

Exhibit A

Preliminary Infringement Contentions

**R2 Solutions LLC v. American Airlines, Inc.
Civil Action No. 4:22-cv-00353-ALM**



U.S. Pat. No.: 8,190,610

Claims: 1-5, 17-21

Preliminary Infringement Contentions

R2 Solutions LLC (“R2” or “Plaintiff”) contends that American Airlines, Inc. (“AA” or “Defendant”) has infringed and continues to infringe U.S. Pat. No. 8,190,610 (the “’610 Patent”) by, among other things, making, having made, using, offering for sale, and/or selling within this judicial district and elsewhere in the United States and/or importing into this judicial district and into the United States – without license or authority – the “Accused Instrumentalities,” which include, without limitation, all instrumentalities associated with mapping and/or reducing functionalities, merging and/or joining data having different schemas, and any other similar functionality and all instrumentalities related to how data is received, input, stored, organized, partitioned, accessed, analyzed, delivered, processed, and transformed into outputs. This includes the data analytics systems utilizing, at least in part, Apache Hadoop, Hive, Spark, Teradata Vantage, and/or similar functionality. The Accused Instrumentalities practice the steps of the asserted claims or are capable of performing the claim elements of the asserted claims, or otherwise satisfy all elements, described or set forth in one or more claims of the ’610 Patent in violation of at least 35 U.S.C. § 271(a). AA’s infringements also include, and R2 is entitled to recover damages for, sales of infringing products and components sold and/or shipped abroad that are subsequently imported into the United States.

Preliminary Infringement Contentions

R2 contends that each and every limitation of the asserted claims of the '610 Patent is literally present or performed and infringed by each and every Accused Instrumentality. In the alternative, R2 contends that all limitations of the asserted claims are present under the doctrine of equivalents because (1) to the extent there are any differences between the Accused Instrumentalities and those claim limitations (which R2 denies), such differences are insubstantial. Equivalency may also be shown by the fact that (2) the Accused Instrumentalities perform substantially the same function, in substantially the same way, to achieve substantially the same result as the functions recited in the limitations of the asserted claims.

R2 reserves the right to supplement or amend its claim charts, preliminary infringement contentions, and expert reports as may be appropriate or permissible under the applicable rules and Scheduling Order, including the Court's modifications to P.R. 3-1 available in the Scheduling Order. Any reference or citation to the specification is for illustrative purposes only and is not intended to suggest that the scope of the claimed invention is limited to any particular preferred embodiment. R2 does not hereby waive any applicable work product protections.

For the avoidance of doubt, the claim elements charted herein are software limitations and/or associated with software limitations for the purposes of P.R. 3-1(g), and to the extent that such elements are performed by hardware, firmware, software, and/or any combination thereof, the particular details are in AA's possession, custody, and control.

References Cited

The following chart presents R2 Solutions' analysis of the Accused Instrumentalities based on publicly available information. The citations in the chart refer to the following publicly available documents, which are incorporated by reference as if fully set forth herein. This infringement analysis is preliminary, and R2's investigation into AA's infringement is ongoing. R2 reserves the right to provide additional theories under which AA products/systems infringe this patent:

- [1] Sachin P. Bappalige, "An introduction to Apache Hadoop for big data," opensource.com (Aug. 26, 2014), *available at* <https://opensource.com/life/14/8/intro-apache-hadoop-big-data> as of 03/15/2022 ("**An introduction to Apache Hadoop for big data**")
- [2] MapReduce Tutorial, *available at* <http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html> as of 03/15/2022 ("**MapReduce Tutorial**")
- [3] Interface Reducer<K2,V2,K3,V3>, *available at* <https://hadoop.apache.org/docs/r2.8.0/api/org/apache/hadoop/mapred/Reducer.html> as of 03/15/2022 ("**Interface Reducer**")
- [4] ASF Infrabot, "HadoopMapReduce," Dashboard (July 9, 2019), *available at* <https://cwiki.apache.org/confluence/display/HADOOP2/HadoopMapReduce> as of 03/15/2022 ("**HadoopMapReduce**")
- [5] AA Associate Developer, IT Applications, *available at* <https://www.salary.com/job/american-airlines/associate-developer-it-applications/j202202092136250030017> as of 03/15/2022 ("**AA Associate Developer**")
- [6] AA Ft. Worth Associate Developer, *available at* <https://jobs.aa.com/job/Ft-Worth-Associate-DeveloperDeveloper%2C-IT-Applications-TX-76101/801819300/> as of 03/15/2022 ("**AA Ft. Worth Associate Developer**") (this position description mirrored the position description of [5] above, but the description is now unavailable because the position has been filled)

References Cited (continued)

- [7] Enum Schema._Fields, available at https://docs.cloudera.com/HDPDocuments/HDP1/HDP-Win-1.1.0/ds_Hive/api/index.html?org/apache/hadoop/hive/metastore/api/Schema._Fields.html as of 03/23/2022 (“Class Schema”)
- [8] Confluence Administrator, “Hive Tutorial,” Dashboard (updated by Owen O’Malley, Mar. 5, 2019), available at <https://cwiki.apache.org/confluence/display/Hive/Tutorial> as of 03/15/2022 (“Hive Tutorial”)
- [9] Confluence Administrator, “LanguageManual Joins,” Dashboard (updated by Dudu Markovitz, June 5, 2017), available at <https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Joins> as of 03/15/2022 (“LanguageManual Joins”)
- [10] Tanel Poder, “Speed Up Your Queries with Hive LLAP Engine on Hadoop or in the Cloud,” Gluent, available at <https://www.slideshare.net/gluent/speed-up-your-queries-with-hive-llap-engine-on-hadoop-or-in-the-cloud> as of 03/15/2022 (“Speed Up Your Queries with Hive LLAP Engine”)
- [11] “Apache Hive,” Wikipedia, available at https://en.wikipedia.org/wiki/Apache_Hive as of 03/15/2022 (“Apache Hive”)
- [12] Petar Zecevic & Marko Bonaci, “Spark in Action” (Manning Publications Nov. 2016), available at https://learning.oreilly.com/library/view/spark-in-action/9781617292606/kindle_split_014.html as of 03/15/2022 (“Spark in Action”)
- [13] “Apache Spark FAQ,” available at <https://spark.apache.org/faq.html> as of 03/15/2022 (“Apache Spark FAQ”)
- [14] “Milind R,” AA Engineer LinkedIn Profile, available at <https://www.linkedin.com/in/milind-r-719395206/> as of 03/15/2022 (“AA Big Data Engineer”)
- [15] “Rajat Gupta,” AA Engineer LinkedIn Profile, available at <https://www.linkedin.com/in/rajat-gupta-100a83209/> as of 03/15/2022 (“AA Data Engineer”)
- [16] “AA Privacy Notice,” available at <https://www.aa.com/i18n/customer-service/support/privacy-policy.jsp> (“AA Privacy Policy”)
- [17] Apache Spark 3.0.1, “RDD Programming Guide,” available at <http://spark.apache.org/docs/latest/rdd-programming-guide.html> as of 03/15/2022 (“RDD Programming Guide”)
- [18] “Learning Spark,” O’Reilly Media, Inc. (February 2015), available at <https://learning.oreilly.com/library/view/learning-spark/9781449359034/ch04.html#chap-pair-RDDS> as of 03/15/2022 (“Learning Spark”)

References Cited (continued)

- [19] “Spark RDD Transformations with examples,” available at <https://sparkbyexamples.com/apache-spark-rdd/spark-rdd-transformations/> as of 03/15/2022 (“**Spark RDD Transformations**”)
- [20] Shrey Mehrotra & Akash Grade, “Apache Quick Start Guide” (Packt Publishing, Jan. 2019), available at <https://learning.oreilly.com/library/view/apache-spark-quick/9781789349108/0ee1a5e2-09d0-49f0-99f5-9dee8336258d.xhtml> as of 03/15/2022 (“**Apache Spark Quick Start Guide**”)
- [21] “Spark SQL Shuffle Partitions,” available at <https://sparkbyexamples.com/spark/spark-shuffle-partitions/> as of 10/10/2022 (“**Spark SQL Shuffle Partitions**”)
- [22] “Apache Spark Map vs. FlatMap Operation,” DataFlair, available at <https://data-flair.training/blogs/apache-spark-map-vs-flatmap/> as of 03/15/2022 (“**Apache Spark Map vs. FlatMap**”)
- [23] Confluence Administrator, “LanguageManual Transform,” Dashboard (updated by Sushanth Sowmyan on Dec. 9, 2015), available at <https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Transform> as of 03/15/2022 (“**LanguageManual Transform**”)
- [24] System Properties Comparison Hive vs. Spark SQL vs. Teradata, available at <https://db-engines.com/en/system/Hive%3BSpark+SQL%3BTeradata> as of 10/12/2022 (“**System Properties Comparison**”)
- [25] Hadoop vs Teradata, available at <https://www.educba.com/hadoop-vs-teradata/> as of 10/12/2022 (“**Hadoop vs Teradata**”)
- [26] Teradata Vantage User Guide, available at <https://docs.teradata.com/r/Teradata-Vantage-User-Guide/January-2019/Access-to-Commands-and-Functions/Teradata-ML-Engine-Analytic-Functions/Teradata-ML-Engine-Analytic-Function-Execution-SQL-Syntax> as of 10/12/2022 (“**Teradata Vantage User Guide**”)
- [27] AA Using Teradata, available at <https://www.teradata.com/Customers/American-Airlines> as of 10/12/2022 (“**AA Using Teradata**”)
- [28] Hadoop Migration Program from Teradata, available at <https://www.teradata.com/Hadoop-Migration-Program> as of 10/12/2022 (“**Hadoop Migration Program**”)

U.S. Patent No. 8,190,610

US008190610B2

(12) **United States Patent**
Dasdan et al.

(10) **Patent No.:** **US 8,190,610 B2**
(45) **Date of Patent:** **May 29, 2012**

(54) **MAPREDUCE FOR DISTRIBUTED DATABASE PROCESSING**

(73) **Inventors:** **Ali Dasdan**, San Jose, CA (US);
Hung-Chih Yang, Sunnyvale, CA (US);
Ruey-Lung Hsiao, Los Angeles, CA (US)

(73) **Assignee:** **Yahoo! Inc.**, Sunnyvale, CA (US)

(*) **Notice:** Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 1105 days.

(21) **Appl. No.:** **11/539,090**

(22) **Filed:** **Oct. 5, 2006**

(65) **Prior Publication Data**
US 2008/0086442 A1 Apr. 10, 2008

(51) **Int. Cl.** **G06F 17/30** (2006.01)

(52) **U.S. CL.** **707/737; 707/968**

(58) **Field of Classification Search** **707/1-3, 707/737, 968**
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS
6,158,844 A * 12/2000 Tibbetts 717/100
6,341,289 B1 * 1/2002 Burroughs et al. 707/737

6,678,691 B1 * 1/2004 Kikkens 707/102
7,065,618 B1 6/2006 Ghemawat et al. 717/108
7,620,936 B2 * 11/2009 Ernst et al. 707/1
2004/0225638 A1 * 11/2004 Geiselhart et al. 707/1
2004/0230567 A1 * 11/2004 Wooley 707/3
2006/0117036 A1 * 6/2006 Cnanes et al. 707/100
2007/0038659 A1 * 2/2007 Datar et al. 707/101
2007/0255685 A1 * 11/2007 Boulit et al. 707/2

OTHER PUBLICATIONS
Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", USENIX Association OSDI '04: 6th Symposium on Operating Systems Design and Implementation, Dec. 6-8, 2004, pp. 137-149.
* cited by examiner

Primary Examiner—Klanth Pham
(74) **Attorney, Agent, or Firm**—Weaver Austin Villeneuve & Sampson LLP

(57) **ABSTRACT**
An input data set is treated as a plurality of grouped sets of key/value pairs, which enhances the utility of the MapReduce programming methodology. By utilizing such a grouping, map processing can be carried out independently on two or more related but possibly heterogeneous datasets (e.g., related by being characterized by a common primary key). The intermediate results of the map processing (key/value pairs) for a particular key can be processed together in a single reduce function by applying a different iterator to intermediate values for each group. Different iterators can be arranged inside reduce functions in ways however desired.

46 Claims, 5 Drawing Sheets

402 302 E M E' 406 E' D' 410 R 414
404 304 D M D' 408 D' 412 R 416

Title:

MAPREDUCE FOR DISTRIBUTED DATABASE PROCESSING

Priority Date:

October 5, 2006

Filing Date:

October 5, 2006

Issue Date:

May 29, 2012

Expiration Date:

October 14, 2029

(57)

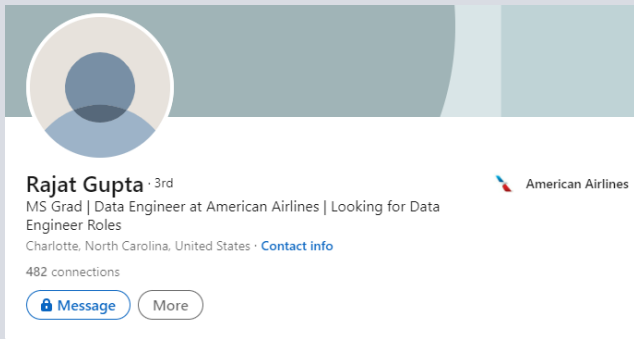
ABSTRACT

An input data set is treated as a plurality of grouped sets of key/value pairs, which enhances the utility of the MapReduce programming methodology. By utilizing such a grouping, map processing can be carried out independently on two or more related but possibly heterogeneous datasets (e.g., related by being characterized by a common primary key). The intermediate results of the map processing (key/value pairs) for a particular key can be processed together in a single reduce function by applying a different iterator to intermediate values for each group. Different iterators can be arranged inside reduce functions in ways however desired.

Claim 1

- 1[pre]** A method of processing data of a data set over a distributed system, wherein the data set comprises a plurality of data groups, the method comprising:
- 1[a]** partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,
- 1[b]** wherein the data of a first data group has a different schema than the data of a second data group and the data of the first data group is mapped differently than the data of the second data group so that different lists of values are output for the corresponding different intermediate data, wherein the different schema and corresponding different intermediate data have a key in common; and
- 1[c]** reducing the intermediate data for the data groups to at least one output data group, including processing the intermediate data for each data group in a manner that is defined to correspond to that data group, so as to result in a merging of the corresponding different intermediate data based on the key in common,
- 1[d]** wherein the mapping and reducing operations are performed by a distributed system.

Claim Chart – Claim 1

Claim Feature	Evidence
1 [pre] A method of processing data of a data set over a distributed system, wherein the data set comprises a plurality of data groups, the method comprising:	AA infringes Claim 1 of the '610 Patent. Each of the Accused Instrumentalities perform a method of processing data of a data set over a distributed system, wherein the data set comprises a plurality of data groups, the method comprising the steps discussed herein. The Accused Instrumentalities are based, at least in part, on Hadoop, Hive, Spark, Teradata Vantage, and/or other functionality, which enables processing of data of a data set over a distributed system. For example, AA utilizes, e.g., data engineers whose duties specifically include Hadoop, Hive, and/or Spark:  Rajat Gupta · 3rd MS Grad Data Engineer at American Airlines Looking for Data Engineer Roles Charlotte, North Carolina, United States · Contact info 482 connections Message More Experience Data Engineer American Airlines · Contract Feb 2020 - Present · 2 yrs 2 mos Worked on setting up AWS DMS and SNS for data transfer and replication and used SQL on the new AWS Databases like RedShift and Relation Data Services. Generates ETL scripts to transform, flatten, and enrich the data from source to target using AWS Glue and created event-driven ETL pipelines with AWS Glue. Executed <u>Hive</u> queries on Parquet tables stored in <u>Hive</u> to perform data analysis to meet the business requirements and worked on importing and exporting data from Oracle and DB2 into HDFS and <u>HIVE</u> using Sqoop. Used <u>Spark</u>-Streaming APIs to perform necessary transformations and actions on the fly for building the common learner data model which gets the data from Kafka in near real time and Persists into Cassandra. Contributed to creating ETL designs and provided end user an easy access to Data Mart Maintained Data Mart up-to-date and in accordance with company's requirements [14] AA Big Data Engineer Milind R · 3rd Actively looking for Hadoop developer/ Big Data developer position(C2C only) United States · Contact info 303 connections Message More Activity 305 followers [14] AA Big Data Engineer Milind hasn't posted lately Milind's recent posts and comments will be displayed here. Show all activity → Experience Big Data Engineer American Airlines · Contract Aug 2019 - Present · 2 yrs 8 mos United States • Design and Develop Data application using <u>Hadoop</u>, <u>Hive</u>, <u>Spark</u>, Scala, Sqoop, Atomic Scheduler, DB2, SQL Server, Teradata, Kafka. • As per business requirements, developed complex <u>Hive</u> and <u>Spark</u> scripts for transforming and analyzing data and maintaining data quality, data organization and metadata. • Involved in creating <u>HIVE</u> partition managed tables for each incremental load. Implemented Dynamic Partitions, Buckets in <u>HIVE</u> for effective data access. • Worked on a migration project to migrate data from different sources (Teradata, Hadoop, DB2) to Google Cloud Platform (GCP) using UDP framework and transforming the data using Spark scripts. • Used Atomic Scheduler to schedule and maintain work flows. • Involved in data profiling activities like data type validation, NULL checks for the ingested data in the Hadoop layer. • Used Spark context, <u>Spark</u>-SQL, Data Frames, Pair RDD's & <u>Spark</u> YARN to improve the performance and optimize existing algorithms. • Production Rollout Support which involves monitoring Workflows, Queues and Bug fixing of Data lake. • Developed Spark code using Scala and <u>Spark</u>-SQL for faster processing of data. • Regular Interaction with the Business Team, DB administrators, to ensure and maintain smooth running of the application.

Claim Chart – Claim 1

Claim Feature	Evidence
1 [pre] A method of processing data of a data set over a distributed system, wherein the data set comprises a plurality of data groups, the method comprising:	In another example, AA employs, e.g., developers qualified in big data solutions such as Hadoop and Spark: Associate Developer, IT Applications POSTED ON 2/11/2022 AVAILABLE BEFORE 8/9/2022  American Airlines  Tulsa, OK  Full Time Job Posting for Associate Developer, IT Applications at American Airlines Location: Tulsa Bus Office (TUL-THDQ) Additional Locations: None Requisition ID: 55022 Intro Are you ready to explore a world of possibilities, both at work and travel the world, grow your expertise and become the best version with flexibility and grace, learning new skills and advancing your career? Your personal and work life and hop on board! Why you'll love this job Responsible for leveraging cutting edge technology to solve business problems, the development process from inception through transition, advocating for the development of tools to analyze, model, design, construct and [5] AA Associate Developer All you'll need for success ... continued Preferred Qualifications- Education & Prior Job Experience Airline industry experience, including business processes and supporting technology desired Experience architecting and implementing data analytics solutions in Azure Experience with Big Data BI solutions including <u>Hadoop</u> (HDInsight/<u>Spark</u>), Azure SQL DW, etc. Experience in programming utilizing C#, Python, JavaScript or similar languages is a plus Experience in predictive analytics (especially Azure ML) Mainframe SCEPTRE Skills, Licenses & Certifications Experience with Agile project management methodology Demonstrated initiative, flexibility, and ability to adapt to changing priorities and work environments Possess a keen desire to learn and expand depth of knowledge Ability to analyze complex problems and implement solutions and/or workarounds Proficiency in Microsoft Suite (Word, Excel, PowerPoint, and Visio)

Claim Chart – Claim 1

Claim Feature	Evidence
1 [pre] A method of processing data of a data set over a distributed system, wherein the data set comprises a plurality of data groups, the method comprising:	In another example, AA utilizes Teradata Vantage, which employs similar functionality to Hadoop, Hive, and/or Spark:  American Airlines is the world’s largest airline when measured by fleet size, scheduled passengers carried, and number of destinations served. Its mission is to “care for people on life’s journey.” The airline averages 6,800 flights per day to more than 365 destinations in 61 countries and in 2021, safely transported 164M passengers. This impressive airline fuels business-critical operations across multiple business departments performing analytics for revenue management, technical operations, network planning, marketing, cargo, passenger sales, and more, all using Teradata Vantage as its connected cloud data platform. Data is the fuel for American Airlines and analytics are the propulsion that lifts American Airlines above the everyday turbulence encountered by airline operators. [27] AA Using Teradata

Claim Chart – Claim 1

Claim Feature	Evidence																																																																
1 [pre] A method of processing data of a data set over a distributed system, wherein the data set comprises a plurality of data groups, the method comprising:	Teradata Vantage employs similar functionality to Hadoop, Hive, and/or Spark: System Properties Comparison Hive vs. Spark SQL vs. Teradata Please select another system to include it in the comparison. <table><tr><th colspan="4">Editorial information provided by DB-Engines</th></tr><tr><th>Name</th><th>Hive X</th><th>Spark SQL X</th><th>Teradata X</th></tr><tr><td>Description</td><td>data warehouse software for querying and managing large distributed datasets, built on Hadoop</td><td>Spark SQL is a component on top of 'Spark Core' for structured data processing</td><td>A hybrid cloud data analytics software platform (Teradata Vantage)</td></tr><tr><td>Primary database model</td><td>Relational DBMS</td><td>Relational DBMS</td><td>Relational DBMS</td></tr><tr><td>Secondary database models</td><td></td><td></td><td>Document store Graph DBMS Spatial DBMS Time Series DBMS</td></tr><tr><td>DB-Engines Ranking </td><td>Score 80.60 Rank #17 Overall #11 Relational DBMS</td><td>Score 22.58 Rank #36 Overall #22 Relational DBMS</td><td>Score 66.07 Rank #18 Overall #12 Relational DBMS</td></tr><tr><td>Website</td><td>hive.apache.org</td><td>spark.apache.org/sql</td><td>www.teradata.com</td></tr><tr><td>Technical documentation</td><td>cwiki.apache.org/confluence/display/Hive/Home</td><td>spark.apache.org/docs/latest/sql-programming-guide.html</td><td>docs.teradata.com</td></tr><tr><td>Developer</td><td>Apache Software Foundation </td><td>Apache Software Foundation</td><td>Teradata</td></tr><tr><td>Initial release</td><td>2012</td><td>2014</td><td>1984</td></tr><tr><td>Current release</td><td>3.1.3, April 2022</td><td>3.3.0 (2.13), June 2022</td><td>Teradata Vantage 1.0 MU2, January 2019</td></tr><tr><td>License </td><td>Open Source </td><td>Open Source </td><td>commercial</td></tr><tr><td>Cloud-based only </td><td>no</td><td>no</td><td>no</td></tr><tr><td>DBaaS offerings (sponsored links) </td><td></td><td></td><td></td></tr><tr><td>Implementation language</td><td>Java</td><td>Scala</td><td></td></tr><tr><td>Server operating systems</td><td>All OS with a Java VM</td><td>Linux OS X Windows</td><td>hosted Linux</td></tr></table> [24] System Properties Comparison	Editorial information provided by DB-Engines				Name	Hive X	Spark SQL X	Teradata X	Description	data warehouse software for querying and managing large distributed datasets, built on Hadoop	Spark SQL is a component on top of 'Spark Core' for structured data processing	A hybrid cloud data analytics software platform (Teradata Vantage)	Primary database model	Relational DBMS	Relational DBMS	Relational DBMS	Secondary database models			Document store Graph DBMS Spatial DBMS Time Series DBMS	DB-Engines Ranking 	Score 80.60 Rank #17 Overall #11 Relational DBMS	Score 22.58 Rank #36 Overall #22 Relational DBMS	Score 66.07 Rank #18 Overall #12 Relational DBMS	Website	hive.apache.org	spark.apache.org/sql	www.teradata.com	Technical documentation	cwiki.apache.org/confluence/display/Hive/Home	spark.apache.org/docs/latest/sql-programming-guide.html	docs.teradata.com	Developer	Apache Software Foundation 	Apache Software Foundation	Teradata	Initial release	2012	2014	1984	Current release	3.1.3, April 2022	3.3.0 (2.13), June 2022	Teradata Vantage 1.0 MU2, January 2019	License 	Open Source 	Open Source 	commercial	Cloud-based only 	no	no	no	DBaaS offerings (sponsored links) 				Implementation language	Java	Scala		Server operating systems	All OS with a Java VM	Linux OS X Windows	hosted Linux
Editorial information provided by DB-Engines																																																																	
Name	Hive X	Spark SQL X	Teradata X																																																														
Description	data warehouse software for querying and managing large distributed datasets, built on Hadoop	Spark SQL is a component on top of 'Spark Core' for structured data processing	A hybrid cloud data analytics software platform (Teradata Vantage)																																																														
Primary database model	Relational DBMS	Relational DBMS	Relational DBMS																																																														
Secondary database models			Document store Graph DBMS Spatial DBMS Time Series DBMS																																																														
DB-Engines Ranking 	Score 80.60 Rank #17 Overall #11 Relational DBMS	Score 22.58 Rank #36 Overall #22 Relational DBMS	Score 66.07 Rank #18 Overall #12 Relational DBMS																																																														
Website	hive.apache.org	spark.apache.org/sql	www.teradata.com																																																														
Technical documentation	cwiki.apache.org/confluence/display/Hive/Home	spark.apache.org/docs/latest/sql-programming-guide.html	docs.teradata.com																																																														
Developer	Apache Software Foundation 	Apache Software Foundation	Teradata																																																														
Initial release	2012	2014	1984																																																														
Current release	3.1.3, April 2022	3.3.0 (2.13), June 2022	Teradata Vantage 1.0 MU2, January 2019																																																														
License 	Open Source 	Open Source 	commercial																																																														
Cloud-based only 	no	no	no																																																														
DBaaS offerings (sponsored links) 																																																																	
Implementation language	Java	Scala																																																															
Server operating systems	All OS with a Java VM	Linux OS X Windows	hosted Linux																																																														

Claim Chart – Claim 1

Claim Feature	Evidence												
1 [pre] A method of processing data of a data set over a distributed system, wherein the data set comprises a plurality of data groups, the method comprising:	Teradata Vantage employs similar functionality to Hadoop, Hive, and/or Spark: Comparison Table Between Hadoop and Teradata Below are the lists of points, describe the Differences between Hadoop and Teradata : <table><tr><th>Basis of Comparison</th><th>Teradata</th><th>Hadoop</th></tr><tr><td>Parallel Processing</td><td>Workload is divided across the system and evenly among the processors in the system.</td><td>Workload is divided among the different nodes on which relevant data is present and each node processes the task individually in parallel which reduces the overall time taken to complete the task.</td></tr><tr><td>Share-nothing Architecture</td><td>Teradata task executing in a virtual processor is independent of the tasks in other virtual processors.</td><td>Task execution on any node of the Hadoop is independent to tasks executing on other nodes.</td></tr><tr><td>Highly Scalable</td><td>More nodes/disks can be added but will increase the licensing cost.</td><td>More number of nodes/disks can be added as and when required to increase the processing and storage power.</td></tr></table> [25] Hadoop vs Teradata	Basis of Comparison	Teradata	Hadoop	Parallel Processing	Workload is divided across the system and evenly among the processors in the system.	Workload is divided among the different nodes on which relevant data is present and each node processes the task individually in parallel which reduces the overall time taken to complete the task.	Share-nothing Architecture	Teradata task executing in a virtual processor is independent of the tasks in other virtual processors.	Task execution on any node of the Hadoop is independent to tasks executing on other nodes.	Highly Scalable	More nodes/disks can be added but will increase the licensing cost.	More number of nodes/disks can be added as and when required to increase the processing and storage power.
Basis of Comparison	Teradata	Hadoop											
Parallel Processing	Workload is divided across the system and evenly among the processors in the system.	Workload is divided among the different nodes on which relevant data is present and each node processes the task individually in parallel which reduces the overall time taken to complete the task.											
Share-nothing Architecture	Teradata task executing in a virtual processor is independent of the tasks in other virtual processors.	Task execution on any node of the Hadoop is independent to tasks executing on other nodes.											
Highly Scalable	More nodes/disks can be added but will increase the licensing cost.	More number of nodes/disks can be added as and when required to increase the processing and storage power.											

Claim Chart – Claim 1

Claim Feature	Evidence
1 [pre] A method of processing data of a data set over a distributed system, wherein the data set comprises a plurality of data groups, the method comprising:	Teradata Vantage employs similar functionality to Hadoop, Hive, and/or Spark: The Best Alternative to Hadoop The Teradata Hadoop Migration Program is a lifeline for companies who need an alternative to Hadoop due to the high cost, production difficulty, infrastructure complexity, lack of skills, and limited insight that comes with the best-intentioned Hadoop deployments. Teradata Vantage is the Hadoop replacement proven to drive maximum business value through analytical insight. This program quickly, easily migrates existing Hadoop data and workloads to Vantage and cloud low-cost object storage using robust and proven migration methodologies and tools. [28] Hadoop Migration Program

Claim Chart – Claim 1

Claim Feature	Evidence
1 [pre] A method of processing data of a data set over a distributed system, wherein the data set comprises a plurality of data groups, the method comprising:	The Accused Instrumentalities are based on Hadoop, Hive, Spark, Teradata Vantage, and/or other functionality, which enables processing of data of a data set over a distributed system. [10] Speed Up Your Queries with Hive LLAP Engine at 12: Introducing Hive LLAP • “Live Long and Process” or “Low Latency Analytical Processing” • Not an execution engine (like Tez), LLAP simply enhances the Hive execution model • Built for fast query response time against smaller data volumes • Allows concurrent execution of analytical workloads • Intelligent memory caching for quick startup and data sharing • Caches most active data in RAM • Shared cache across clients • Persistent server used to instantly execute queries • LLAP daemons are “always on” • Data passed to execution as it becomes ready 

Claim Chart – Claim 1

Claim Feature	Evidence
1 [pre] A method of processing data of a data set over a distributed system , wherein the data set comprises a plurality of data groups, the method comprising:	Each of the Accused Instrumentalities are implemented over a distributed system: [1] An introduction to Apache Hadoop for big data at 2: The Apache Hadoop framework is composed of the following modules 1. Hadoop Common: contains libraries and utilities needed by other Hadoop modules 2. Hadoop Distributed File System (HDFS): a distributed file-system that stores data <u>on the commodity machines, providing very high aggregate bandwidth across the cluster</u> 3. Hadoop YARN: a resource-management platform responsible for managing compute resources in clusters and using them for scheduling of users' applications 4. Hadoop MapReduce: a programming model for large scale data processing <u>All the modules in Hadoop are designed with a fundamental assumption that hardware failures (of individual machines, or racks of machines) are common and thus should be automatically handled in software by the framework.</u> Apache Hadoop's MapReduce

Claim Chart – Claim 1

Claim Feature	Evidence
1 [pre] A method of processing data of a data set over a distributed system , wherein the data set comprises a plurality of data groups, the method comprising:	Each of the Accused Instrumentalities are implemented over a distributed system: [11] Apache Hive: <u>Apache Hive is a data warehouse software project built on top of Apache Hadoop for providing data query and analysis.</u>^[3] Hive gives an <u>SQL-like interface</u> to query data stored in various databases and file systems that integrate with Hadoop. Traditional SQL queries must be implemented in the <u>MapReduce Java API</u> to execute SQL applications and queries over distributed data. Hive provides the necessary SQL abstraction to integrate SQL-like queries (HiveQL) into the underlying Java without the need to implement queries in the low-level Java API. Since most data warehousing applications work with SQL-based querying languages, Hive aids portability of SQL-based applications to Hadoop.^[4] While initially developed by <u>Facebook</u>, Apache Hive is used and developed by other companies such as <u>Netflix</u> and the <u>Financial Industry Regulatory Authority (FINRA)</u>.^{[5][6]} Amazon maintains a software fork of Apache Hive included in <u>Amazon Elastic MapReduce</u> on <u>Amazon Web Services</u>.^[7]

Claim Chart – Claim 1

Claim Feature	Evidence												
1 [pre] A method of processing data of a data set over a distributed system, wherein the data set comprises a plurality of data groups, the method comprising:	The Accused Instrumentalities include a method of processing data of a data set over a distributed system, wherein the data set comprises a plurality of data groups: Comparison Table Between Hadoop and Teradata Below are the lists of points, describe the Differences between Hadoop and Teradata : <table><tr><th>Basis of Comparison</th><th>Teradata</th><th>Hadoop</th></tr><tr><td>Parallel Processing</td><td>Workload is divided across the system and evenly among the processors in the system.</td><td>Workload is divided among the different nodes on which relevant data is present and each node processes the task individually in parallel which reduces the overall time taken to complete the task.</td></tr><tr><td>Share-nothing Architecture</td><td>Teradata task executing in a virtual processor is independent of the tasks in other virtual processors.</td><td>Task execution on any node of the Hadoop is independent to tasks executing on other nodes.</td></tr><tr><td>Highly Scalable</td><td>More nodes/disks can be added but will increase the licensing cost.</td><td>More number of nodes/disks can be added as and when required to increase the processing and storage power.</td></tr></table> [25] Hadoop vs Teradata	Basis of Comparison	Teradata	Hadoop	Parallel Processing	Workload is divided across the system and evenly among the processors in the system.	Workload is divided among the different nodes on which relevant data is present and each node processes the task individually in parallel which reduces the overall time taken to complete the task.	Share-nothing Architecture	Teradata task executing in a virtual processor is independent of the tasks in other virtual processors.	Task execution on any node of the Hadoop is independent to tasks executing on other nodes.	Highly Scalable	More nodes/disks can be added but will increase the licensing cost.	More number of nodes/disks can be added as and when required to increase the processing and storage power.
Basis of Comparison	Teradata	Hadoop											
Parallel Processing	Workload is divided across the system and evenly among the processors in the system.	Workload is divided among the different nodes on which relevant data is present and each node processes the task individually in parallel which reduces the overall time taken to complete the task.											
Share-nothing Architecture	Teradata task executing in a virtual processor is independent of the tasks in other virtual processors.	Task execution on any node of the Hadoop is independent to tasks executing on other nodes.											
Highly Scalable	More nodes/disks can be added but will increase the licensing cost.	More number of nodes/disks can be added as and when required to increase the processing and storage power.											

Claim Chart – Claim 1

Claim Feature	Evidence
1 [pre] A method of processing data of a data set over a distributed system, wherein the data set comprises a plurality of data groups , the method comprising:	The Accused Instrumentalities include a data set comprising a plurality of data groups. For example, AA collects a myriad of data from a variety of sources (e.g., websites, customers, vendors, etc.): Information we collect and how we collect it We collect and maintain personal information about you from many sources to understand and meet your needs, facilitate your travel, manage our business, and for other purposes disclosed to you. For example, we collect personal information about you from: ■ You, when you voluntarily provide us with information ■ Your transactions with American ■ Other third-party sources, such as travel agents, other airlines (e.g., code share and alliance partners) and travel and hospitality service providers (e.g., transportation or tour operators). If the information is to be collected directly from you, you may in some cases have the option to decline providing that information. However, your choice to not provide information may impact your use of certain features or services. <u>The personal information we collect about you through these various sources may include, but is not limited to:</u> ■ Name, addresses and telephone numbers ■ Date of birth and gender ■ Email addresses, fax numbers and pager numbers ■ Social media handles, including when you communicate with us through social media ■ AAdvantage® account number ■ Credit/debit card payment information, including card number(s), associated billing address(es), and expiration date(s) ■ Information required to facilitate travel or other services, including travel companion(s) names, emergency contacts, and photographs ■ Information about your travel, including booking and itinerary information, requests for assistance, interactions with staff and cabin crew, and dietary restrictions ■ Digital identity credentials (such as credentials linked to facial or fingerprint verification, vaccination status, or negative test results) that you provide for specific purposes, such as compliance with customs and immigration requirements during international travel or services in which you choose to participate ■ Details of your previous travel, such as previous flights and travel-related issues ■ Driver's license number ■ Passport number, nationality and country of residence ■ Technical information such as your browser type, IP address, type of operating system you use, your geolocation, the name of your internet service provider, mobile advertising identifiers, and pages visited on our Services ■ Information provided via survey, focus group or other marketing research efforts ■ When you send us an email or online customer service request, we will retain the content of the message, your email address or any contact information you provide, and our response in order to handle any follow-up questions you may have. We also use this information to measure how effectively we address your concerns. ■ Corporate-contract, employer and/or other corporate affiliation (e.g., employer name, title, work address and contact information), including Business Extra® account information (e.g., tax ID, business type, number of employees, number of business travelers, and travel manager information) [16] AA Privacy Policy

Claim Chart – Claim 1

Claim Feature	Evidence
1 [pre] A method of processing data of a data set over a distributed system, wherein the data set comprises a plurality of data groups, the method comprising:	In another example, Hadoop runs in clusters, indicating that the data set comprises a plurality of data groups. Additionally, Spark revolves around RDDs which are a plurality of elements (e.g., data groups) that can be operated in parallel. [13] Apache Spark FAQ at 1: How does Spark relate to Apache Hadoop? Spark is a fast and general processing engine compatible with Hadoop data. It can run in Hadoop clusters through YARN or Spark's standalone mode, and it can process data in HDFS, HBase, Cassandra, Hive, and any Hadoop InputFormat. It is designed to perform both batch processing (similar to MapReduce) and new workloads like streaming, interactive queries, and machine learning. [17] RDD Programming Guide at 4, 2: Resilient Distributed Datasets (RDDs) Spark revolves around the concept of a <i>resilient distributed dataset</i> (RDD), which is a fault-tolerant collection of elements that can be operated on in parallel. There are two ways to create RDDs: <u>parallelizing an existing collection in your driver program, or referencing a dataset in an external storage system, such as a shared filesystem, HDFS, HBase, or any data source offering a Hadoop InputFormat.</u> Overview At a high level, every Spark application consists of a <i>driver program</i> that runs the user's main function and executes various <i>parallel operations</i> on a cluster. The main abstraction Spark provides is a <i>resilient distributed dataset</i> (RDD), which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel. RDDs are created by starting with a file in the Hadoop file system (or any other Hadoop-supported file system), or an existing Scala collection in the driver program, and transforming it. Users may also ask Spark to <i>persist</i> an RDD in memory, allowing it to be reused efficiently across parallel operations. Finally, RDDs automatically recover from node failures.

Claim Chart – Claim 1

Claim Feature	Evidence
1 [a] partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,	The method performed by the Accused Instrumentalities include partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group. For example, in the Accused Instrumentalities' Hadoop MapReduce framework, the input data is partitioned into independent chunks (data partitions), which are processed by Mapper functions: [2] MapReduce Tutorial at 2: <u>A MapReduce job usually splits the input data-set into independent chunks which are processed by the <i>map tasks</i> in a completely parallel manner.</u> The framework sorts the outputs of the maps, which are then input to the <i>reduce tasks</i>. Typically both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.

Claim Chart – Claim 1

Claim Feature	Evidence
1 [a] <u>partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable</u> to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,	The data partitions with key-value pairs are passed to user-configurable Mapper functions for further processing: [4] HadoopMapReduce at 1: <u>As key-value pairs are read from the RecordReader they are passed to the configured Mapper. The user supplied Mapper does whatever it wants with the input pair and calls <code>OutputCollector.collect</code> with key-value pairs of its own choosing. The output it generates must use one key class and one value class. This is because the Map output will be written into a <code>SequenceFile</code> which has per-file type information and all the records must have the same type (use subclassing if you want to output different data structures). The Map input and output key-value pairs are not necessarily related typewise or in cardinality.</u>

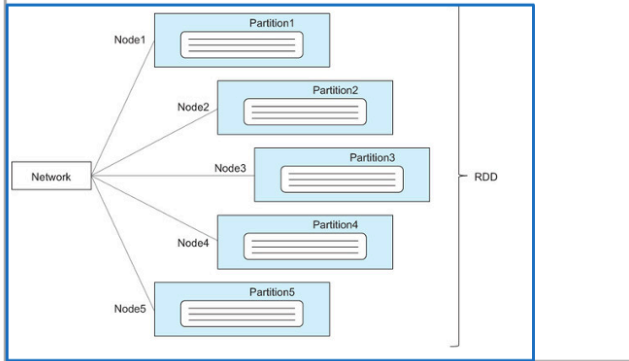
Claim Chart – Claim 1

Claim Feature	Evidence
1 [a] partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,	The Mapper function maps the key-value pairs to form intermediate key-value pairs: [2] MapReduce Tutorial at 7: Mapper Mapper maps <u>input key/value pairs</u> to a set of <u>intermediate key/value pairs</u>. Maps are the individual tasks that transform <u>input records</u> into <u>intermediate records</u>. The transformed intermediate records do not need to be of the same type as the input records. A given input pair may map to zero or many output pairs.

Claim Chart – Claim 1

Claim Feature	Evidence
1 [a] partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,	In another example, with Spark, a data set is first partitioned into a plurality of elements and distributed across nodes. This distribution of elements is called a resilient distributed dataset (RDD): [17] RDD Programming Guide at 2, 4, 5: Overview At a high level, every Spark application consists of a <i>driver program</i> that runs the user's main function and executes various <i>parallel operations</i> on a cluster. <u>The main abstraction Spark provides is a <i>resilient distributed dataset</i> (RDD), which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel.</u> RDDs are created by starting with a file in the Hadoop file system (or any other Hadoop-supported file system), or an existing Scala collection in the driver program, and transforming it. Users may also ask Spark to <i>persist</i> an RDD in memory, allowing it to be reused efficiently across parallel operations. Finally, RDDs automatically recover from node failures. Resilient Distributed Datasets (RDDs) Spark revolves around the concept of a <i>resilient distributed dataset</i> (RDD), which is a fault-tolerant collection of elements that can be operated on in parallel. <u>There are two ways to create RDDs: <i>parallelizing</i> an existing collection in your driver program, or <i>referencing a dataset in an external storage system</i>, such as a shared filesystem, HDFS, HBase, or any data source offering a Hadoop InputFormat.</u> One important parameter for parallel collections is the number of <i>partitions</i> to cut the dataset into. Spark will run one task for each partition of the cluster. <u>Typically you want 2-4 partitions for each CPU in your cluster. Normally, Spark tries to set the number of partitions automatically based on your cluster.</u> However, you can also set it manually by passing it as a second parameter to <code>parallelize</code> (e.g. <code>sc.parallelize(data, 10)</code>). Note: some places in the code use the term <i>slices</i> (a synonym for partitions) to maintain backward compatibility.

Claim Chart – Claim 1

Claim Feature	Evidence
1 [a] partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,	[12] Spark in Action: Each part (piece or slice) of an RDD is called a <i>partition</i>.^[2] When you load a text file from your local filesystem into Spark, for example, the file's contents are split into partitions, which are evenly distributed to nodes in a cluster. More than one partition may end up on the same node. The sum of all those partitions forms your RDD. This is where the word <i>distributed</i> in <i>resilient distributed dataset</i> comes from. Figure 4.1 shows the distribution of lines of a text file loaded into an RDD in a five-node cluster. The original file had 15 lines of text, so each RDD partition was formed with 3 lines of text. Each RDD maintains a list of its partitions and an optional list of preferred locations for computing the partitions. Figure 4.1. Simplified look at partitions of an RDD in a five-node cluster. The RDD was created by loading a text file using the <code>textFile</code> method of <code>SparkContext</code>. The loaded text file had 15 lines of text, so each partition was formed with 3 lines of text. 

Claim Chart – Claim 1

Claim Feature	Evidence
1 [a] partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,	In the Accused Instrumentalities, the data partitions have a plurality of key-value pairs. For example, one form of the RDDs is a pair RDD, which includes key/value pairs: [18] Learning Spark: Spark provides special operations on <u>RDDs containing key/value pairs</u>. These RDDs are called <u>pair RDDs</u>. <u>Pair RDDs are a useful building block in many programs, as they expose operations that allow you to act on each key in parallel or regroup data across the network</u>. For example, pair RDDs have a <code>reduceByKey()</code> method that can aggregate data separately for each key, and a <code>join()</code> method that can merge two RDDs together by grouping elements with the same key. It is common to extract fields from an RDD (representing, for instance, an event time, customer ID, or other identifier) and use those fields as keys in pair RDD operations. [12] Spark in Action: 4.1. WORKING WITH PAIR RDDS Storing data as key-value pairs offers a simple, general, and extensible data model because each <u>key-value pair can be stored independently</u> and it's easy to add new types of keys and new types of values. This extensibility and simplicity have made this practice fundamental to several frameworks and applications. For example, many popular caching systems and NoSQL databases, such as memcached and Redis, are key-value stores. Hadoop's MapReduce also operates on key-value pairs (as you can see in appendix A). Keys and values can be simple types such as integers and strings, as well as more-complex data structures. Data structures traditionally used to represent key-value pairs are <i>associative arrays</i>, also called <i>dictionaries</i> in Python and <i>maps</i> in Scala and Java. <u>In Spark, RDDs containing key-value tuples are called pair RDDs</u>. Although you don't have to use data in Spark in the form of key-value pairs (as you've seen in the previous chapters), pair RDDs are well suited (and indispensable) for many use cases. <u>Having keys along with the data enables you to aggregate, sort, and join the data</u>, as you'll soon see. But before doing any of that, the first step is to create a pair RDD, of course.

Claim Chart – Claim 1

Claim Feature	Evidence
1 [a] partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,	The Accused Instrumentalities provide each data partition to one of a plurality of mapping functions. For example, the RDDs support two types of data operations: transformations and actions. [17] RDD Programming Guide at 6: RDD Operations RDDs support two types of operations: <u>transformations</u>, which create a new dataset from an existing one, and <u>actions</u>, which return a value to the driver program after running a computation on the dataset. <u>For example, map is a transformation that passes each dataset element through a function and returns a new RDD representing the results.</u> On the other hand, reduce is an action that aggregates all the elements of the RDD using some function and returns the final result to the driver program (although there is also a parallel reduceByKey that returns a distributed dataset). All transformations in Spark are <i>lazy</i>, in that they do not compute their results right away. Instead, they just remember the transformations applied to some base dataset (e.g. a file). The transformations are only computed when an action requires a result to be returned to the driver program. This design enables Spark to run more efficiently. For example, we can realize that a dataset created through map will be used in a reduce and return only the result of the reduce to the driver, rather than the larger mapped dataset.

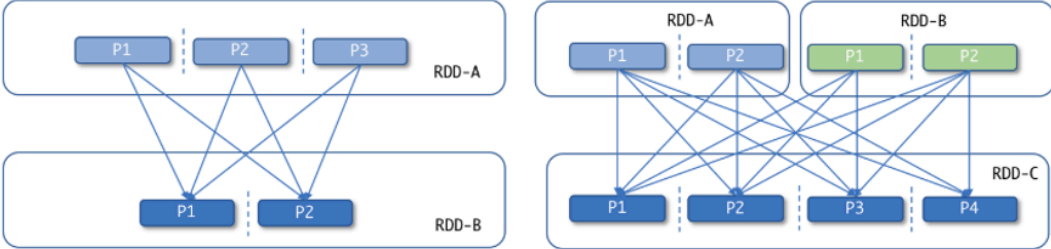
Claim Chart – Claim 1

Claim Feature	Evidence																		
1 [a] partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,	Examples of transformation functions: [17] RDD Programming Guide at 11–12: Transformations ↗ The following table lists some of the common transformations supported by Spark. Refer to the RDD API doc (Scala, Java, Python, R) and pair RDD functions doc (Scala, Java) for details. <table> <tr> <th>Transformation</th><th>Meaning</th></tr> <tr> <td><code>map(func)</code></td><td>Return a new distributed dataset formed by passing each element of the source through a function <i>func</i>.</td></tr> <tr> <td><code>filter(func)</code></td><td>Return a new dataset formed by selecting those elements of the source on which <i>func</i> returns true.</td></tr> <tr> <td><code>flatMap(func)</code></td><td>Similar to map, but each input item can be mapped to 0 or more output items (so <i>func</i> should return a Seq rather than a single item).</td></tr> <tr> <td><code>mapPartitions(func)</code></td><td>Similar to map, but runs separately on each partition (block) of the RDD, so <i>func</i> must be of type <code>Iterator<T> => Iterator<U></code> when running on an RDD of type T.</td></tr> <tr> <td><code>mapPartitionsWithIndex(func)</code></td><td>Similar to mapPartitions, but also provides <i>func</i> with an integer value representing the index of the partition, so <i>func</i> must be of type <code>(Int, Iterator<T>) => Iterator<U></code> when running on an RDD of type T.</td></tr> <tr> <td><code>groupByKey([numPartitions])</code></td><td>When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs. Note: If you are grouping in order to perform an aggregation (such as a sum or average) over each key, using <code>reduceByKey</code> or <code>aggregateByKey</code> will yield much better performance. Note: By default, the level of parallelism in the output depends on the number of partitions of the parent RDD. You can pass an optional <code>numPartitions</code> argument to set a different number of tasks.</td></tr> <tr> <td><code>reduceByKey(func, [numPartitions])</code></td><td>When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function <i>func</i>, which must be of type <code>(V,V) => V</code>. Like in <code>groupByKey</code>, the number of reduce tasks is configurable through an optional second argument.</td></tr> <tr> <td><code>aggregateByKey(zeroValue)(seqOp, combOp, [numPartitions])</code></td><td>When called on a dataset of (K, V) pairs, returns a dataset of (K, U) pairs where the values for each key are aggregated using the given combine functions and a neutral "zero" value. Allows an aggregated value type that is different than the input value type, while avoiding unnecessary allocations. Like in <code>groupByKey</code>, the number of reduce tasks is configurable through an optional second argument.</td></tr> </table>	Transformation	Meaning	<code>map(func)</code>	Return a new distributed dataset formed by passing each element of the source through a function <i>func</i> .	<code>filter(func)</code>	Return a new dataset formed by selecting those elements of the source on which <i>func</i> returns true.	<code>flatMap(func)</code>	Similar to map, but each input item can be mapped to 0 or more output items (so <i>func</i> should return a Seq rather than a single item).	<code>mapPartitions(func)</code>	Similar to map, but runs separately on each partition (block) of the RDD, so <i>func</i> must be of type <code>Iterator<T> => Iterator<U></code> when running on an RDD of type T.	<code>mapPartitionsWithIndex(func)</code>	Similar to mapPartitions, but also provides <i>func</i> with an integer value representing the index of the partition, so <i>func</i> must be of type <code>(Int, Iterator<T>) => Iterator<U></code> when running on an RDD of type T.	<code>groupByKey([numPartitions])</code>	When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs. Note: If you are grouping in order to perform an aggregation (such as a sum or average) over each key, using <code>reduceByKey</code> or <code>aggregateByKey</code> will yield much better performance. Note: By default, the level of parallelism in the output depends on the number of partitions of the parent RDD. You can pass an optional <code>numPartitions</code> argument to set a different number of tasks.	<code>reduceByKey(func, [numPartitions])</code>	When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function <i>func</i> , which must be of type <code>(V,V) => V</code> . Like in <code>groupByKey</code> , the number of reduce tasks is configurable through an optional second argument.	<code>aggregateByKey(zeroValue)(seqOp, combOp, [numPartitions])</code>	When called on a dataset of (K, V) pairs, returns a dataset of (K, U) pairs where the values for each key are aggregated using the given combine functions and a neutral "zero" value. Allows an aggregated value type that is different than the input value type, while avoiding unnecessary allocations. Like in <code>groupByKey</code> , the number of reduce tasks is configurable through an optional second argument.
Transformation	Meaning																		
<code>map(func)</code>	Return a new distributed dataset formed by passing each element of the source through a function <i>func</i> .																		
<code>filter(func)</code>	Return a new dataset formed by selecting those elements of the source on which <i>func</i> returns true.																		
<code>flatMap(func)</code>	Similar to map, but each input item can be mapped to 0 or more output items (so <i>func</i> should return a Seq rather than a single item).																		
<code>mapPartitions(func)</code>	Similar to map, but runs separately on each partition (block) of the RDD, so <i>func</i> must be of type <code>Iterator<T> => Iterator<U></code> when running on an RDD of type T.																		
<code>mapPartitionsWithIndex(func)</code>	Similar to mapPartitions, but also provides <i>func</i> with an integer value representing the index of the partition, so <i>func</i> must be of type <code>(Int, Iterator<T>) => Iterator<U></code> when running on an RDD of type T.																		
<code>groupByKey([numPartitions])</code>	When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs. Note: If you are grouping in order to perform an aggregation (such as a sum or average) over each key, using <code>reduceByKey</code> or <code>aggregateByKey</code> will yield much better performance. Note: By default, the level of parallelism in the output depends on the number of partitions of the parent RDD. You can pass an optional <code>numPartitions</code> argument to set a different number of tasks.																		
<code>reduceByKey(func, [numPartitions])</code>	When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function <i>func</i> , which must be of type <code>(V,V) => V</code> . Like in <code>groupByKey</code> , the number of reduce tasks is configurable through an optional second argument.																		
<code>aggregateByKey(zeroValue)(seqOp, combOp, [numPartitions])</code>	When called on a dataset of (K, V) pairs, returns a dataset of (K, U) pairs where the values for each key are aggregated using the given combine functions and a neutral "zero" value. Allows an aggregated value type that is different than the input value type, while avoiding unnecessary allocations. Like in <code>groupByKey</code> , the number of reduce tasks is configurable through an optional second argument.																		

Claim Chart – Claim 1

Claim Feature	Evidence
1 [a] partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,	Exemplary transformation operations include narrow and wide transformations. Wide transformations (also called "shuffle") operate on data from multiple partitions. Wide transformations include aggregateByKey, reduceByKey, etc. [19] Spark RDD Transformations at 2–3: RDD Transformation Types There are two types are transformations. Narrow Transformation Narrow transformations are the result of <code>map()</code> and <code>filter()</code> functions and these compute data that live on a single partition meaning there will not be any data movement between partitions to execute narrow transformations. Functions such as <code>map()</code>, <code>mapPartition()</code>, <code>flatMap()</code>, <code>filter()</code>, <code>union()</code> are some examples of narrow transformation Wider Transformation Wider transformations are the result of <code>groupByKey()</code> and <code>reduceByKey()</code> functions and <u>these compute data that live on many partitions</u> meaning there will be data movements between partitions to execute wider transformations. Since these shuffles the data, they also called shuffle transformations. Functions such as <code>groupByKey()</code>, <code>aggregateByKey()</code>, <code>aggregate()</code>, <code>join()</code>, <code>repartition()</code> are some examples of a wider transformations.

Claim Chart – Claim 1

Claim Feature	Evidence
1 [a] partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,	[20] Apache Spark Quick Start Guide: Wide transformations Wide transformations involve a shuffle of the data between the partitions. The <code>groupByKey()</code>, <code>reduceByKey()</code>, <code>join()</code>, <code>distinct()</code>, and <code>intersect()</code> are some examples of wide transformations. In the case of these transformations, <u>the result will be computed using data from multiple partitions and thus requires a shuffle.</u> Wide transformations are similar to the shuffle-and-sort phase of MapReduce. Let's understand the concept with the help of the following example:  (A) (B)

Claim Chart – Claim 1

Claim Feature	Evidence
1 [a] partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,	Another example of a mapping function is a shuffle transformation, which, when executed, runs a map task on all data partitions first. [21] Spark SQL Shuffle Partitions: What is Spark Shuffle? Shuffling is a mechanism Spark uses to <u>redistribute the data across different executors and even across machines</u>. Spark shuffling triggers for transformation operations like <code>groupByKey()</code>, <code>reduceByKey()</code>, <code>join()</code>, <code>groupByKey()</code> e.t.c Spark Shuffle is an expensive operation since it involves the following • Disk I/O • Involves data serialization and deserialization • Network I/O When <u>creating an RDD</u>, Spark doesn't necessarily store the data for all keys in a partition since at the time of creation there is no way we can set the key for the data set. Hence, when we run the <code>reduceByKey()</code> operation to aggregate the data on keys, Spark does the following. • <u>Spark first runs map tasks on all partitions which groups all values for a single key.</u> • The results of the map tasks are kept in memory. • When results do not fit in memory, Spark stores the data on a disk. • <u>Spark shuffles the mapped data across partitions, some times it also stores the shuffled data into a disk for reuse when it needs to recalculate.</u> • Run the garbage collection • Finally runs reduce tasks on each partition based on key.

Claim Chart – Claim 1

Claim Feature	Evidence
1 [a] partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,	The Accused Instrumentalities include a plurality of mapping functions that are each user-configurable. [22] Apache Spark Map vs. FlatMap at 3–4: i. Spark Map Transformation A map is a transformation operation in Apache Spark. It applies to each element of RDD and it returns the result as new RDD. <u>In the Map, operation developer can define his own custom business logic.</u> The same logic will be applied to all the elements of RDD. <u>Spark Map function takes one element as input process it according to custom code (specified by the developer) and returns one element at a time.</u> Map transforms an RDD of length N into another RDD of length N. The input and output RDDs will typically have the same number of records.

Claim Chart – Claim 1

Claim Feature	Evidence
1 [a] partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,	When a mapping function is applied, intermediate files are created for that partition. For example, as seen below, Spark takes the key values and writes that data to intermediate files on each node of the distributed system. [12] Spark in Action: 4.2.2. Understanding and avoiding unnecessary shuffling Physical movement of data between partitions is called <i>shuffling</i>. It occurs when data from multiple partitions needs to be combined in order to build partitions for a new RDD. When grouping elements by key, for example, <u>Spark needs to examine all of the RDD's partitions, find elements with the same key, and then physically group them, thus forming new partitions.</u> To visualize what happens with partitions during a shuffle, we'll use the previous example with the <u>aggregateByKey transformation</u> (from section 4.1.2). The shuffle that occurs during that transformation is illustrated in <u>figure 4.2.</u> <u>The transform function puts all values of each key in a single partition (partitions P1 to P3) into a list. Spark then writes this data to intermediate files on each node. In the next phase, the merge function is called to merge lists from different partitions, but of the same key, into a single list for each key. The default partitioner (HashPartitioner) then kicks in and puts each key in its proper partition.</u> <u>Tasks that immediately precede and follow the shuffle are called map and reduce tasks, respectively. The results of map tasks are written to intermediate files (often to the OS's filesystem cache only) and read by reduce tasks. In addition to being written to disk, the data is sent over the network, so it's important to try to minimize the number of shuffles during Spark jobs.</u>

Claim Chart – Claim 1

Claim Feature	Evidence
1 [a] partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,	In the below example, each of the plurality of data partitions has a plurality of key-value pairs. Each data partition (e.g., RDD1 – P1 on Node 1) is provided to mapping functions (e.g., the pictured transform function). The transform function is user-configurable. Each transform function outputs a plurality of list files (e.g., files 6, 2, 3, in Interim files 1). This intermediate data each correspond to a set of keys in the data partition (see the arrows between each key in RDD1 – P1 and Interim files 1). These values are identifiable by the data group. [12] Spark in Action: The diagram illustrates a shuffle operation during an <code>aggregateByKey</code> transformation on an RDD with three partitions. It shows three nodes (Node 1, Node 2, Node 3) each containing RDD1 partitions (P1, P2, P3). Each RDD1 partition is processed by a 'Transform' function (Mapping Function) to produce 'Interim files' (Intermediate Data). These interim files are then merged by a 'Merge' function to produce the final RDD2 partitions. The diagram includes labels for 'Data Partition', 'Key-Value Pairs', 'Mapping Function', 'Intermediate Data', and 'Merge'. Figure 4.2: A shuffle during the example <code>aggregateByKey</code> transformation on an RDD with three partitions. The transform function, passed to <code>aggregateByKey</code>, merges values in partitions. The merge function merges values between partitions during the shuffle phase. Intermediate files hold values merged per partition and are used during the shuffle phase.

Claim Chart – Claim 1

Claim Feature	Evidence
1 [a] partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,	In another example, Teradata Vantage includes partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group. The particular details are in AA's possession, custody, and control.  American Airlines is the world's largest airline when measured by fleet size, scheduled passengers carried, and number of destinations served. Its mission is to “care for people on life's journey.” The airline averages 6,800 flights per day to more than 365 destinations in 61 countries and in 2021, safely transported 164M passengers. This impressive airline fuels business-critical operations across multiple business departments performing analytics for revenue management, technical operations, network planning, marketing, cargo, passenger sales, and more, all using Teradata Vantage as its connected cloud data platform. Data is the fuel for American Airlines and analytics are the propulsion that lifts American Airlines above the everyday turbulence encountered by airline operators. [27] AA Using Teradata

Claim Chart – Claim 1

Claim Feature	Evidence
1 [b] wherein the data of a first data group has a different schema than the data of a second data group and the data of the first data group is mapped differently than the data of the second data group so that different lists of values are output for the corresponding different intermediate data, wherein the different schema and corresponding different intermediate data have a key in common; and	In the Accused Instrumentalities, the data of a first data group has a different schema than the data of a second data group and the data of the first data group is mapped differently than the data of the second data group so that different lists of values are output for the corresponding different intermediate data, wherein the different schema and corresponding different intermediate data have a key in common. For example, the Schema class in Hive is defined for each data set. Thus, the data of a first data group can have a different schema than the data of a second data group. [7] Class Schema: <pre>public static enum Schema._Fields extends Enum<Schema._Fields> implements org.apache.thrift.TFieldIdEnum</pre> The set of fields this struct contains, along with convenience methods for finding and manipulating them.

Claim Chart – Claim 1

Claim Feature	Evidence
1 [b] wherein the data of a first data group has a different schema than the data of a second data group and the data of the first data group is mapped differently than the data of the second data group so that different lists of values are output for the corresponding different intermediate data, wherein the different schema and corresponding different intermediate data have a key in common; and	In the Accused Instrumentalities, the data of the first data group is mapped differently than the data of the second data group. For example, data partitions can be processed by using different user-configured Mappers: [8] Hive Tutorial at 15: <u>Users can also plug in their own custom mappers and reducers in the data stream by using features natively supported in the Hive language.</u> for example, in order to run a custom mapper script - map_script - and a custom reducer script - reduce_script - the user can issue the following command which uses the TRANSFORM clause to embed the mapper and the reducer scripts. [4] HadoopMapReduce: As key-value pairs are read from the RecordReader they are passed to the configured  Mapper. The user supplied Mapper does whatever it wants with the input pair and calls  <u>OutputCollector.collect</u> with key-value pairs of its own choosing. The output it generates must use one key class and one value class. This is because the Map output will be written into a SequenceFile which has per-file type information and all the records must have the same type (use subclassing if you want to output different data structures). The Map input and output key-value pairs are not necessarily related typewise or in cardinality.

Claim Chart – Claim 1

Claim Feature	Evidence
1 [b] wherein the data of a first data group has a different schema than the data of a second data group and the data of the first data group is mapped differently than the data of the second data group so that different lists of values are output for the corresponding different intermediate data, wherein the different schema and corresponding different intermediate data have a key in common; and	The Mapper function maps the key-value pairs to form intermediate key-value pairs. Similarly, different Mappers generate different intermediate results: [2] MapReduce Tutorial: Mapper maps input key/value pairs to a set of intermediate key/value pairs. <u>Maps are the individual tasks that transform input records into intermediate records.</u> The transformed intermediate records do not need to be of the same type as the input records. A given input pair may map to zero or many output pairs.

Claim Chart – Claim 1

Claim Feature	Evidence
1 [b] wherein the data of a first data group has a different schema than the data of a second data group and the data of the first data group is mapped differently than the data of the second data group so that different lists of values are output for the corresponding different intermediate data, wherein the different schema and corresponding different intermediate data have a key in common ; and	A Hive JOIN query may be used to join two datasets that have a key in common: [9] LanguageManual Joins: Examples Some salient points to consider when writing join queries are as follows: • Complex join expressions are allowed e.g. <pre>SELECT a.* FROM a JOIN b ON (<u>a.id = b.id</u>)</pre><pre>SELECT a.* FROM a JOIN b ON (<u>a.id = b.id AND a.department = b.department</u>)</pre><pre>SELECT a.* FROM a LEFT OUTER JOIN b ON (a.id <> b.id)</pre>

Claim Chart – Claim 1

Claim Feature	Evidence
1 [b] wherein the data of a first data group has a different schema than the data of a second data group and the data of the first data group is mapped differently than the data of the second data group so that different lists of values are output for the corresponding different intermediate data, wherein the different schema and corresponding different intermediate data have a key in common; and	Spark supports multiple types of structure data, each with its own schema. [18] Learning Spark: Structured Data with Spark SQL Spark SQL is a component added in Spark 1.0 that is quickly becoming Spark's preferred way to work with structured and semistructured data. <u>By structured data, we mean data that has a schema</u>—that is, a consistent set of fields across data records. <u>Spark SQL supports multiple structured data sources as input, and because it understands their schema, it can efficiently read only the fields you require from these data sources.</u> We will cover Spark SQL in more detail in Chapter 9, but for now, we show how to use it to load data from a few common sources. Apache Hive <u>One common structured data source on Hadoop is Apache Hive. Hive can store tables in a variety of formats, from plain text to column-oriented formats, inside HDFS or other storage systems.</u> Spark SQL can load any table supported by Hive. JSON If you have <u>JSON data with a consistent schema across records</u>, Spark SQL can infer their schema and load this data as rows as well, making it very simple to pull out the fields you need. To load JSON data, first create a HiveContext as when using Hive. (No installation of Hive is needed in this case, though—that is, you don't need a <i>hive-site.xml</i> file.) Then use the <code>HiveContext.jsonFile</code> method to get an RDD of Row objects for the whole file. Apart from using the whole Row object, you can also register this RDD as a table and select specific fields from it. For example, suppose that we had a JSON file containing tweets in the format shown in Example 5-33, one per line.

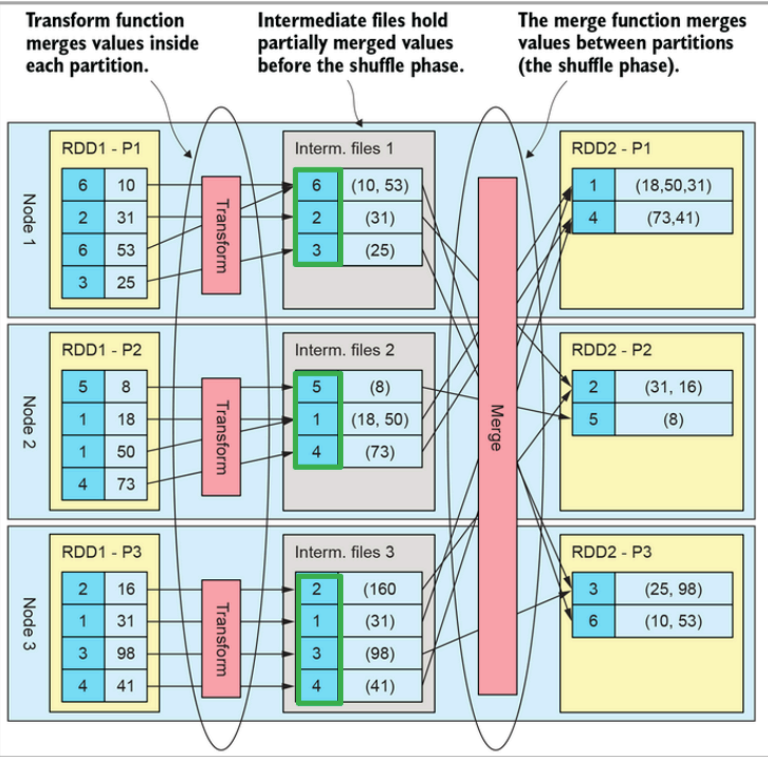
Claim Chart – Claim 1

Claim Feature	Evidence																					
1 [b] wherein the data of a first data group has a different schema than the data of a second data group and the data of the first data group is mapped differently than the data of the second data group so that different lists of values are output for the corresponding different intermediate data, wherein the different schema and corresponding different intermediate data have a key in common; and	[18] Learning Spark: File Formats Spark makes it very simple to load and save data in a large number of file formats. Formats range from <u>unstructured, like text, to semistructured, like JSON, to structured, like SequenceFiles</u> (see Table 5-1). The input formats that Spark wraps all transparently handle compressed formats based on the file extension. Table 5-1. Common supported file formats <table><tr><th>Format name</th><th>Structured</th><th>Comments</th></tr><tr><td>Text files</td><td>No</td><td>Plain old text files. Records are assumed to be one per line.</td></tr><tr><td>JSON</td><td>Semi</td><td>Common text-based format, semistructured; most libraries require one record per line.</td></tr><tr><td>CSV</td><td>Yes</td><td>Very common text-based format, often used with spreadsheet applications.</td></tr><tr><td>SequenceFiles</td><td>Yes</td><td>A common Hadoop file format used for key/value data.</td></tr><tr><td>Protocol buffers</td><td>Yes</td><td>A fast, space-efficient multilanguage format.</td></tr><tr><td>Object files</td><td>Yes</td><td>Useful for saving data from a Spark job to be consumed by shared code. Breaks if you change your classes, as it relies on Java Serialization.</td></tr></table>	Format name	Structured	Comments	Text files	No	Plain old text files. Records are assumed to be one per line.	JSON	Semi	Common text-based format, semistructured; most libraries require one record per line.	CSV	Yes	Very common text-based format, often used with spreadsheet applications.	SequenceFiles	Yes	A common Hadoop file format used for key/value data.	Protocol buffers	Yes	A fast, space-efficient multilanguage format.	Object files	Yes	Useful for saving data from a Spark job to be consumed by shared code. Breaks if you change your classes, as it relies on Java Serialization.
Format name	Structured	Comments																				
Text files	No	Plain old text files. Records are assumed to be one per line.																				
JSON	Semi	Common text-based format, semistructured; most libraries require one record per line.																				
CSV	Yes	Very common text-based format, often used with spreadsheet applications.																				
SequenceFiles	Yes	A common Hadoop file format used for key/value data.																				
Protocol buffers	Yes	A fast, space-efficient multilanguage format.																				
Object files	Yes	Useful for saving data from a Spark job to be consumed by shared code. Breaks if you change your classes, as it relies on Java Serialization.																				

Claim Chart – Claim 1

Claim Feature	Evidence
1 [b] wherein the data of a first data group has a different schema than the data of a second data group and the data of the first data group is mapped differently than the data of the second data group so that different lists of values are output for the corresponding different intermediate data, wherein the different schema and corresponding different intermediate data have a key in common; and	[13] Apache Spark FAQ at 1: How does Spark relate to Apache Hadoop? Spark is a fast and general processing engine compatible with Hadoop data. It can run in Hadoop clusters through YARN or Spark's standalone mode, and it can process data in HDFS, HBase, Cassandra, <u>Hive</u>, and any Hadoop InputFormat. It is designed to perform both batch processing (similar to MapReduce) and new workloads like streaming, interactive queries, and machine learning.
	[23] LanguageManual Transform at 1: Transform/Map-Reduce Syntax <u>Users can also plug in their own custom mappers and reducers</u> in the data stream by using features natively supported in the <u>Hive</u> language. e.g. in order to run a custom mapper script - map_script - and a custom reducer script - reduce_script - the user can issue the following command which uses the TRANSFORM clause to embed the mapper and the reducer scripts.

Claim Chart – Claim 1

Claim Feature	Evidence
1 [b] wherein the data of a first data group has a different schema than the data of a second data group and the data of the first data group is mapped differently than the data of the second data group so that different lists of values are output for the corresponding different intermediate data, wherein the different schema and corresponding different intermediate data have a key in common; and	In the Accused Instrumentalities, the different schema and corresponding different intermediate data have a key in common. [12] Spark in Action:  Figure 4.2. A shuffle during the example aggregateByKey transformation on an RDD with three partitions. The transform function, passed to aggregateByKey, merges values in partitions. The merge function merges values between partitions during the shuffle phase. Intermediate files hold values merged per partition and are used during the shuffle phase.

Claim Chart – Claim 1

Claim Feature	Evidence
1 [b] wherein the data of a first data group has a different schema than the data of a second data group and the data of the first data group is mapped differently than the data of the second data group so that different lists of values are output for the corresponding different intermediate data, wherein the different schema and corresponding different intermediate data have a key in common; and	In another example, in Teradata Vantage, the data of a first data group has a different schema than the data of a second data group and the data of the first data group is mapped differently than the data of the second data group so that different lists of values are output for the corresponding different intermediate data, wherein the different schema and corresponding different intermediate data have a key in common. For example, Teradata Vantage enables merging of data of different schema. The particular details are in AA's possession, custody, and control. <pre>SELECT select_options FROM { table_name [[AS] correlation_name] join (subquery) [AS] derived_table_name [(column_name_list)] table_function table_operator analytic_function [AS] correlation_name } [, ...] other_options</pre> Syntax Elements select_options For information on the select options, see the description of SELECT statements in <i>Teradata® Database SQL Data Manipulation Language</i>. table_name For information, see the description of SELECT statements in <i>Teradata® Database SQL Data Manipulation Language</i>. correlation_name For information, see the description of SELECT statements in <i>Teradata® Database SQL Data Manipulation Language</i>. subquery For information, see the description of SELECT statements in <i>Teradata® Database SQL Data Manipulation Language</i>. derived_table_name For information, see the description of SELECT statements in <i>Teradata® Database SQL Data Manipulation Language</i>. column_name_list For information, see the description of SELECT statements in <i>Teradata® Database SQL Data Manipulation Language</i>. analytic_function The name of the Teradata ML Engine function to be run. For information on Teradata ML Engine functions, see <i>Teradata® Vantage Machine Learning Engine Analytic Function Reference</i>, B700-4003. [26] Teradata Vantage User Guide

Claim Chart – Claim 1

Claim Feature	Evidence
1 [c] reducing the intermediate data for the data groups to at least one output data group, including processing the intermediate data for each data group in a manner that is defined to correspond to that data group, so as to result in a merging of the corresponding different intermediate data based on the key in common,	The method performed by the Accused Instrumentalities include reducing the intermediate data for the data groups to at least one output data group, including processing the intermediate data for each data group in a manner that is defined to correspond to that data group, so as to result in a merging of the corresponding different intermediate data based on the key in common. For example, the Reducer function reduces the intermediate values into a smaller set of values. This function operates in three phases: (1) Shuffle; (2) Sort; and (3) Reduce. The Shuffle and Sort phases operate on the output of various Mappers and group data before the final Reduce phase based on a common key. [2] MapReduce Tutorial: Reducer <u>Reducer reduces a set of intermediate values which share a key to a smaller set of values.</u> The number of reduces for the job is set by the user via <code>Job.setNumReduceTasks(int)</code>. Overall, Reducer implementations are passed the Job for the job via the <code>Job.setReducerClass(Class)</code> method and can override it to initialize themselves. The framework then calls <code>reduce(WritableComparable, Iterable<Writable>, Context)</code> method for each <key, (list of values)> pair in the grouped inputs. Applications can then override the <code>cleanup(Context)</code> method to perform any required cleanup.

Claim Chart – Claim 1

Claim Feature	Evidence
1 [c] reducing the intermediate data for the data groups to at least one output data group, including processing the intermediate data for each data group in a manner that is defined to correspond to that data group, so as to result in a merging of the corresponding different intermediate data based on the key in common,	For example, the intermediate data is processed such that the corresponding different intermediate data is merged based on keys that are common to individual elements of the intermediate data. [3] Interface Reducer: Reduces a set of intermediate values which share a key to a smaller set of values. The number of Reducers for the job is set by the user via <code>JobConf.setNumReduceTasks(int)</code>. Reducer implementations can access the <code>JobConf</code> for the job via the <code>JobConfigurable.configure(JobConf)</code> method and initialize themselves. Similarly they can use the <code>Closeable.close()</code> method for de-initialization. Reducer has 3 primary phases: 1. Shuffle Reducer is input the grouped output of a Mapper. In the phase the framework, for each Reducer, fetches the relevant partition of the output of all the Mappers, via HTTP. 2. Sort The framework groups Reducer inputs by keys (since different Mappers may have output the same key) in this stage. The shuffle and sort phases occur simultaneously i.e. while outputs are being fetched they are merged. Secondary Sort If equivalence rules for keys while grouping the intermediates are different from those for grouping keys before reduction, then one may specify a Comparator via <code>JobConf.setOutputValueGroupingComparator(Class)</code>. Since <code>JobConf.setOutputKeyComparatorClass(Class)</code> can be used to control how intermediate keys are grouped, these can be used in conjunction to simulate <i>secondary sort on values</i>. For example, say that you want to find duplicate web pages and tag them all with the url of the "best" known example. You would set up the job like: • Map Input Key: url • Map Input Value: document • Map Output Key: document checksum, url pagerank • Map Output Value: url • Partitioner: by checksum • OutputKeyComparator: by checksum and then decreasing pagerank • OutputValueGroupingComparator: by checksum 3. Reduce In this phase the <code>reduce(Object, Iterator, OutputCollector, Reporter)</code> method is called for each <key, (list of values)> pair in the grouped inputs.

Claim Chart – Claim 1

Claim Feature	Evidence
1 [c] reducing the intermediate data for the data groups to at least one output data group, including processing the intermediate data for each data group in a manner that is defined to correspond to that data group, so as to result in a merging of the corresponding different intermediate data based on the key in common,	The Reduce function operates on different sets of input data (intermediate data) that have a key in common. From this processing, the Reduce function populates an output file. [3] Interface Reducer: reduce <pre>void reduce(K2 key, Iterator<V2> values, OutputCollector<K3,V3> output, Reporter reporter) throws IOException</pre> <u>Reduces values for a given key.</u> The framework calls this method for each <key, (list of values)> pair in the grouped inputs. Output values must be of the same type as input values. Input keys must not be altered. The framework will reuse the key and value objects that are passed into the reduce, therefore the application should clone the objects they want to keep a copy of. In many cases, all values are combined into zero or one value. <u>Output pairs are collected with calls to <code>OutputCollector.collect(Object, Object)</code>.</u> Applications can use the Reporter provided to report progress or just indicate that they are alive. In scenarios where the application takes a significant amount of time to process individual key/value pairs, this is crucial since the framework might assume that the task has timed-out and kill that task. The other way of avoiding this is to set <code>mapreduce.task.timeout</code> to a high-enough value (or even zero for no time-outs). Parameters: key - the key. values - the list of values to reduce. output - to collect keys and combined values. reporter - facility to report progress. Throws: IOException

Claim Chart – Claim 1

Claim Feature	Evidence
1 [c] reducing the intermediate data for the data groups to at least one output data group, including processing the intermediate data for each data group in a manner that is defined to correspond to that data group, so as to result in a merging of the corresponding different intermediate data based on the key in common,	Further, in Spark, the intermediate data is reduced to an output data group. [21] Spark SQL Shuffle Partitions: What is Spark Shuffle? Shuffling is a mechanism Spark uses to redistribute the data across different executors and even across machines. Spark shuffling triggers for transformation operations like <code>groupByKey()</code>, <code>reduceByKey()</code>, <code>join()</code>, <code>groupByKey()</code> e.t.c Spark Shuffle is an expensive operation since it involves the following • Disk I/O • Involves data serialization and deserialization • Network I/O When creating an RDD, Spark doesn't necessarily store the data for all keys in a partition since at the time of creation there is no way we can set the key for the data set. Hence, when we run the <code>reduceByKey()</code> operation to aggregate the data on keys, Spark does the following. • Spark first runs <i>map</i> tasks on all partitions which groups all values for a single key. • The results of the map tasks are kept in memory. • When results do not fit in memory, Spark stores the data on a disk. • Spark shuffles the mapped data across partitions, some times it also stores the shuffled data into a disk for reuse when it needs to recalculate. • Run the garbage collection • <u>Finally runs reduce tasks on each partition based on key.</u>

Claim Chart – Claim 1

Claim Feature	Evidence
1 [c] reducing the intermediate data for the data groups to at least one output data group, including processing the intermediate data for each data group in a manner that is defined to correspond to that data group, so as to result in a merging of the corresponding different intermediate data based on the key in common,	The Accused Instrumentalities include reducing the intermediate data for the data groups to at least one output data group (e.g., a partition 1 in an RDD2). The intermediate data is processed in a manner that is defined to correspond to that data group (e.g., aggregateByKey transformation). The result is a merging of the corresponding intermediate data based on the key in common. [12] Spark in Action: 4.2.2. Understanding and avoiding unnecessary shuffling Physical movement of data between partitions is called <i>shuffling</i>. It occurs when data from multiple partitions needs to be combined in order to build partitions for a new RDD. When grouping elements by key, for example, Spark needs to examine all of the RDD's partitions, find elements with the same key, and then physically group them, thus forming new partitions. To visualize what happens with partitions during a shuffle, we'll use the previous example with the <u>aggregateByKey transformation</u> (from section 4.1.2). The shuffle that occurs during that transformation is illustrated in <u>figure 4.2.</u> The transform function puts all values of each key in a single partition (partitions P1 to P3) into a list. Spark then writes this data to intermediate files on each node. <u>In the next phase, the merge function is called to merge lists from different partitions, but of the same key, into a single list for each key.</u> The default partitioner (HashPartitioner) then kicks in and puts each key in its proper partition. <u>Tasks that immediately precede and follow the shuffle are called map and reduce tasks, respectively.</u> The results of map tasks are written to intermediate files (often to the OS's filesystem cache only) and read by <u>reduce tasks</u>. In addition to being written to disk, the data is sent over the network, so it's important to try to minimize the number of shuffles during Spark jobs.

Claim Chart – Claim 1

Claim Feature	Evidence																								
1 [c] reducing the intermediate data for the data groups to at least one output data group, including processing the intermediate data for each data group in a manner that is defined to correspond to that data group, so as to result in a merging of the corresponding different intermediate data based on the key in common,	[12] Spark in Action: The diagram illustrates the shuffle phase of an <code>aggregateByKey</code> transformation. It shows three nodes (Node 1, Node 2, Node 3) each with an RDD1 partition. A transform function merges values within each partition into intermediate files. A merge function then merges values between partitions of the same key across all nodes to produce the final RDD2 partitions. Transform function merges values inside each partition. Intermediate files hold partially merged values before the shuffle phase. The merge function merges values between partitions (the shuffle phase). Figure 4.2. A shuffle during the example <code>aggregateByKey</code> transformation on an RDD with three partitions. The transform function, passed to <code>aggregateByKey</code>, merges values in partitions. The merge function merges values between partitions during the shuffle phase. Intermediate files hold values merged per partition and are used during the shuffle phase. <table><tr><th>Node</th><th>RDD1 - P1</th><th>Transform</th><th>Interm. files</th><th>Merge</th><th>RDD2 - P1</th></tr><tr><td>Node 1</td><td>6 10 2 31 6 53 3 25</td><td></td><td>6 (10, 53) 2 (31) 3 (25)</td><td></td><td>1 (18, 50, 31) 4 (73, 41)</td></tr><tr><td>Node 2</td><td>5 8 1 18 1 50 4 73</td><td></td><td>5 (8) 1 (18, 50) 4 (73)</td><td></td><td>2 (31, 16) 5 (8)</td></tr><tr><td>Node 3</td><td>2 16 1 31 3 98 4 41</td><td></td><td>2 (160) 1 (31) 3 (98) 4 (41)</td><td></td><td>3 (25, 98) 6 (10, 53)</td></tr></table>	Node	RDD1 - P1	Transform	Interm. files	Merge	RDD2 - P1	Node 1	6 10 2 31 6 53 3 25		6 (10, 53) 2 (31) 3 (25)		1 (18, 50, 31) 4 (73, 41)	Node 2	5 8 1 18 1 50 4 73		5 (8) 1 (18, 50) 4 (73)		2 (31, 16) 5 (8)	Node 3	2 16 1 31 3 98 4 41		2 (160) 1 (31) 3 (98) 4 (41)		3 (25, 98) 6 (10, 53)
Node	RDD1 - P1	Transform	Interm. files	Merge	RDD2 - P1																				
Node 1	6 10 2 31 6 53 3 25		6 (10, 53) 2 (31) 3 (25)		1 (18, 50, 31) 4 (73, 41)																				
Node 2	5 8 1 18 1 50 4 73		5 (8) 1 (18, 50) 4 (73)		2 (31, 16) 5 (8)																				
Node 3	2 16 1 31 3 98 4 41		2 (160) 1 (31) 3 (98) 4 (41)		3 (25, 98) 6 (10, 53)																				

Claim Chart – Claim 1

Claim Feature	Evidence
1 [c] reducing the intermediate data for the data groups to at least one output data group, including processing the intermediate data for each data group in a manner that is defined to correspond to that data group, so as to result in a merging of the corresponding different intermediate data based on the key in common,	In another example, Teradata Vantage includes reducing the intermediate data for the data groups to at least one output data group, including processing the intermediate data for each data group in a manner that is defined to correspond to that data group, so as to result in a merging of the corresponding different intermediate data based on the key in common. For example, Teradata Vantage enables merging of data of different schema. The particular details are in AA's possession, custody, and control. [26] Teradata Vantage User Guide <pre> Usage Syntax: kmeans@coprocessor(ON(...) as InputTable [ON(...) as CentroidsTable] [MaxIterNum('INTEGER')] [Seed('INTEGER')] [UnpackColumns('BOOLEAN')] [InitialSeeds('STRING')] [NumClusters('INTEGER')] [Threshold('DOUBLE')] [ClusteredOutput('TABLE_NAME')] OutputTable('TABLE_NAME')) Input Columns: <Input Table> should be in the following format< id ,point1 real ...pointn real> Output Columns: <Output Table will be in the following format <clusterid, packedColumn, size, withins> Function Owner: db_superuser Creation Time: 2017-08-29 16:02:06.230091 Function Version: 7.0_rel_1.7_r65218 Interfaces Implemented: Partition function </pre>

Claim Chart – Claim 1

Claim Feature	Evidence
1 [d] wherein the mapping and reducing operations are performed by a distributed system .	The mapping and reducing operations are performed by each of the Accused Instrumentalities, which are a distributed system. [1] An introduction to Apache Hadoop for big data: The Apache Hadoop framework is composed of the following modules 1. Hadoop Common: contains libraries and utilities needed by other Hadoop modules 2. Hadoop Distributed File System (HDFS): a distributed file-system that stores data <u>on the commodity machines, providing very high aggregate bandwidth across the cluster</u> 3. <u>Hadoop YARN</u>: a resource-management platform responsible for managing compute resources in clusters and using them for scheduling of users' applications 4. <u>Hadoop MapReduce: a programming model for large scale data processing</u> <u>All the modules in Hadoop are designed with a fundamental assumption that hardware failures (of individual machines, or racks of machines) are common and thus should be automatically handled in software by the framework.</u> Apache Hadoop's MapReduce

Claim Chart – Claim 1

Claim Feature	Evidence												
1 [d] wherein the mapping and reducing operations are performed by a distributed system.	The mapping and reducing operations are performed by each of the Accused Instrumentalities, which are a distributed system. Comparison Table Between Hadoop and Teradata Below are the lists of points, describe the Differences between Hadoop and Teradata : <table><tr><th>Basis of Comparison</th><th>Teradata</th><th>Hadoop</th></tr><tr><td>Parallel Processing</td><td>Workload is divided across the system and evenly among the processors in the system.</td><td>Workload is divided among the different nodes on which relevant data is present and each node processes the task individually in parallel which reduces the overall time taken to complete the task.</td></tr><tr><td>Share-nothing Architecture</td><td>Teradata task executing in a virtual processor is independent of the tasks in other virtual processors.</td><td>Task execution on any node of the Hadoop is independent to tasks executing on other nodes.</td></tr><tr><td>Highly Scalable</td><td>More nodes/disks can be added but will increase the licensing cost.</td><td>More number of nodes/disks can be added as and when required to increase the processing and storage power.</td></tr></table>	Basis of Comparison	Teradata	Hadoop	Parallel Processing	Workload is divided across the system and evenly among the processors in the system.	Workload is divided among the different nodes on which relevant data is present and each node processes the task individually in parallel which reduces the overall time taken to complete the task.	Share-nothing Architecture	Teradata task executing in a virtual processor is independent of the tasks in other virtual processors.	Task execution on any node of the Hadoop is independent to tasks executing on other nodes.	Highly Scalable	More nodes/disks can be added but will increase the licensing cost.	More number of nodes/disks can be added as and when required to increase the processing and storage power.
Basis of Comparison	Teradata	Hadoop											
Parallel Processing	Workload is divided across the system and evenly among the processors in the system.	Workload is divided among the different nodes on which relevant data is present and each node processes the task individually in parallel which reduces the overall time taken to complete the task.											
Share-nothing Architecture	Teradata task executing in a virtual processor is independent of the tasks in other virtual processors.	Task execution on any node of the Hadoop is independent to tasks executing on other nodes.											
Highly Scalable	More nodes/disks can be added but will increase the licensing cost.	More number of nodes/disks can be added as and when required to increase the processing and storage power.											

Claim 2

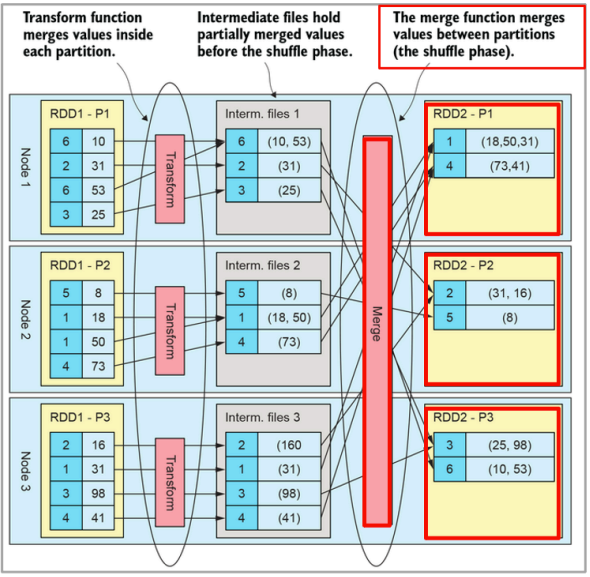
2[pre] The method of claim 1, wherein:

2[a] the at least one output data group is a plurality of output data groups.

Claim Chart – Claim 2

Claim Feature	Evidence
2 [pre] The method of claim 1, wherein:	AA infringes Claim 2 of the '610 patent. R2 incorporates herein by reference its discussion above of how AA infringes each and every element of Claim 1.

Claim Chart – Claim 2

Claim Feature	Evidence
2 [a] the at least one output data group is a plurality of output data groups.	Additionally, in the method performed by the Accused Instrumentality, the at least one output data group is a plurality of output data groups. For example, in the illustration below, RDD2 – P1 is one of the at least one output data groups. In turn, RDD2 – P1 comprises a plurality of output data groups, 1 and 4. [12] Spark Data Groups:  Figure 4.2. A shuffle during the example <u>aggregateByKey</u> transformation on an RDD with three partitions. The transform function, passed to <u>aggregateByKey</u>, merges values in partitions. The <u>merge function merges values between partitions during the shuffle phase</u>. Intermediate files hold values merged per partition and are used during the shuffle phase.

Claim Chart – Claim 2

Claim Feature	Evidence
2 [a] the at least one output data group is a plurality of output data groups.	In another example, in Teradata Vantage, the at least one output data group is a plurality of output data groups. The particular details are in AA's possession, custody, and control. [26] Teradata Vantage User Guide <pre> SELECT select_options FROM { table_name [[AS] correlation_name] join (subquery) [AS] derived_table_name [(column_name_list)] table_function table_operator analytic_function [AS] correlation_name } [, ...] other_options </pre> Syntax Elements select_options For information on the select options, see the description of SELECT statements in <i>Teradata® Database SQL Data Manipulation Language</i>. table_name For information, see the description of SELECT statements in <i>Teradata® Database SQL Data Manipulation Language</i>. correlation_name For information, see the description of SELECT statements in <i>Teradata® Database SQL Data Manipulation Language</i>. subquery For information, see the description of SELECT statements in <i>Teradata® Database SQL Data Manipulation Language</i>. derived_table_name For information, see the description of SELECT statements in <i>Teradata® Database SQL Data Manipulation Language</i>. column_name_list For information, see the description of SELECT statements in <i>Teradata® Database SQL Data Manipulation Language</i>. analytic_function The name of the Teradata ML Engine function to be run. For information on Teradata ML Engine functions, see <i>Teradata® Vantage Machine Learning Engine Analytic Function Reference</i>, B700-4003.

Claim 3

3[pre] The method of claim 1, wherein:

3[a] corresponding intermediate data for a data group being identifiable to the data group includes the corresponding intermediate data being stored such that metadata associated with a manner of storage is identifiable to the data group.

Claim Chart – Claim 3

Claim Feature	Evidence
3 [pre] The method of claim 1, wherein:	AA infringes Claim 3 of the '610 patent. R2 incorporates herein by reference its discussion above of how AA infringes each and every element of Claim 1.

Claim Chart – Claim 3

Claim Feature	Evidence
3 [a] corresponding intermediate data for a data group being identifiable to the data group includes the corresponding intermediate data being stored such that metadata associated with a manner of storage is identifiable to the data group.	Additionally, in the method performed by the Accused Instrumentalities, corresponding intermediate data for a data group being identifiable to the data group includes the corresponding intermediate data being stored such that metadata associated with a manner of storage is identifiable to the data group. For example, intermediate data (e.g., records emitted from map functions) are stored such that metadata in association with the data that is identifiable to the data group. [2] MapReduce Tutorial Mapper Mapper maps input key/value pairs to a set of intermediate key/value pairs. Maps are the individual tasks that transform input records into intermediate records. The transformed intermediate records do not need to be of the same type as the input records. A given input pair may map to zero or many output pairs. Map Parameters <u>A record emitted from a map will be serialized into a buffer and metadata will be stored into accounting buffers.</u>

Claim Chart – Claim 3

Claim Feature	Evidence
3 [a] corresponding intermediate data for a data group being identifiable to the data group includes the corresponding intermediate data being stored such that metadata associated with a manner of storage is identifiable to the data group.	In another example, in Teradata Vantage, corresponding intermediate data for a data group being identifiable to the data group includes the corresponding intermediate data being stored such that metadata associated with a manner of storage is identifiable to the data group. The particular details are in AA's possession, custody, and control. 1. Contract Phase: Teradata SQL Engine receives the SQL request In the SQL Engine, the Parsing Engine (PE) performs the SQL parsing, and sends a metadata message through QueryGrid™ fabric to the Planner Parser in the ML Engine to discover metadata for function execution. In the metadata message, in addition to the basic information about function name, function version, function type and function descriptions, and so on, it contains parameters specifying the input and output tables, and the input columns. [26] Teradata Vantage User Guide

Claim 4

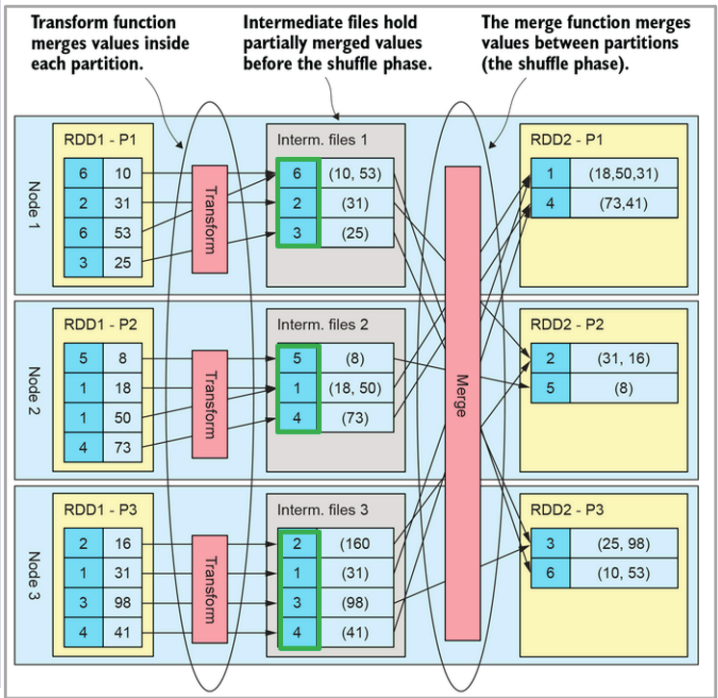
4[pre] The method of claim 1, wherein:

4[a] corresponding intermediate data for a data group being identifiable to the data group includes the corresponding intermediate data being stored in association with data that is identifiable to the data group.

Claim Chart – Claim 4

Claim Feature	Evidence
4 [pre] The method of claim 1, wherein:	AA infringes Claim 4 of the '610 patent. R2 incorporates herein by reference its discussion above of how AA infringes each and every element of Claim 1.

Claim Chart – Claim 4

Claim Feature	Evidence
4 [a] corresponding intermediate data for a data group being identifiable to the data group includes the corresponding intermediate data being stored in association with data that is identifiable to the data group.	Additionally, in the method performed by the Accused Instrumentality, corresponding intermediate data for a data group being identifiable to the data group includes the corresponding intermediate data being stored in association with data that is identifiable to the data group. The intermediate data (e.g., intermediate files) are stored in association with the data that is identifiable to the data group (e.g., group identifier) [12] Spark in Action  The diagram illustrates the Spark in Action process across three nodes (Node 1, Node 2, Node 3). Each node contains an RDD (Resilient Distributed Dataset) and a Transform function. The Transform function merges values inside each partition. The intermediate files hold partially merged values before the shuffle phase. The Merge function merges values between partitions (the shuffle phase). Transform function merges values inside each partition. Intermediate files hold partially merged values before the shuffle phase. The merge function merges values between partitions (the shuffle phase). Node 1: - RDD1 - P1: 6 10, 2 31, 6 53, 3 25 - Transform: 6 (10, 53), 2 (31), 3 (25) - Interm. files 1: 6 (10, 53), 2 (31), 3 (25) - RDD2 - P1: 1 (18,50,31), 4 (73,41) Node 2: - RDD1 - P2: 5 8, 1 18, 1 50, 4 73 - Transform: 5 (8), 1 (18, 50), 4 (73) - Interm. files 2: 5 (8), 1 (18, 50), 4 (73) - RDD2 - P2: 2 (31, 16), 5 (8) Node 3: - RDD1 - P3: 2 16, 1 31, 3 98, 4 41 - Transform: 2 (160), 1 (31), 3 (98), 4 (41) - Interm. files 3: 2 (160), 1 (31), 3 (98), 4 (41) - RDD2 - P3: 3 (25, 98), 6 (10, 53)

Claim Chart – Claim 4

Claim Feature	Evidence
4 [a] corresponding intermediate data for a data group being identifiable to the data group includes the corresponding intermediate data being stored in association with data that is identifiable to the data group.	[18] Learning Spark Spark provides special operations on RDDs containing key/value pairs. These RDDs are called pair RDDs. Pair RDDs are a useful building block in many programs, as they expose operations that allow you to act on each key in parallel or regroup data across the network. For example, pair RDDs have a <code>reduceByKey()</code> method that can aggregate data separately for each key, and a <code>join()</code> method that can merge two RDDs together by grouping elements with the same key. It is common to extract fields from an RDD (representing, for instance, an event time, customer ID, or other identifier) and use those fields as keys in pair RDD operations.
	[12] Spark in Action 4.1. WORKING WITH PAIR RDDS Storing data as key-value pairs offers a simple, general, and extensible data model because each key-value pair can be stored independently and it's easy to add new types of keys and new types of values. This extensibility and simplicity have made this practice fundamental to several frameworks and applications. For example, many popular caching systems and NoSQL databases, such as memcached and Redis, are key-value stores. Hadoop's MapReduce also operates on key-value pairs (as you can see in appendix A). Keys and values can be simple types such as integers and strings, as well as more-complex data structures. Data structures traditionally used to represent key-value pairs are <i>associative arrays</i>, also called <i>dictionaries</i> in Python and <i>maps</i> in Scala and Java. In Spark, RDDs containing key-value tuples are called <i>pair RDDs</i>. Although you don't have to use data in Spark in the form of key-value pairs (as you've seen in the previous chapters), pair RDDs are well suited (and indispensable) for many use cases. Having keys along with the data enables you to aggregate, sort, and join the data, as you'll soon see. But before doing any of that, the first step is to create a pair RDD, of course.

Claim Chart – Claim 4

Claim Feature	Evidence
4 [a] corresponding intermediate data for a data group being identifiable to the data group includes the corresponding intermediate data being stored in association with data that is identifiable to the data group.	In another example, in Teradata Vantage, corresponding intermediate data for a data group being identifiable to the data group includes the corresponding intermediate data being stored in association with data that is identifiable to the data group. The particular details are in AA’s possession, custody, and control. [26] Teradata Vantage User Guide <pre> SELECT select_options FROM { table_name [[AS] correlation_name] join (subquery) [AS] derived_table_name [(column_name_list)] table_function table_operator analytic_function [AS] correlation_name } [, ...] other_options </pre> Syntax Elements select_options For information on the select options, see the description of SELECT statements in <i>Teradata® Database SQL Data Manipulation Language</i>. table_name For information, see the description of SELECT statements in <i>Teradata® Database SQL Data Manipulation Language</i>. correlation_name For information, see the description of SELECT statements in <i>Teradata® Database SQL Data Manipulation Language</i>. subquery For information, see the description of SELECT statements in <i>Teradata® Database SQL Data Manipulation Language</i>. derived_table_name For information, see the description of SELECT statements in <i>Teradata® Database SQL Data Manipulation Language</i>. column_name_list For information, see the description of SELECT statements in <i>Teradata® Database SQL Data Manipulation Language</i>. analytic_function The name of the Teradata ML Engine function to be run. For information on Teradata ML Engine functions, see <i>Teradata® Vantage Machine Learning Engine Analytic Function Reference</i>, B700-4003.

Claim 5

5[pre] The method of claim 1, wherein:

5[a] processing the intermediate data for each data group in a manner that is defined to correspond to that data group includes, for each data group, employing an iterator that corresponds to that data group.

Claim Chart – Claim 5

Claim Feature	Evidence
5 [pre] The method of claim 1, wherein:	AA infringes Claim 5 of the '610 patent. R2 incorporates herein by reference its discussion above of how AA infringes each and every element of Claim 1.

Claim Chart – Claim 5

Claim Feature	Evidence
5 [a] processing the intermediate data for each data group in a manner that is defined to correspond to that data group includes, for each data group, employing an iterator that corresponds to that data group.	Additionally, in the method performed by the Accused Instrumentality, processing the intermediate data for each data group in a manner that is defined to correspond to the data group includes, for each data group, employing an iterator that corresponds to that data group. For example, the Accused Instrumentalities employs iterators that correspond to, e.g., a particular RDD. The particular details are in AA's possession, custody, and control. [12] Spark in Action <code>zipPartitions</code> accepts two sets of arguments. In the first set, you give it RDDs; and in the second, a function that takes a matching number of <code>Iterator</code> objects used for accessing elements in each partition. The function must return a <u>new <code>Iterator</code>, which can be a different type (matching the resulting RDD)</u>. This function has to take into account that RDDs may have different numbers of elements in partitions and guard against iterating beyond <code>Iterator</code> lengths.

Claim 17

- 17[pre]** A computer system including a plurality of computing devices, the computer system configured to process data of a data set, wherein the data set comprises a plurality of data groups, the computer system comprises at least one processor and memory that are operable to perform the following operations:
- 17[a]** partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,
- 17[b]** wherein the data of a first data group has a different schema than the data of a second data group and the data of the first data group is mapped differently than the data of the second data group so that different lists of values are output for the corresponding different intermediate data, wherein the different schema and corresponding different intermediate data have a key in common; and
- 17[c]** reduce the intermediate data for the data groups to at least one output data group, including processing the intermediate data for each data group in a manner that is defined to correspond to that data group so as to result in a merging of the corresponding different intermediate data based on the key in common.

Claim Chart – Claim 17

Claim Feature	Evidence
17 [pre] A computer system including a plurality of computing devices, the computer system configured to process data of a data set, wherein the data set comprises a plurality of data groups, the computer system comprises at least one processor and memory that are operable to perform the following operations:	AA infringes Claim 17 of the '610 patent. The Accused Instrumentalities include a plurality of computing devices, the system configured to process data of a data set, wherein the data set comprises a plurality of data groups, the computer system comprises at least one processor and memory that are operable to perform operations described below and on the following slides. As discussed for Claim 1, the Accused Instrumentality is a distributed system (e.g., Hadoop Distributed File System, Teradata Vantage, etc.) that comprises multiple clusters (i.e., a plurality of computing devices). The Accused Instrumentality comprises a data set having a plurality of data groups. [1] An introduction to Apache Hadoop for big data The Apache Hadoop framework is composed of the following modules 1. Hadoop Common: contains libraries and utilities needed by other Hadoop modules 2. Hadoop Distributed File System (HDFS): <u>a distributed file-system that stores data on the commodity machines, providing very high aggregate bandwidth across the cluster</u> 3. Hadoop YARN: a resource-management platform responsible for managing <u>compute resources in clusters</u> and using them for scheduling of users' applications 4. Hadoop MapReduce: a programming model for large scale data processing All the modules in Hadoop are designed with a fundamental assumption that hardware failures (of individual machines, or racks of machines) are common and thus should be automatically handled in software by the framework. Apache Hadoop's MapReduce

Claim Chart – Claim 17

Claim Feature	Evidence
17 [pre] A computer system including a plurality of computing devices, the computer system configured to process data of a data set, wherein the data set comprises a plurality of data groups, the computer system comprises at least one processor and memory that are operable to perform the following operations:	For example, an RDD can be one of a plurality of groups within a data set. [17] RDD Programming Guide Resilient Distributed Datasets (RDDs) <u>Spark revolves around the concept of a <i>resilient distributed dataset</i> (RDD), which is a fault-tolerant collection of elements that can be operated on in parallel. There are two ways to create RDDs: <i>parallelizing</i> an existing collection in your driver program, or <i>referencing a dataset in an external storage system</i>, such as a shared filesystem, HDFS, HBase, or any data source offering a Hadoop InputFormat.</u> Overview At a high level, every Spark application consists of a <i>driver program</i> that runs the user's main function and executes various <i>parallel operations</i> on a cluster. The main abstraction Spark provides is a <i>resilient distributed dataset</i> (RDD), which is a collection of elements <u>partitioned across the nodes of the cluster that can be operated on in parallel</u>. RDDs are created by starting with a file in the Hadoop file system (or any other Hadoop-supported file system), or an existing Scala collection in the driver program, and transforming it. Users may also ask Spark to <i>persist</i> an RDD in memory, allowing it to be reused efficiently across parallel operations. Finally, RDDs automatically recover from node failures.

Claim Chart – Claim 17

Claim Feature	Evidence
17 [a] partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,	The at least one processor and memory of the Accused Instrumentality are configured to perform the steps of partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group. The discussion for Claim element 1[a] is incorporated herein by reference.

Claim Chart – Claim 17

Claim Feature	Evidence
17 [b] wherein the data of a first data group has a different schema than the data of a second data group and the data of the first data group is mapped differently than the data of the second data group so that different lists of values are output for the corresponding different intermediate data, wherein the different schema and corresponding different intermediate data have a key in common; and	In the Accused Instrumentalities, the data of a first data group has a different schema than the data of a second data group and the data of the first data group is mapped differently than the data of the second data group so that different lists of values are output for the corresponding different intermediate data, wherein the different schema and corresponding different intermediate data have a key in common. The discussion for Claim element 1[b] is incorporated herein by reference.

Claim Chart – Claim 17

Claim Feature	Evidence
17 [c] reduce the intermediate data for the data groups to at least one output data group, including processing the intermediate data for each data group in a manner that is defined to correspond to that data group so as to result in a merging of the corresponding different intermediate data based on the key in common.	The at least one processor and memory of the Accused Instrumentality are further configured to perform the steps of reducing the intermediate data for the data groups to at least one output data group, including processing the intermediate data for each data group in a manner that is defined to correspond to that data group so as to result in a merging of the corresponding different intermediate data based on the key in common. The discussion for Claim element 1[c] is incorporated herein by reference.

Claim 18

18[pre] The computer system of claim 17, wherein:

18[a] the at least one output data group is a plurality of output data groups.

Claim Chart – Claim 18

Claim Feature	Evidence
18 [pre] The computer system of claim 17, wherein:	AA infringes Claim 18 of the '610 patent. R2 incorporates herein by reference its discussion above of how AA infringes each and every element of Claim 17.

Claim Chart – Claim 18

Claim Feature	Evidence
18 [a] the at least one output data group is a plurality of output data groups.	Additionally, in the Accused Instrumentality, the at least one output data group is a plurality of output data groups. The discussion for Claim 2 is incorporated herein by reference.

Claim 19

19[pre] The computer system of claim 17, wherein:

19[a] corresponding intermediate data for a data group being identifiable to the data group includes the corresponding intermediate data being stored such that metadata associated with the manner of storage is identifiable to the data group.

Claim Chart – Claim 19

Claim Feature	Evidence
19 [pre] The computer system of claim 17, wherein:	AA infringes Claim 19 of the '610 patent. R2 incorporates herein by reference its discussion above of how AA infringes each and every element of Claim 17.

Claim Chart – Claim 19

Claim Feature	Evidence
19 [a] corresponding intermediate data for a data group being identifiable to the data group includes the corresponding intermediate data being stored such that metadata associated with the manner of storage is identifiable to the data group.	Additionally, in the Accused Instrumentality, corresponding intermediate data for a data group being identifiable to the data group includes the corresponding intermediate data being stored such that metadata associated with a manner of storage is identifiable to the data group. The discussion for Claim 3 is incorporated herein by reference.

Claim 20

20[pre] The computer system of claim 17, wherein:

20[a] corresponding intermediate data for a data group being identifiable to the data group includes the computer system being configured to store the corresponding intermediate data in association with data that is identifiable to the data group.

Claim Chart – Claim 20

Claim Feature	Evidence
20 [pre] The computer system of claim 19, wherein:	AA infringes Claim 20 of the '610 patent. R2 incorporates herein by reference its discussion above of how AA infringes each and every element of Claim 17.

Claim Chart – Claim 20

Claim Feature	Evidence
20 [a] corresponding intermediate data for a data group being identifiable to the data group includes the computer system being configured to store the corresponding intermediate data in association with data that is identifiable to the data group	Additionally, in the Accused Instrumentality, corresponding intermediate data for a data group being identifiable to the data group includes the computer system being configured to store the corresponding intermediate data in association with data that is identifiable to the data group. The discussion for Claim 4 is incorporated herein by reference.

Claim 21

21[pre] The computer system of claim 17, wherein:

21[a] the at least one processor and memory being configured to process the intermediate data for each data group in a manner that is defined to correspond to that data group includes, for each data group, the at least one processor and memory being configured to employ an iterator that corresponds to that data group.

Claim Chart – Claim 21

Claim Feature	Evidence
21 [pre] The computer system of claim 17, wherein:	AA infringes Claim 21 of the '610 patent. R2 incorporates herein by reference its discussion above of how AA infringes each and every element of Claim 17.

Claim Chart – Claim 21

Claim Feature	Evidence
21 [a] the at least one processor and memory being configured to process the intermediate data for each data group in a manner that is defined to correspond to that data group includes, for each data group, the at least one processor and memory being configured to employ an iterator that corresponds to that data group.	Additionally, in the Accused Instrumentality, the at least one processor and memory being configured to process the intermediate data for each data group in a manner that is defined to correspond to the data group includes, for each data group, the at least one processor and memory being configured to employ an iterator that corresponds to that data group. The discussion for Claim 5 is incorporated herein by reference.

END

