'.

- There is no information on the progress of INDEX creation or the INSERT or COPY process, which makes it difficult to anticipate the completion time of the process.

- The next version of the project (9.6) should bring better query execution performance as it will take advantage of multi-core processing.

REPLY DATA

BIG DATA TECHNOLOGIES

TECHNOLOGY COMPARISON

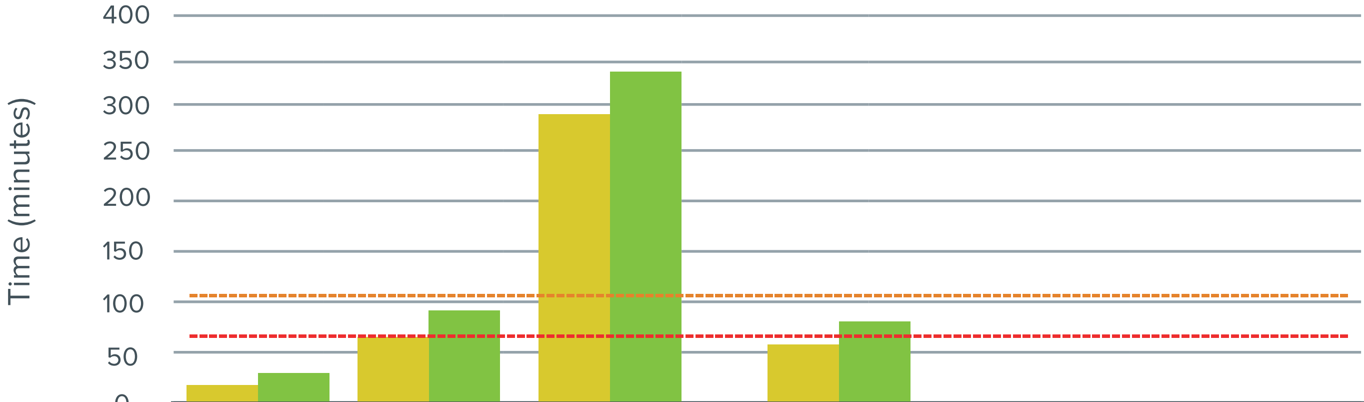
REPLY DATA

SIMPLE QUERIES

This section compares the performance of all technologies under benchmark for the simple queries. Simple queries include those which perform bbox, intersects & time range operations. For Dataset 1 this includes queries 1-5 and 1-4 for dataset 3.

Note on data tables beneath each chart: The query execution times are rounded to 2 decimal places for all the graphs below. Note therefore that queries taking less than 0.005 seconds for execution are rounded to 0.00. Note also that queries that were not run or failed, appear as null ("-").

Dataset 1 – QUERY 1

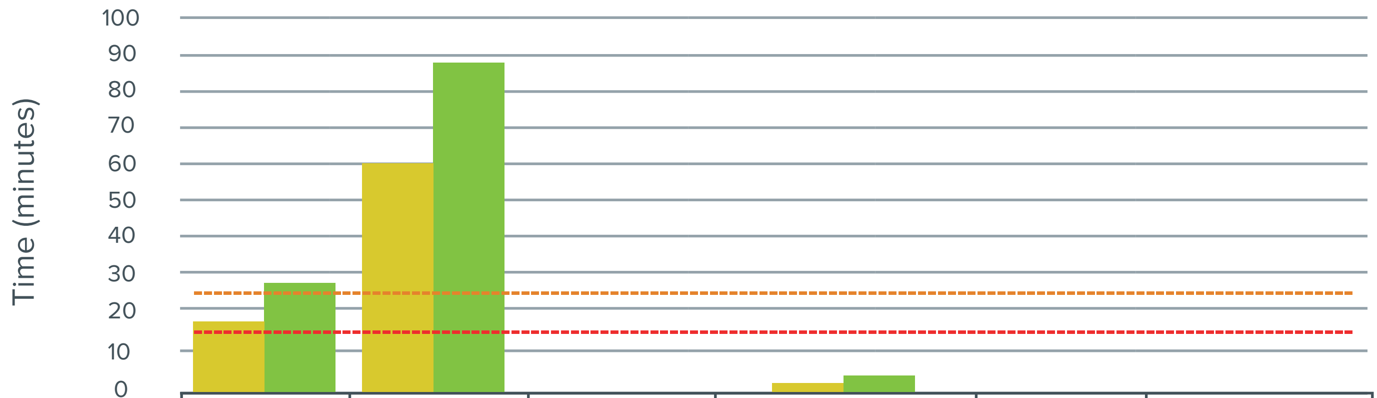


	HIVE	GEOSPARK	MONGODB	ELASTICSEARCH	GEOMESA	POSTGRES-XL
6 bn	16.33	60.88	282.50	55.69	0.77	0.77
10 bn	27.85	89.63	338.00	78.83	1.07	-
-- 6 bn Avg	69.49	69.49	69.49	69.49	69.49	69.49
-- 10 bn Avg	107.08	107.08	107.08	107.08	107.08	107.08

Select * from dataset1 where dataset1.geo is within bbox1

Datase 1 query 1 involves returning all the points that are within bounding-box1 which is the size of UK. GeoMesa & Postgres-XL were the fastest to complete the execution of this query. This was due to the fact that they were using GeoSpatial indexes and returning full results in a batch. Hive on the other hand had to scan all the results but as it uses ORC format and Tez engine it performs better without indexes. MongoDB & Elasticsearch also used indexes but as they output results & write them to a file using scroll apis it takes longer (depending on the results they return per batch) compared to GeoMesa or Postgres-XL. GeoSpark also performs well as it does not have to scroll through the results and uses R-tree indexes.

Dataset 1 – QUERY 2



	HIVE	GEOSPARK	MONGODB	ELASTICSEARCH	GEOMESA	POSTGRES-XL
6 bn	16.19	60.24	0.00	1.35	0.11	0.00
10 bn	27.50	86.74	0.00	3.29	0.12	-
6 bn Avg	12.98	12.98	12.98	12.98	12.98	12.98
10 bn Avg	23.53	23.53	23.53	23.53	23.53	23.53

Select * from dataset1 where dataset1.geo is within bbox2

Dataset 1 query 2 involves returning all the points that are within bounding-box2 which is the size of Hyde Park. Despite this reduced size, Hive still needs to scan the full table, whereas the other technologies use indexes and hence, given far fewer points to lookup, take far less time to return a result. GeoSpark is an exception because for each query the data needed to be ingested and indexed.

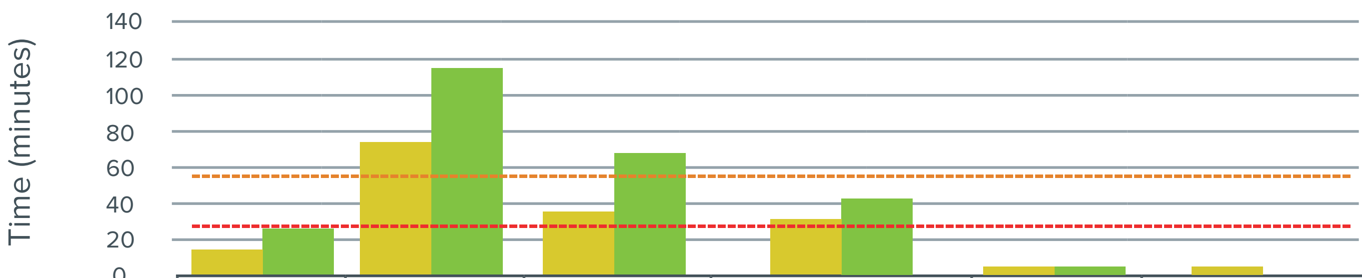
Dataset 1 – QUERY 3

	HIVE	GEOSPARK	ELASTICSEARCH	GEOMESA	POSTGRES-XL
6 bn	26.30	43.90	41.83	213.67	255.75
10 bn	44.09	62.27	47.80	344.67	-
6 bn Avg	116.29	116.29	116.29	116.29	116.29
10 bn Avg	124.71	124.71	124.71	124.71	124.71

Select * from Dataset 1 where Dataset 1.timestamp < time1 and Dataset 1.timestamp > time2

Dataset 1 query 3 involves returing all the points that are within a given time period. MongoDB timed out for this query so it is not included in the graph. Similarly as the execution for Dataset 1 10 bn was not carried out for Postgres-XL, it is also not included in the graph. These queries don't involve geo-spatial computation, and there were no indexes created on timestamps (except for Elasticsearch). GeoMesa & Postgres-XL were slow; their execution time exceeded the average across all technologies. This is because these technologies needed to do a full scan of the table. Hive & GeoSpark leverage in-memory computation so they are relatively fast when a full scan is necessary. Elasticsearch builds indexes for timestamps, hence it performed better than other

Dataset 1 – QUERY 4

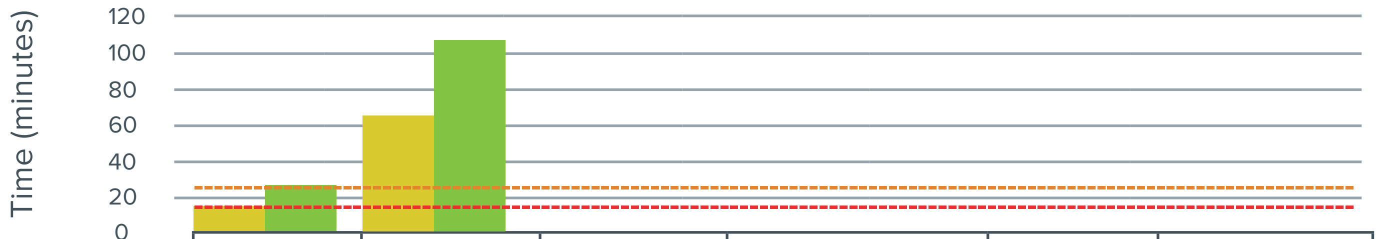


	HIVE	GEOSPARK	MONGODB	ELASTICSEARCH	GEOMESA	POSTGRES-XL
6 bn	16.80	72.91	36.38	32.38	1.26	0.75
10 bn	28.11	116.69	71.68	44.61	0.69	-
6 bn Avg	26.75	26.75	26.75	26.75	26.75	26.75
10 bn Avg	52.36	52.36	52.36	52.36	52.36	52.36

Select * from Dataset 1 where Dataset 1.geo is within bbox1 and Dataset 1.timestamp < time1 and Dataset 1.timestamp > time2

Dataset 1 query 4 is a geo-temporal query that involves returning records that are within bounding-box1 and within a given time period. GeoMesa builds geo-temporal indexes, hence the execution time is very low. This query uses the same bounding box as Query 1, which filters the records. The relative performance of the other technologies is in line with Dataset 1 Query 1. This is not true for GeoSpark as it checks for both conditions (within Bounding Box1 and timestamp) and does not filter out results with bounding box1 first and then check for timestamp condition on the filtered results like the other technologies.

Dataset 1 – QUERY 5

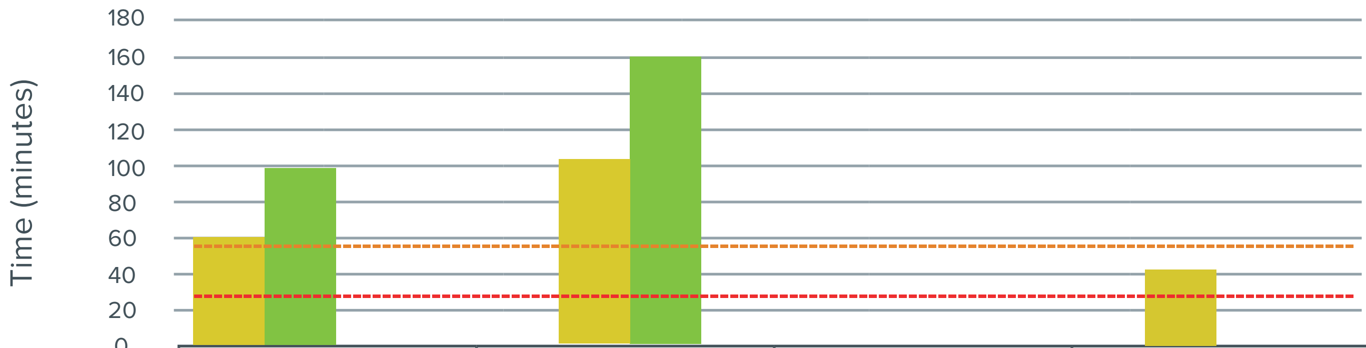


	HIVE	GEOSPARK	MONGODB	ELASTICSEARCH	GEOMESA	POSTGRES-XL
6 bn	16.95	70.03	0.00	0.12	0.20	0.19
10 bn	28.00	105.59	0.00	1.21	0.45	-
6 bn Avg	14.58	14.58	14.58	14.58	14.58	14.58
10 bn Avg	27.05	27.05	27.05	27.05	27.05	27.05

Select * from Dataset 1 where Dataset 1.geo is within bbox2 and Dataset 1.timestamp < time1 and Dataset 1.timestamp > time2

Dataset 1 query 5 performs similarly to Dataset 1 Query 2 given that it has similar bounding box conditions.

DATASET 3 – QUERY 1

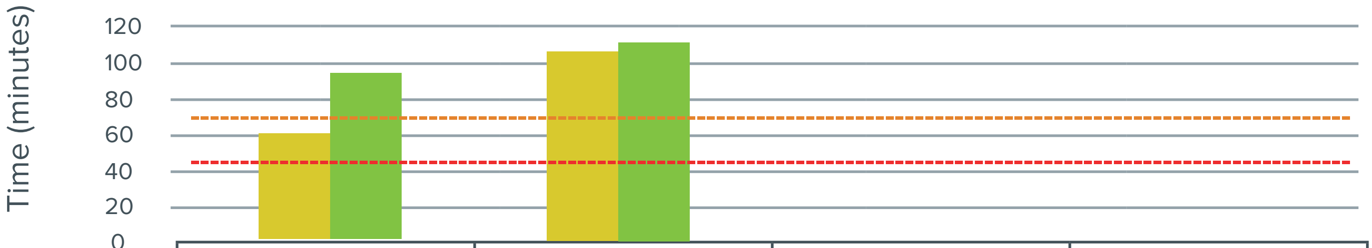


	HIVE	GEOSPARK	GEOMESA	POSTGRES-XL
6 bn	60.73	104.37	1.27	44.35
10 bn	99.84	159.35	2.17	-
6 bn Avg	52.68	52.68	52.68	52.68
10 bn Avg	87.12	87.12	87.12	87.12

Select * from dataset3 where dataset3.geo intersects bbox1

Dataset 3 Query 1 performs similarly to Dataset 1 Query 1 as they have similar geo-spatial complexity. This Query was not run for MongoDB because the indexes timed out nor & Elasticsearch because the ingestion timed out (see respective technology sections above).

DATASET 3 – QUERY 2



	HIVE	GEOSPARK	GEOMESA	POSTGRES-XL
6 bn	60.53	109.52	0.14	0.00
10 bn	97.80	110.54	0.13	-
-- 6 bn Avg	42.55	42.55	42.55	42.55
-- 10 bn Avg	69.49	69.49	69.49	69.49

Select * from dataset3 where dataset3.geo intersects bbox2

Dataset 3 Query 2 performs similarly to Dataset 1 Query 2 as they have similar geo-spatial complexity. This Query was not run for MongoDB because the indexes timed out nor & Elasticsearch because the ingestion timed out (see respective technology sections above).

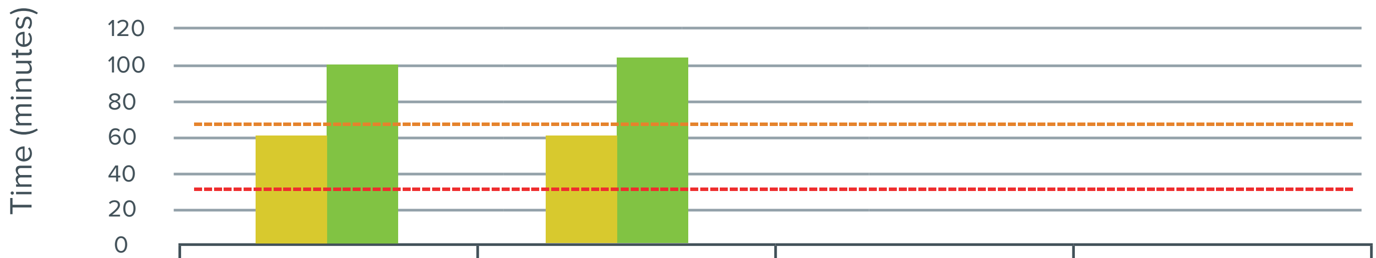
DATASET 3 – QUERY 3

	HIVE	GEOSPARK	GEOMESA	POSTGRES-XL
6 bn	62.01	62.10	4.03	0.36
10 bn	101.46	98.23	3.28	-
-- 6 bn Avg	32.13	32.13	32.13	32.13
-- 10 bn Avg	67.66	67.66	67.66	67.66

Select * from dataset3 where dataset3.geo intersects bbox1 and dataset3.timestamp < time1 and dataset3.timestamp > time2

Dataset 3 Query 3 performs similarly to Dataset 1 Query 4 as they have similar geo-spatial complexity. This Query was not run for MongoDB because the indexes timed out nor & Elasticsearch because the ingestion timed out (see respective technology sections above).

DATASET 3 – QUERY 4



	HIVE	GEOSPARK	GEOMESA	POSTGRES-XL
6 bn	61.41	58.80	0.82	0.00
10 bn	100.35	114.85	1.04	-
6 bn Avg	30.26	30.26	30.26	30.26
10 bn Avg	72.08	72.08	72.08	72.08

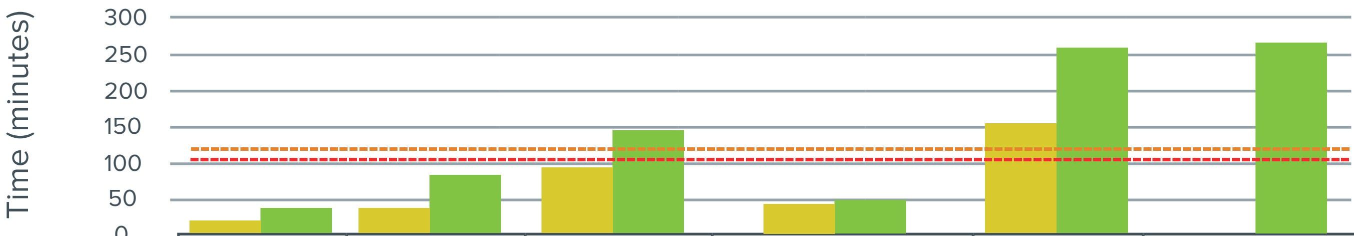
Select * from dataset3 where dataset3.geo intersects bbox2 and dataset3.timestamp < time1 and dataset3.timestamp > time2

Dataset 3 Query 4 performs similarly to Dataset 1 Query 5 as they have similar geo-spatial complexity. This Query was not run for MongoDB because the indexes timed out nor & Elasticsearch because the ingestion timed out (see respective technology sections above).

STRING QUERIES

This section compares the performance of all technologies for string queries, i.e. Queries 6, 7 and 8 for Dataset 1.

Dataset 1 – QUERY 6

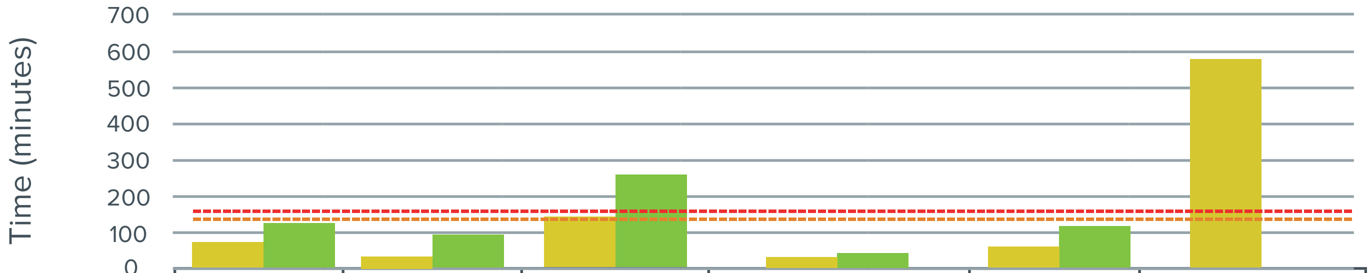


	HIVE	GEOSPARK	MONGODB	ELASTICSEARCH	GEOMESA	POSTGRES-XL
6 bn	21.61	43.41	93.54	44.24	157.31	261.87
10 bn	35.84	84.20	148.05	49.23	254.23	-
-- 6 bn Avg	103.66	103.66	103.66	103.66	103.66	103.66
-- 10 bn Avg	114.31	114.31	114.31	114.31	114.31	114.31

Select * from dataset1 where Short_text_field contains "dog"

Dataset 1 Query 6 is a regular expression query. Elasticsearch performs better than most of the technologies, which is unsurprising given it is a Lucene-based search engine. The performance for MongoDB, GeoMesa & Postgres-XL was directly proportional to the size of the result set; accordingly, these technologies take longer to execute queries as the result of this query is almost 1/6th of the total dataset size. As this query involves a full table scan for all technologies, we see that Hive outperforms the rest.

Dataset 1 – QUERY 7

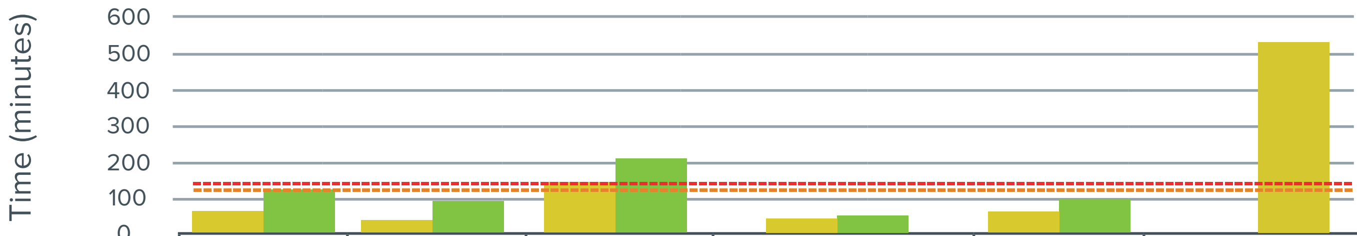


	HIVE	GEOSPARK	MONGODB	ELASTICSEARCH	GEOMESA	POSTGRES-XL
6bn	72.65	46.85	128.06	45.25	69.52	584.41
10bn	119.40	98.63	255.26	49.51	111.07	-
--- 6bn Avg	157.79	157.79	157.79	157.79	157.79	157.79
--- 10bn Avg	126.78	126.78	126.78	126.78	126.78	126.78

Select * from dataset1 where Long_text_field_1 contains "dog"

Dataset 1 Query 7 is another regular expression query that performs similarly to Dataset 1 Query 6. The exception is GeoMesa for which we used Spark-SQL (with GeoMesa libraries) to perform this query instead of the inbuilt GeoMesa export tool. Hence, execution performance is better than for Query 6 (it was not used for Query 6 because where possible we were using in-built tools). As the field this query looks up contains much more text, Elasticsearch performs better in this case relative to Hive (compare Dataset 1 Query 6).

Dataset 1 – QUERY 8



	HIVE	GEOSPARK	MONGODB	ELASTICSEARCH	GEOMESA	POSTGRES-XL
6 bn	71.43	40.90	156.40	42.76	65.03	527.95
10 bn	116.13	91.48	211.28	49.54	103.50	-
-- 6 bn Avg	150.75	150.75	150.75	150.75	150.75	150.75
-- 10 bn Avg	114.39	114.39	114.39	114.39	114.39	114.39

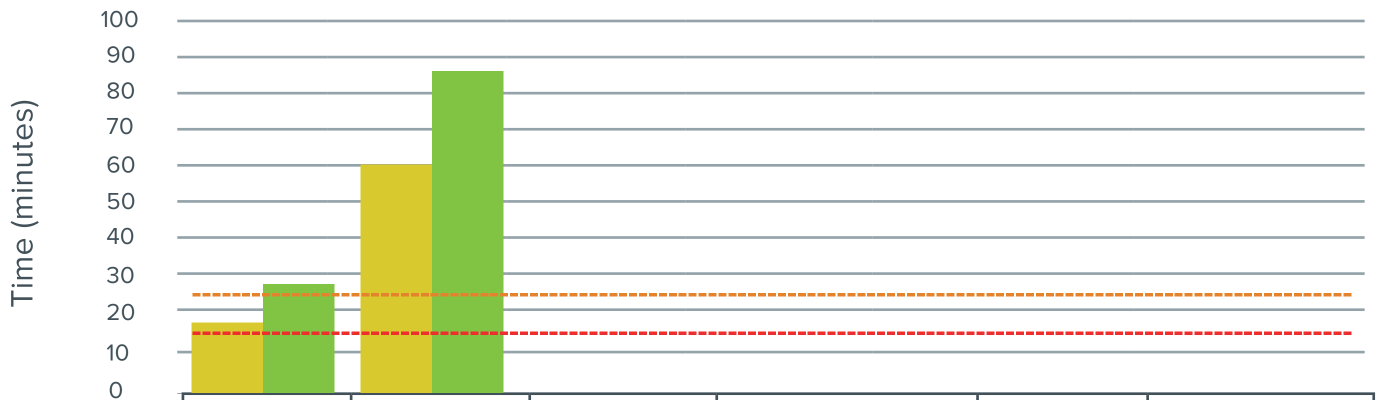
Select * from dataset1 where Long_text_field_2 contains "迎"

The performance for Dataset 1 query 8 is similar to that of Dataset 1 query 7.

COMPLEX QUERIES

This section compares the performance of all technologies for complex queries. These include queries 9-11 for Dataset 1 and 5-7 for dataset 3.

Dataset 1 – QUERY 9

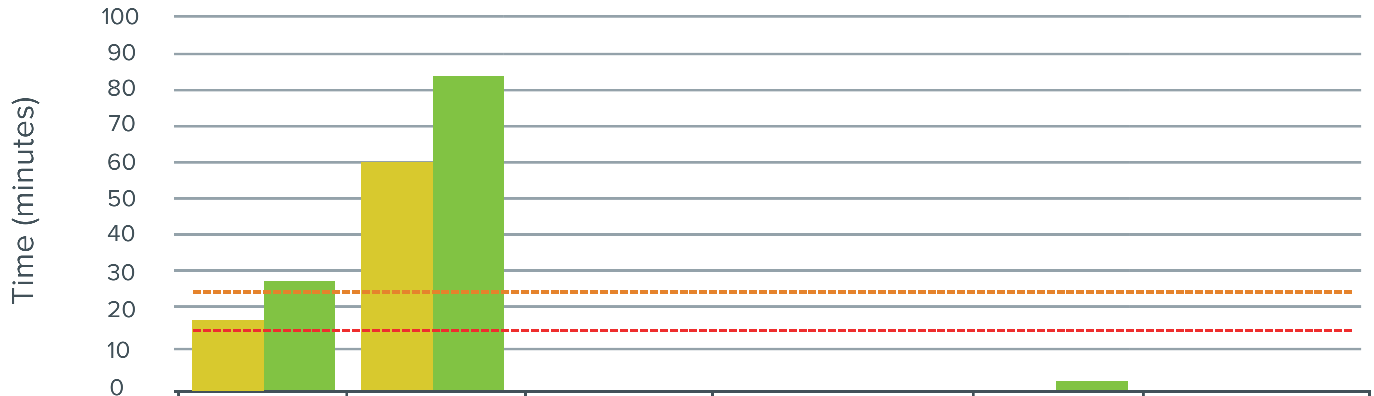


	HIVE	GEOSPARK	MONGODB	ELASTICSEARCH	GEOMESA	POSTGRES-XL
6 bn	14.96	59.67	0.00	0.29	0.87	0.00
10 bn	28.22	85.06	0.54	0.39	1.04	-
-- 6 bn Avg	12.63	12.63	12.63	12.63	12.63	12.63
-- 10 bn Avg	23.05	23.05	23.05	23.05	23.05	23.05

Select * from dataset1 where dataset1.geo is within 10km of point1, order by distance to point1

Dataset 1 query 9 involves returning all the points that are within 10 km of point1. Again due to the geo-spatial indexes and the fact that the execution time is directly proportional to the size of result set for MongoDB, Elasticsearch & Postgres-XL, these technologies complete the execution of the query within seconds.

Dataset 1 – QUERY 10



	HIVE	GEOSPARK	MONGODB	ELASTICSEARCH	GEOMESA	POSTGRES-XL
6 bn	17.22	59.98	0.00	0.31	0.84	0.00
10 bn	27.98	83.79	0.00	0.53	0.88	-
-- 6 bn Avg	13.06	13.06	13.06	13.06	13.06	13.06
-- 10 bn Avg	22.64	22.64	22.64	22.64	22.64	22.64

Select * from dataset 1 where dataset1.geo is within 10km of point1, order by timestamp

Dataset 1 query 10 performs similar to Dataset 1 query 9 as fundamentally the query are same, except the sort field.

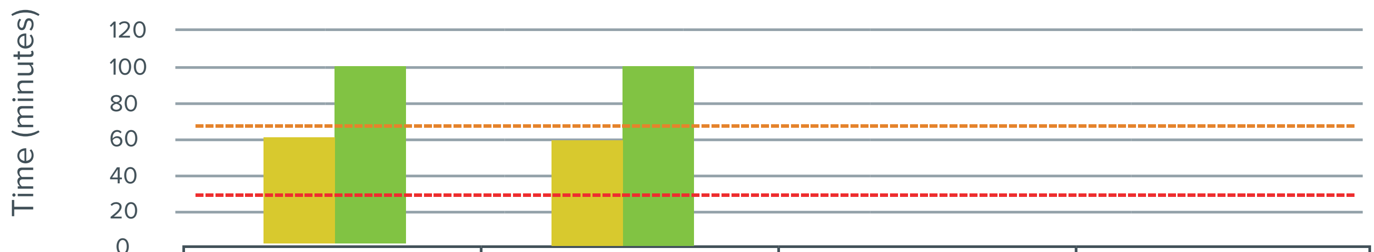
DATASET 3 – QUERY 5

	HIVE	GEOSPARK	GEOMESA	POSTGRES-XL
6 bn	63.47	84.96	1.19	0.00
10 bn	104.72	93.22	0.88	-
-- 6 bn Avg	37.41	37.41	37.41	37.41
-- 10 bn Avg	66.27	66.27	66.27	66.27

Select * from dataset3 where dataset3.geo is within 10km of point1, order by closest distance to point1

Dataset 3 query 5 performs similar to Dataset 1 query 9 as they have similar geo-spatial complexity for all the technologies execpt MongoDB & Elasticsearch for which Dataset 3 query execution was not carried out.

DATASET 3 – QUERY 6



	HIVE	GEOSPARK	GEOMESA	POSTGRES-XL
6 bn	63.01	60.93	1.00	0.00
10 bn	103.94	101.11	0.82	-
6 bn Avg	31.23	31.23	31.23	31.23
10 bn Avg	68.62	68.62	68.62	68.62

Select * from dataset3 where dataset3.geo is within 10km of point1, order by timestamp

Dataset 3 query 6 performs similar to Dataset 1 query 10 as they have similar geo-spatial complexity for all the technologies execpt MongoDB & Elasticsearch for which Dataset 3 query execution was not carried out.

JOIN QUERIES

JOIN QUERIES

This section shows whether the technology was able to execute joins within the TIME-OUT period or not.

TECHNOLOGY	DATASET 1	DATASET 3
Hive	FAILED	FAILED
GeoSpark	FAILED (Completed in 15 hrs)	FAILED
MongoDB	FAILED	FAILED
Elasticsearch	FAILED	FAILED
GeoMesa	FAILED	FAILED
Postgres-XL	FAILED	FAILED

RECOMMENDATIONS

■ If the dataset is updated after a long-time period (e.g. weeks or months) and is frequently queried, technologies like Elasticsearch that use indexing may offer better performance in terms of query execution time. As these technologies require re-indexing when ingesting new/changed data, frequent data updates (insert or update) would become a bottleneck and indexing would be very slow. For such use cases, Hive & GeoMesa are likely to outperform others. GeoMesa also uses indexes but they are created on-the-fly as separate tables during ingestion.

■ When selecting between Hive & GeoMesa, bear in mind that GeoMesa would require double or triple the disk space (depending on the indexes you want) as compared to Hive.

■ The complexity of setting up and configuring infrastructure may be a consideration for some organisations. Elasticsearch & Hive worked out of the box and were easy to setup and required the least configuration changes. MongoDB & Postgres-XL were difficult to setup and required lots of configuration decisions and tuning. The others were somewhere in between.

■ GeoMesa had the best community support; the committers were available (on gitter) to help most of the time. Postgres-XL had the least to no community support.

■ Elasticsearch (used with Kibana & x-pack) and Hive (Tez & Ambari UI) offer very user-friendly monitoring tooling, so tasks such as monitoring the cluster, exporting logs and monitoring query execution are greatly simplified. Tez UI even shows the percentage execution for running queries.

■ Almost all of the technologies have auto failover except for Postgres-XL where a manual command is needed.

■ For technologies involving Spark performance is, as one would expect, highly sensitive to Spark configuration - especially the parameters given in the table in Spark Configuration (in GeoSpark section).

To aid the reader we provide below a summary table comparing the relative strengths of each technology against 4 selection criteria. They are each rated from "High" (most positive) to "Low" (least positive).

TECHNOLOGY	EASE OF SETUP	DEVELOPER COMMUNITY	EASE OF IMPLEMENTING GEOSPATIAL QUERIES	MONITORING TOOLS
Hive	High	Medium	High	High
GeoSpark	Medium	Medium	Medium	Medium
MongoDB	Low	Medium	Medium	Low
Elasticsearch	High	Medium	High	High
GeoMesa	Medium	High	High	Medium
Postgres-XL	Low	Low	High	Low

ENCHMARKING OF BIG DATA TECHNOLOGIES FOR INGESTING AND QUERYING GEOSPATIAL DATASETS

APPENDICES

APPENDIX 1: CLUSTER SPECIFICATION

- **Data Generation and storage:** 20 persistent disks with 1500 GB disk space.
- **HDFS:**
  - 15 n1-standard-16 (16 vCPUs, 60 GB memory) VMs with 20 GB Boot disk.
  - 12 Data nodes are attached with 5750 GB persistent disk each.
  - 3 Master nodes are attached with 1000 GB persistent disk each.
- **Non-HDFS:**
  - 15 n1-standard-16 (16 vCPUs, 60 GB memory) VMs with 20 GB Boot disk.
  - 12 Data nodes are attached with 5750 GB persistent disk each.
  - 3 Master nodes are attached with 1000 GB persistent disk each.
- **OS: CentOS 6**

REPLY DATA

APPENDIX 2: HADOOP CONFIGURATION

HDFS CONFIGURATION

PARAMETER	VALUE
Hadoop max heap size	8GB
Block replication	2
NameNode New Gen min/max	384/384
NameNode PermGen min/max	128/256
DataNode Heap	3GB
NameNode Heap	1GB

YARN CONFIGURATION

PARAMETER	VALUE
Total memory for all containers	59904mb
Min/Max container memory	19968mb/59904mb
Min/Max container cores	1/12
RM, NM & YARN max heap	1GB
Scheduler	org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler

HIVE & TEZ ADDITIONAL CONFIGURATION

PARAMETER	VALUE
ORC stripe size	64MB
Compression	zlib
HiveServer2 heap size	22.114GB
Metastore heap size	7548MB
Client heap size	1024MB
Map join memory per map	2184.5MB
Data per reducer memory	64MB
Enable cost based optimizer	True
Resource memory	8GB
Task resource	8GB

MAPREDUCE CONFIGURATION

PARAMETER	VALUE
Map/Reduce virtual memory	19968mb/19968mb
Sort allocation memory	7987mb
AppMaster virtual memory	19968mb
Map max heap size	15974mb
Reduce max heap size	15974mb
AppMaster max heap size	15974mb

REPLY DATA

APPENDIX 3: HDFS INGESTION

HDFS PUTS

DATASET/BATCH	TIME	TYPE	MINUTES	SECONDS	TIME
1 (4 bn)	real 1265m33.229s	Real	1,265	33	1,265.6
	user 77m48.591s	User	77	49	77.8
	sys 65m19.105s	Sys	65	19	65.3
1 (6 bn)	real 1520m16.092s	Real	1,520	16	1,520.3
	user 109m22.649s	User	109	23	109.4
	sys 93m56.159s	Sys	93	56	93.9
2 (10 m)	real 0m42.708s	Real	0	43	0.7
	user 0m9.459s	User	0	9	0.2
	sys 0m2.179s	Sys	0	2	0.0
3 (4 bn)	real 1321m21.373s	Real	1,321	21	1,321.4
	user 71m58.891s	User	71	59	72.0
	sys 54m35.304s	Sys	54	35	54.6
3 (6 bn)	real 1779m27.241s	Real	1,779	27	1,779.5
	user 104m57.757s	User	104	58	105.0
	sys 82m21.506s	Sys	82	22	82.4
4 (4 bn)	real 705m31.785s	Real	705	32	705.5
	user 38m6.213s	User	38	6	38.1
	sys 27m15.464s	Sys	27	15	27.3
4 (6 bn)	real 970m45.640s	Real	970	46	970.8
	user 56m7.277s	User	56	7	56.1
	sys 41m7.965s	Sys	41	8	41.1
5 (4 bn)	real 1015m18.092s	Real	1,015	18	1,015.3
	user 57m6.566s	User	57	7	57.1
	sys 43m38.762s	Sys	43	39	43.6
5 (6 bn)	real 1401m50.858s	Real	1,401	51	1,401.8
	user 83m34.186s	User	83	34	83.6
	sys 65m20.837s	Sys	65	21	65.3

HDFS BALANCER

RUN	RUNTIME	PARAMETERS
1	real 278m1.033s	Default
2	real 888m26.524s / Balancing took 14.807050277777778 hours	See the table below

PARAMETER	VALUE
moverThreads (data node)	10
Bandwith (data node)	100000000
Ddfs.balancer.movedWinWidth (client)	5400000
Ddfs.balancer.moverThreads (client)	1000
Ddfs.balancer.dispatcherThreads (client)	200
Ddfs.datanode.balance.bandwidthPerSec (client)	100000000
Ddfs.balancer.max-size-to-move (client)	10737418240
Threshold (client)	8

APPENDIX 4: TYPE HANDLING ISSUE

MongoDB & Elasticsearch include a tool to ingest data (including JSON data) out of the box. Unfortunately, it is not always able to correctly determine the type of each field which, for example, can lead to dates being represented as strings.

There are two solutions to this problem:

1. Re-cast the data once ingested (too computationally intensive given the dataset size).
2. Add type annotations to the file (requires changes to the source files).

Since the goal of this exercise is to compare each technology fairly from the same source file (GeoJSON format), we decided to opt for a different solution and develop our own multi-threaded, multi-client ingestion tool which would handle all types and parse the GeoJSON file as expected. This removed the requirement to re-cast any data (i.e. only one pass is required) and also meant that no special annotations needed to be made to the source files (that would otherwise have made MongoDB & Elasticsearch a 'special case').

REPLY DATA

APPENDIX 5: QUERIES

TABLE 1: LIST OF QUERIES IN SCOPE

DATASET ID	QUERY ID	QUERY
1	1	Select * from dataset1 where dataset1.geo is within bbox1
1	2	Select * from dataset1 where dataset1.geo is within bbox2
1	3	Select * from dataset1 where dataset1.timestamp < time1 and dataset1.timestamp > time2
1	4	Select * from dataset1 where dataset1.geo is within bbox1 and dataset1. timestamp < time1 and dataset1.timestamp > time2
1	5	Select * from dataset1 where dataset1.geo is within bbox2 and dataset1.timestamp < time1 and dataset1.timestamp > time2
1	6	Select * from dataset1 where Short_text_field contains "dog"
1	7	Select * from dataset1 where Long_text_field_1 contains "dog"
1	8	Select * from dataset1 where Long_text_field_2 contains "迎"
1	9	Select * from dataset1 where dataset1.geo is within 10km of point1, order by distance to point1
1	10	Select * from dataset1 where dataset1.geo is within 10km of point1, order by timestamp
1	11	Select * from dataset1 where dataset1.geo is within 10km of any point in dataset2.geo
3	1	Select * from dataset3 where dataset3.geo intersects bbox1
3	2	Select * from dataset3 where dataset3.geo intersects bbox2
3	3	Select * from dataset3 where dataset3.geo intersects bbox1 and dataset3.timestamp < time1 and dataset3.timestamp > time2
3	4	Select * from dataset3 where dataset3.geo intersects bbox2 and dataset3.timestamp < time1 and dataset3.timestamp > time2
3	5	Select * from dataset3 where dataset3.geo is within 10km of point1, order by closest distance to point1
3	6	Select * from dataset3 where dataset3.geo is within 10km of point1, order by timestamp
3	7	Select * from dataset3 where dataset3.geo is within 10km of any point in dataset2.geo

REPLY DATA

APPENDIX 6: ADDITIONAL DATASETS

DATASET4: DATA GENERATION

10 bn random (with respect to shape, size, and location) polygons were generated with no holes and no intersecting edges. Based on the requirements, the polygons have between 3 and 8 vertices, and a random (uniformly distributed) area between 10800m^2 (a maximum size of a football field) and 4km^2 (size of a small town).

DATASET5: DATA GENERATION

10 bn random lines were generated with between 2 and 50 vertices with a random total length between 0.1 and 200km. The number of vertices was skewed towards lines with fewer vertices in order to reduce query complexity (as agreed with DSTL). Any two subsequent points on the random lines are within -180 and 180 degrees from each other, which ensures that the lines do not self-intersect. Line segment length was selected randomly given the random number of points in a given line and its random total length. Orientation angles of the lines were sampled uniformly.

TABLE 2: LIST OF QUERIES IN ADDITIONAL DATASETS

DATASET ID	QUERY ID	QUERY
4	1	Select * from dataset4 where dataset4.geo intersects bbox1
4	2	Select * from dataset4 where dataset4.geo intersects bbox2
4	3	Select * from dataset4 where dataset4.geo intersects bbox1 and dataset4.timestamp < time1 and dataset4.timestamp > time2
4	4	Select * from dataset4 where dataset4.geo intersects bbox2 and dataset4.timestamp < time1 and dataset4.timestamp > time2
4	5	Select * from dataset4 where dataset4.geo is within 10km of point1, order by closest distance to point1
4	6	Select * from dataset4 where dataset4.geo is within 10km of point1, order by timestamp
4	7	Select * from dataset4 where dataset4.geo is within 10km of any point in dataset2.geo
5	1	Select * from dataset5 where dataset5.geo intersects bbox1
5	2	Select * from dataset5 where dataset5.geo intersects bbox2
5	3	Select * from dataset5 where dataset5.geo intersects bbox1 and dataset5.timestamp < time1 and dataset5.timestamp > time2
5	4	Select * from dataset5 where dataset5.geo intersects bbox2 and dataset5.timestamp < time1 and dataset5.timestamp > time2
5	5	Select * from dataset5 where dataset5.geo is within 10km of point1, order by closest distance to point1
5	6	Select * from dataset5 where dataset5.geo is within 10km of point1, order by timestamp
5	7	Select * from dataset5 where dataset5.geo is within 10km of any point in dataset2.geo

APPENDIX 7: GEOSPARK VS MAGELLAN

We made the decision to focus our benchmarking on GeoSpark over Magellan due to the following reasons:

- GeoSpark uses R-tree indexing which improves performance when compared to Magellan which doesn't support them.

- As compared to Magellan, GeoSpark was under active development and had better community support.

- GeoSpark had better documentation with non-trivial examples as compared to Magellan.

- At the beginning of the project Magellan did not support Spark 2.0 and used Spark 1.4 which was quite outdated and hence impacted performance.

REPLY
DATA

APPENDIX 8: GEOWAVE EXCLUSION

The original scope included benchmarking GeoWave.

Following some investigation it was determined that at the time of writing this technology:

- does not support the GeoJSON format (a Dstl requirement) out of the box, and

- does not have JSON configurable ingest tooling to enable a user to write an ingest tool to parse GeoJSON.

As a result GeoWave was de-scoped.

In addition (but separate from the scoping decision) for those considering this technology it should be noted that at the time of writing the documentation is relatively difficult to follow and the API is written in Java. Depending on the technical capabilities of the user, this may lead to difficulties in setting up and/or using GeoWave.

REPLY DATA

APPENDIX 9: GNU FREE DOC LICENSE

GNU FREE DOCUMENTATION LICENSE

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document.

These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3. You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- **A.** Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- **B.** List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- **C.** State on the Title page the name of the publisher of the Modified Version, as the publisher.
- **D.** Preserve all the copyright notices of the Document.
- **E.** Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- **F.** Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- **G.** Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- **H.** Include an unaltered copy of this License.
- **I.** Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- **J.** Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- **K.** For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- **L.** Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- **M.** Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- **N.** Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- **O.** Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: HOW TO USE THIS LICENSE FOR YOUR DOCUMENTS

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (C) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with … Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

DATA REPLY is the Reply Group company specialising in data management using Big Data & Advanced Analytics methodologies. Data Reply supports customers in the design and implementation of data platforms that aim to enhance and capitalize on corporate information assets.

www.datareply.co.uk