

**IN THE UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF TEXAS
MIDLAND DIVISION**

INTELLECTUAL VENTURES I LLC and	)	
INTELLECTUAL VENTURES II LLC,	)	
	)	
<i>Plaintiffs,</i>	)	C.A. No. 7:24-cv-00277-ADA
	)	
v.	)	
	)	JURY TRIAL DEMANDED
SOUTHWEST AIRLINES CO.,	)	
	)	
<i>Defendant.</i>	)	

**PLAINTIFFS' DISCLOSURE OF
PRELIMINARY INFRINGEMENT CONTENTIONS**

Pursuant to the Court's Standing Order Governing Proceedings ("OGP"), Plaintiffs Intellectual Ventures I LLC and Intellectual Ventures II LLC (collectively, "Intellectual Ventures," "IV," or "Plaintiffs"), hereby provides its Initial Infringement Contentions to Defendant Southwest Airlines Co. ("Southwest" or "Defendant") for United States Patent No. 8,332,844 ("the '844 Patent"), United States Patent No. 8,407,722 ("the '722 Patent"), United States Patent No. 7,949,785 ("the '785 Patent"), United States Patent No. 8,027,326 ("the '326 Patent"), United States Patent No. 7,324,469 ("the '469 Patent"), and United States Patent No. 7,257,582 ("the '582 Patent") (collectively, the "Asserted Patents" or "Patents-in-Suit").

IV makes this disclosure based on the information presently available to it. Discovery in this case has not started, and IV reserves its right to amend or supplement these disclosures as permitted by the Federal Rules of Civil Procedure, by the local rules of the Western District of Texas, and by order of the Court.

For each Asserted Patent, Plaintiffs identify the following Accused Instrumentalities of which it is currently aware. The identification of Accused Instrumentalities is based on Plaintiffs' research and analysis to date, without the benefit of discovery from the Defendant. Plaintiffs

reserve the right to add, delete, substitute or otherwise amend this list of Accused Instrumentalities based on discovery or other circumstances, in a manner consistent with the Federal Rules of Civil Procedures, local rules, and standing order.

The Accused Instrumentalities include, without limitation, the following:

- '844 Patent. The Accused Instrumentalities include, without limitation, Southwest systems that utilize Docker, and all past, current and future Southwest products that operate in the same or substantially similar manner as the specifically identified products and components, and all past, current and future Southwest products that have the same or substantially similar features as the specifically identified products and components provided in Exhibit 1.
- '722 Patent. The Accused Instrumentalities include, without limitation, Southwest systems that utilize Kafka, and all past, current and future Southwest products that operate in the same or substantially similar manner as the specifically identified products and components, and all past, current, and future Southwest products that have the same or substantially similar features as the specifically identified products and components provided in Exhibit 2.
- '785 Patent. The Accused Instrumentalities include, without limitation, Southwest systems that utilize Kubernetes, and all past, current and future Southwest products that operate in the same or substantially similar manner as the specifically identified products and components, and all past, current and future Southwest products that have the same or substantially similar features as the specifically identified products and components provided in Exhibit 3.
- '326 Patent. The Accused Instrumentalities include, without limitation, Southwest systems that provide Wi-Fi Access Points that support at least IEEE 802.11n and/or 802.11ac, and all past, current, and future Southwest products that operate in the same or substantially similar manner as the specifically identified products and components, and all past, current and future Southwest products that have the same or substantially similar features as the specifically identified products and components. Exhibit 4.
- '469 Patent. The Accused Instrumentalities include, without limitation, Southwest systems that provide satellite-based onboard WiFi, and all past, current and future Southwest products that operate in the same or substantially similar manner as the specifically identified products and components, and all past, current and future Southwest products that have the same or substantially similar features as the specifically identified products and components. Exhibit 5.
- '582 Patent. The Accused Instrumentalities include, without limitation, Southwest systems that utilize Hadoop and/or Spark, and all past, current and future Southwest products that operate in the same or substantially similar manner as the specifically identified products and components, and all past, current and future Southwest products that have the same or substantially similar features as the specifically identified products and components provided in Exhibits 6 and 7.
- All past, current and future Southwest products and services that operate in the same or substantially similar manner as the specifically identified products and services above and

described in Exhibits 1-7 (claim charts identifying specifically where each element of each asserted claim is found within each Accused Instrumentality).

- All past, current and future Southwest products and services that have the same or substantially similar features as the specifically identified products and services above and described in Exhibits 1-7, (claim charts identifying specifically where each element of each asserted claim is found within each Accused Instrumentality).

Plaintiffs' infringement contentions apply to the Accused Instrumentalities as well as all other past, current and future hardware and software products and services developed, made, used, offered for sale, sold, imported, and provided by Southwest that contain or make use of the Patented Technology.¹ This identification of Accused Instrumentalities is based upon Plaintiffs' diligent investigation to date, and without the benefit of any discovery from the Defendant and without the benefit of the Court's claim construction. Plaintiffs expressly reserve the right to add, delete, substitute or otherwise amend this list of Accused Instrumentalities based on discovery, the Court's claim construction, or other circumstances, in a manner consistent with the Federal Rules of Civil Procedure and the Court's rules, including the local Patent Rules.

These preliminary infringement contentions are based upon publicly available information and Plaintiffs' diligent investigation to date, and without the benefit of any substantial discovery from the Defendant and without the benefit of the Court's claim construction. Plaintiffs anticipate that discovery may provide additional evidence of Defendant's infringement, may lead to the discovery of additional instances of infringement, and may also enable identification of additional claims that are infringed by Defendant.

Based upon currently available information, IV asserts that Southwest has infringed and/or continues to infringe the patents and claims identified in the attached claim charts (the "Asserted Claims" of the Patents-in-Suit). Infringement claim charts evidencing the correspondence between (i) the elements of the Asserted Claims, and (ii) the corresponding items of the Accused

¹ "Patented Technology" means all technologies described in the AA/ASW/ALP-1036, p. 3 of 61
American Airlines, et. al. v. Intellectual Ventures, et. al.
IPR2025-00785

Instrumentalities are attached hereto. Further, Exhibits 1-7, which are attached hereto and incorporated by reference, are exemplary infringement claim charts identifying specifically where each limitation of each Asserted Claim is found within each Accused Instrumentality or practiced by each Accused Instrumentality.

Plaintiffs assert that Defendant has directly infringed and continues to directly infringe the Asserted Claims literally through the Accused Instrumentalities by making, using, offering for sale, selling, and/or importing into the United States the Accused Instrumentalities. To the extent that Defendant alleges that one or more limitations of the Asserted Claims are not literally found in the Accused Instrumentalities, Plaintiffs allege that such limitations are found in or practiced by the Accused Instrumentalities under the doctrine of equivalents. Any differences alleged to exist between any of the Asserted Claims and any of the Accused Instrumentalities are insubstantial and that each Accused Instrumentality also meets each limitation under the doctrine of equivalents as the identified features of the Accused Instrumentality performs substantially the same function in substantially the same way to achieve substantially the same result as the corresponding claim limitation. IV reserves the right to assert infringement solely under the doctrine of equivalents with respect to any particular claim element(s), if warranted by discovery, further analysis, and/or claim constructions in this case.

These assertions are based upon Plaintiffs' diligent investigation to date, and without the benefit of any substantial discovery from the Defendant and without the benefit of the Court's claim construction. Plaintiffs reserve the right to add, delete, substitute or otherwise amend these infringement allegations based on discovery, the Court's claim construction, or other circumstances, in a manner consistent with the Federal Rules of Civil Procedure and the Court's rules, including the local Patent Rules.

Plaintiffs further assert that Defendant has indirectly infringed and continues to indirectly infringe by actively inducing infringement of one or more of the claims of the Asserted Patents through the Accused Instrumentalities. *See Exhibits 1-7.*

Plaintiffs also assert that these third parties directly infringe at least one or more of the claims of the Asserted Patents through the manufacture, use, sale, offer to sell, or importation of the Accused Instrumentalities. *See Exhibits 1-7.*

For example, Defendant has actively induced infringement by encouraging the use of the Accused Instrumentalities in ways that infringe each Asserted Claim. Defendant knew or should have known that such encouragement would induce infringement. Defendant has taken active steps with the specific intent to encourage and cause others to use each Accused Instrumentality in ways that infringes each Asserted Claim. Such active steps by Defendant with specific intent to induce infringement have included, among other things, advertising, promoting, marketing, making available for use, offering to sell, and/or selling the Accused Instrumentalities to others; encouraging and influencing others to import, offer to sell, and/or sell the Accused Instrumentalities; directing and instructing others to use the Accused Instrumentalities in infringing ways, including providing and promoting the described hardware and/or software components and features that when used in their normal and customary ways as intended and designed; and by providing the Accused Instrumentalities to others. Southwest has performed the aforementioned steps with the knowledge of the Asserted Patents through receipt of a letter dated September 30, 2024, and received by Defendant on the same date, and also as of the date when the complaint in this case was filed. Southwest has known or should have known that the acts it has induced constitute infringement because, for instance, it has been aware that its customers, employees, and/or third parties, such as vendors, will use the Accused Instrumentalities, resulting in direct infringement.

IV further reserves the right to supplement or revise its infringement contentions and/or charts. Such supplement may be based upon, for example, further discovery. Further, IV reserves the right to supplement or revise its infringement contentions and/or chart(s), including identification of additional asserted claims, based on, for example, new versions or variations of one or more of the accused products that are later discovered.

The Asserted Claims of the Patents in Suit are entitled to at least the priority dates identified below:

- Each of the Asserted Claims of the '844 Patent is entitled to a priority date of no later than December 30, 2004.
- Each of the Asserted Claims of the '722 Patent is entitled to a priority date of no later than December 18, 2000.
- Each of the Asserted Claims of the '785 Patent is entitled to a priority date of no later than March 21, 2003.
- Each of the Asserted Claims of the '326 Patent is entitled to a priority date of no later than January 12, 2004.
- Each of the Asserted Claims of the '469 Patent is entitled to a priority date of no later than September 29, 2003.
- Each of the Asserted Claims of the '582 Patent is entitled to a priority date of no later than March 13, 2002.

The subject matter described by the Asserted Claims may have been conceived and reduced to practice prior to these dates. Plaintiffs' research and analysis is ongoing, and Plaintiffs reserve the right to assert that the claims are entitled to a priority date that is earlier than the above dates.

Dated: February 10, 2025

Respectfully submitted,

By: /s/ Mark D. Siegmund
Mark D. Siegmund (TX Bar No. 24117055)
msiegmund@cjsjlaw.com
CHERRY JOHNSON SIEGMUND
JAMES PLLC
7901 Fish Pond Road
Ste 2nd Floor
Waco, TX 76710
254-732-2242
Fax: 866-627-3509
Email: msiegmund@cjsjlaw.com

Jonathan K. Waldrop (CA Bar No. 297903)
(Admitted in this District)
jwaldrop@kasowitz.com
Darcy L. Jones (CA Bar No. 309474)
(Admitted in this District)
djones@kasowitz.com
Marcus A. Barber (CA Bar No. 307361)
(Admitted in this District)
mbarber@kasowitz.com
John W. Downing (CA Bar No. 252850)
(Admitted in this District)
jdowning@kasowitz.com
Heather S. Kim (CA Bar No. 277686)
(Admitted in this District)
hkim@kasowitz.com
ThucMinh Nguyen (CA Bar No. 304382)
(Admitted in this District)
tnguyen@kasowitz.com
Jonathan Hicks (CA Bar No. 274634)
(Admitted in this District)
jhicks@kasowitz.com
KASOWITZ BENSON TORRES LLP
333 Twin Dolphin Drive, Suite 200
Redwood Shores, California 94065
Telephone: (650) 453-5170
Facsimile: (650) 453-5171

Attorneys for Plaintiffs
INTELLECTUAL VENTURES I LLC and
INTELLECTUAL VENTURES II LLC

CERTIFICATE OF SERVICE

I hereby certify that a true and correct copy of the foregoing instrument was served or delivered electronically, via email, to all counsel of record, on this 10th day of February 2025.

/s/ Jonathan K. Waldrop
Jonathan K. Waldrop

**Exhibit 6 to
Intellectual Ventures I LLC's and Intellectual Ventures II LLC's
Preliminary Infringement Contentions**

**Infringement Claim Chart of
U.S. Patent No. 7,257,582 (“’582 Patent”)**

The Accused Systems and Services include without limitation Southwest systems and services that utilize Hadoop; all past, current, and future systems and services that operate in the same or substantially similar manner as the specifically identified systems and services; and all past, current, and future Southwest systems and services that have the same or substantially similar features as the specifically identified systems and services (“Example Southwest Count 6 Systems and Services” or “Southwest Systems and Services”).¹ On information and belief, Southwest’s Systems and Services directly infringe the asserted claims of the ’582 Patent, either literally or under the doctrine of equivalents, through at least using Hadoop in its various systems. In addition, Southwest directly infringes the ’582 Patent by testing the Southwest Systems and Services and its various systems that use Hadoop. On information and belief, Southwest, with knowledge at least since the time of filing the complaint, also indirectly infringes the ’582 Patent by inducing its employees, customers, and other third parties (such as vendors) to use Southwest Systems and Services, and provides documents that include instructions regarding how to use Southwest products and services that use Hadoop in the Southwest Systems and Services in an infringing manner. On information and belief, Southwest supplies its employees, customers, and other third parties with access to the Southwest Systems and Services through various products and services that allows its employees, customers, and other third parties to use the claimed method. On information and belief, Southwest provides instructions on how to use its various products and services that use Hadoop in the Southwest Systems and Services in an infringing manner, the use of which results in infringement of the ’582 Patent claims through performance of the claimed methods.

Consistent with this Court’s Standing Order Governing Proceedings (OGP), IV identifies Hadoop as used in Southwest’s systems and offerings as the Accused Instrumentality. Information regarding how specifically Southwest implements Hadoop, including for which products and offerings, is non-public. IV intends to obtain discovery regarding how the Hadoop is used in Southwest products and services and will update its infringement positions once discovery is completed.

IV does not intend this exemplary claim chart to be limiting, and IV reserves its rights to pursue other accused instrumentalities, patent claims, evidence, and infringement arguments in this case. Discovery has yet to begin, and this case is still in its initial stages. Accordingly, IV reserves the right to amend and/or supplement these contentions to the full extent allowed by the Court, including but not limited to, incorporating additional information, claims, theories, and / or accused products.

¹ For the avoidance of doubt, Plaintiffs do not accuse public clouds of Southwest if those services are provided by a cloud provider with a license to Plaintiffs’ patents that covers Southwest’s activities. IV will provide relevant license agreements for cloud providers in discovery. To the extent any of these licenses are relevant to Southwest’s activities, Plaintiffs will meet and confer with Southwest about the impact of such license(s).

On information and belief, the Southwest Systems and Services use Hadoop in Southwest's private cloud(s). For example, Southwest posts, or has posted, job opportunities that require familiarity with Hadoop concepts.

See <https://www.linkedin.com/in/meghanaa-malleboina/>, job profile of data engineer who stated that they enhanced Hadoop YARN cluster efficiency by 35% through optimized resource allocation.

See <https://www.linkedin.com/in/rohan-kumar-9861bb127>, job profile of Senior Hadoop Developer who stated that they are/were a Hadoop developer for Southwest Airlines. (last accessed 10/8/24).

See <https://www.linkedin.com/in/harish-t-b82094128>, job profile of Senior Hadoop Developer who stated that they are/were a Hadoop developer for Southwest Airlines. (last accessed 10/8/24).

As another example, Southwest has stated that it is investing in cloud technology and has "moved about 50% of its technology" to the cloud and has indicated cloud migration is one of its areas of focus for 2024 and beyond. Source:

<https://www.phocuswire.com/southwest-airlines-cio-tech-investment>.

On information and belief, other information confirms Southwest uses Hadoop technology.

How Data Analytics Make Airlines Fly

Look up at the sky and think in numbers: Each day, nearly 7,000 commercial aircraft take off on 24,000 flights, according to [the Federal Aviation Administration](#). With the expert guidance of 14,000 air traffic controllers, they fly about 2.2 million passengers every 24 hours. To do that, airlines and controllers coordinate airplanes' movements through internal dispatch offices and 476 control towers. Add to the mix some 200,000 general aviation aircraft traveling between more than 19,000 airports, and you can envision the logistical challenges involved in moving [719 million passengers](#) around the U.S. each year. It doesn't take much to imagine the complex logistics involved in making a system of such scale work properly. While "data analytics" is a relatively new term to the mainstream, airlines have been applying such principles to the business for years, often under the phrase "operations modeling," said [Doug Gray, director of enterprise data analytics for Dallas-based Southwest Airlines](#). At first, the data was used to guide decisions about fueling, crew schedules and flight itineraries. Today, analytics are used across the organization, in functions from marketing to operations, explained Gray, who's also a member of the International Institute for Analytics' Expert Network. "They have a significant impact on costs and profitability," he said. Indeed, calling real-time analytics "integral" to any airline's success isn't going too far. While specialists in fueling, crew scheduling, flight scheduling and other areas may have their plans laid out perfectly, their work is never immune from unexpected developments. "Daily operations is where stuff hits the fan," Gray observed. "You can't predict a heart attack in-flight, or a control-tower fire." Schedulers have only so much advance warning of bad weather. Even on the best of days, it seems an airline's plans can only be regarded as tentative.

Industry Expertise Helps

Of course, airlines need people who can implement and maintain analytics systems—software engineers and developers who need to have at least some expertise with data, Gray said. For his part, Gray is particularly interested in tech pros familiar with Oracle, Teradata and Amazon Cloud Services. Also important are relational database skills such as NoSQL and Mongo DB. Although he sees the company experimenting more with Hadoop (“It’s better in the cloud,”), Gray said that “unless something better comes along,” the company will continue to rely heavily on R and Alteryx, a “data science self-service desktop” that combines ETL, R and visualization in one GUI-driven application. Like many employers, Southwest will hire “the right person and train them” on needed skills, especially if they’re just out of school, Gray said. And, he believes, industry experience can give candidates an advantage. As more people pursue careers using data science, analytics and operations research, “more departments will have their own [data and analytics] experts, joined together by a center of excellence.” “Our biggest constraint isn’t data,” he added. “It’s teaming up the right data people with the right subject-matter experts. There’s going to be a battle for talent, and we need people with a passion for the airline business.”

Source: <https://www.dice.com/career-advice/how-data-analytics-airlines-fly>.²

² All sources cited in this document were publicly accessible as of the filing date of the Complaint.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim 1	Example Southwest Count 6 Systems and Services
1. A method of effecting on a preexisting input file a computer-executable process comprised of a plurality of subtasks, the method comprising the steps of:	To the extent this preamble is limiting, on information and belief, the Southwest Count 6 Systems and Services practice a method of effecting on a preexisting input file a computer-executable process comprised of a plurality of subtasks. On information and belief, Hadoop includes MapReduce, a software framework that processes multiple map tasks in parallel. An input data set is split into multiple chunks and each chunk is processed by a map task. Overview Hadoop MapReduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner. A MapReduce <i>job</i> usually splits the input data-set into independent chunks which are processed by the <i>map tasks</i> in a completely parallel manner. 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. Typically the compute nodes and the storage nodes are the same, that is, the MapReduce framework and the Hadoop Distributed File System (see HDFS Architecture Guide) are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim 1	Example Southwest Count 6 Systems and Services
	The diagram illustrates the Hadoop MapReduce architecture. On the left, two 'Client' nodes (red and blue ovals) send 'Job Submission' (dashed arrows) to a central 'Resource Manager' (yellow rounded rectangle). The Resource Manager communicates with three 'Node Manager' nodes (yellow rounded rectangles) on the right. Each Node Manager contains one or more 'Container' nodes (pink ovals) and an 'App Mstr' (blue oval). 'Node Status' (dotted arrows) are sent from each Node Manager to the Resource Manager. 'Resource Request' (dashed arrows with arrowheads) are sent from the Resource Manager to the App Mstrs. 'MapReduce Status' (solid arrows) are sent from the App Mstrs back to the Resource Manager. A legend at the bottom left defines the arrow types: MapReduce Status (solid arrow), Job Submission (dashed arrow), Node Status (dotted arrow), and Resource Request (dashed arrow with arrowhead). Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.
(a) automatically determining file allocation and logically subdividing records of said input file into a plurality of partitions;	On information and belief, the Southwest Count 6 Systems and Services practice automatically determining file allocation and logically subdividing records of said input file into a plurality of partitions. On information and belief, Hadoop's InputFormat describes the input-specification for a MapReduce job. The input file is logically split into multiple InputSplit instances.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim 1	Example Southwest Count 6 Systems and Services
	 Job Input InputFormat describes the input-specification for a MapReduce job. The MapReduce framework relies on the InputFormat of the job to: 1. Validate the input-specification of the job. 2. Split-up the input file(s) into logical InputSplit instances, each of which is then assigned to an individual Mapper. 3. Provide the RecordReader implementation used to glean input records from the logical InputSplit for processing by the Mapper. The default behavior of file-based InputFormat implementations, typically sub-classes of FileInputFormat, is to split the input into <i>logical</i> InputSplit instances based on the total size, in bytes, of the input files. However, the FileSystem blocksize of the input files is treated as an upper bound for input splits. A lower bound on the split size can be set via <code>mapreduce.input.fileinputformat.split.minsize</code>. Clearly, logical splits based on input-size is insufficient for many applications since record boundaries must be respected. In such cases, the application should implement a RecordReader, who is responsible for respecting record-boundaries and presents a record-oriented view of the logical InputSplit to the individual task. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. On information and belief, MapReduce InputSplit length is measured in bytes and InputSplit has storage locations. MapReduce uses storage locations to place map tasks close InputSplit's data. Map tasks are generally processed in the order of the size of the splits so larger jobs are processed first. Sources: https://data-flair.training/blogs/hadoop-optimization/; https://data-flair.training/blogs/inputsplit-in-hadoop-mapreduce/. See also <code>org.apache.hadoop.mapreduce.JobSubmitter.java</code> at lines 589-606; <code>org.apache.hadoop.mapreduce.v2.app.job.impl.JobImpl.java</code> at lines 1544-1561.
(b) distributing descriptions of all of said partitions to each of a plurality of subtask processors	On information and belief, the Southwest Count 6 Systems and Services practice distributing descriptions of all of said partitions to each of a plurality of subtask processors. On information and belief, Hadoop's InputSplit represents data that presents a byte oriented view of the input. A recordreader converts the byte oriented view of the data to a record oriented view and presents

U.S. Patent No. 7,257,582 (Claim 1)	
Claim 1	Example Southwest Count 6 Systems and Services
	it to a mapper. Mappers run inside a compute node. Typically the compute nodes and the storage nodes are the same, that is, the MapReduce framework and the Hadoop Distributed File System (see HDFS Architecture Guide) are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. ➤ Job Input InputFormat describes the input-specification for a MapReduce job. The MapReduce framework relies on the InputFormat of the job to: 1. Validate the input-specification of the job. 2. Split-up the input file(s) into logical InputSplit instances, each of which is then assigned to an individual Mapper. 3. Provide the RecordReader implementation used to glean input records from the logical InputSplit for processing by the Mapper. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim 1	Example Southwest Count 6 Systems and Services
	If TextInputFormat is the InputFormat for a given job, the framework detects input-files with the .gz extensions and automatically decompresses them using the appropriate CompressionCodec. However, it must be noted that compressed files with the above extensions cannot be <i>split</i> and each compressed file is processed in its entirety by a single mapper. InputSplit InputSplit represents the data to be processed by an individual Mapper. Typically InputSplit presents a byte-oriented view of the input, and it is the responsibility of RecordReader to process and present a record-oriented view. FileSplit is the default InputSplit. It sets mapreduce.map.input.file to the path of the input file for the logical split. RecordReader RecordReader reads <key, value> pairs from an InputSplit. Typically the RecordReader converts the byte-oriented view of the input, provided by the InputSplit, and presents a record-oriented to the Mapper implementations for processing. RecordReader thus assumes the responsibility of processing record boundaries and presents the tasks with keys and values. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim 1	Example Southwest Count 6 Systems and Services
	HDFS Architecture The diagram illustrates the HDFS Architecture. At the top, a box labeled 'Metadata (Name, replicas, ...): /home/foo/data, 3, ...' is connected to a 'Namenode' box. The 'Namenode' is connected to a 'Client' (orange oval) via 'Metadata ops' (dashed arrow) and to a group of 'Datanodes' via 'Block ops' (solid arrow). The 'Datanodes' are organized into 'Rack 1' and 'Rack 2'. 'Rack 1' contains three 'Datanodes' and 'Rack 2' contains two 'Datanodes'. Each 'Datanode' contains 'Blocks' (green squares). A 'Client' (orange oval) is shown at the bottom, performing 'Read' and 'Write' operations. 'Replication' is shown between 'Datanodes' in 'Rack 1' and 'Rack 2'. Source: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html.
(c) simultaneously executing at least a respective one of the subtasks of the computer-executable process in each of at least some of said processors on a respective one of the partitions with each subtask reading and processing the respective	On information and belief, the Southwest Count 6 Systems and Services practice simultaneously executing at least a respective one of the subtasks of the computer-executable process in each of at least some of said processors on a respective one of the partitions with each subtask reading and processing the respective partition so as to process the respective partition and produce respective subtask output. On information and belief, Multiple map tasks are executed in parallel. Each Maptask processes a respective InputSplit.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim 1	Example Southwest Count 6 Systems and Services
partition so as to process the respective partition and produce respective subtask output and;	Overview Hadoop MapReduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner. A MapReduce <i>job</i> usually splits the input data-set into independent chunks which are processed by the <i>map tasks</i> in a completely parallel manner. 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. Typically the compute nodes and the storage nodes are the same, that is, the MapReduce framework and the Hadoop Distributed File System (see HDFS Architecture Guide) are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. Job Input <code>InputFormat</code> describes the input-specification for a MapReduce job. The MapReduce framework relies on the <code>InputFormat</code> of the job to: 1. Validate the input-specification of the job. 2. Split-up the input file(s) into logical <code>InputSplit</code> instances, each of which is then assigned to an individual Mapper. 3. Provide the <code>RecordReader</code> implementation used to glean input records from the logical <code>InputSplit</code> for processing by the Mapper. Source: Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim 1	Example Southwest Count 6 Systems and Services
	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. The Hadoop MapReduce framework spawns one map task for each InputSplit generated by the InputFormat for the job. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. How Many Maps? The number of maps is usually driven by the total size of the inputs, that is, the total number of blocks of the input files. The right level of parallelism for maps seems to be around 10-100 maps per-node, although it has been set up to 300 maps for very cpu-light map tasks. Task setup takes a while, so it is best if the maps take at least a minute to execute. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. On information and belief, Each MapTask generates an intermediate map output. Overview Hadoop MapReduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner. A MapReduce <i>job</i> usually splits the input data-set into independent chunks which are processed by the <i>map tasks</i> in a completely parallel manner. 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. Typically the compute nodes and the storage nodes are the same, that is, the MapReduce framework and the Hadoop Distributed File System (see HDFS Architecture Guide) are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim 1	Example Southwest Count 6 Systems and Services
	Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. InputSplit InputSplit represents the data to be processed by an individual Mapper. Typically InputSplit presents a byte-oriented view of the input, and it is the responsibility of RecordReader to process and present a record-oriented view. FileSplit is the default InputSplit. It sets mapreduce.map.input.file to the path of the input file for the logical split. RecordReader RecordReader reads <key, value> pairs from an InputSplit. Typically the RecordReader converts the byte-oriented view of the input, provided by the InputSplit, and presents a record-oriented to the Mapper implementations for processing. RecordReader thus assumes the responsibility of processing record boundaries and presents the tasks with keys and values. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. 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. The Hadoop MapReduce framework spawns one map task for each InputSplit generated by the InputFormat for the job. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim 1	Example Southwest Count 6 Systems and Services
	Data Compression Hadoop MapReduce provides facilities for the application-writer to specify compression for both intermediate map-outputs and the job-outputs i.e. output of the reduces. It also comes bundled with CompressionCodec implementation for the zlib compression algorithm. The gzip, bzip2, snappy, and lz4 file format are also supported. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. On information and belief, MapReduce InputSplit length is measured in bytes and InputSplit has storage locations. MapReduce uses storage locations to place map tasks close InputSplit's data. Map tasks are generally processed in the order of the size of the splits so larger jobs are processed first. Sources: https://data-flair.training/blogs/hadoop-optimization/; https://data-flair.training/blogs/inputsplit-in-hadoop-mapreduce/. See also org.apache.hadoop.mapreduce.JobSubmitter.java at lines 589-606; org.apache.hadoop.mapreduce.v2.app.job.impl.JobImpl.java at lines 1544-1561.
(d) thereafter repeating step (c) in at least some of the subtask processors each with another unprocessed partition on a first-come/first-served basis; and	On information and belief, the Southwest Count 6 Systems and Services practice repeating step (c) in at least some of the subtask processors each with another unprocessed partition on a first-come/first-served basis. On information and belief, Hadoop includes a FIFO scheduler. Each mapper processes an InputSplit at a time. For example, a mapper processes one line of the word count application at a time.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim 1	Example Southwest Count 6 Systems and Services
	Overview  Overview The Yarn scheduler is a fertile area of interest with different implementations, e.g., Fifo, Capacity and Fair schedulers. Meanwhile, several optimizations are also made to improve scheduler performance for different scenarios and workload. Each scheduler algorithm has its own set of features, and drives scheduling decisions by many factors, such as fairness, capacity guarantee, resource availability, etc. It is very important to evaluate a scheduler algorithm very well before we deploy in a production cluster. Unfortunately, currently it is non-trivial to evaluate a scheduler algorithm. Evaluating in a real cluster is always time and cost consuming, and it is also very hard to find a large-enough cluster. Hence, a simulator which can predict how well a scheduler algorithm for some specific workload would be quite useful. Source: https://hadoop.apache.org/docs/r2.9.2/hadoop-sls/SchedulerLoadSimulator.html. Example: WordCount v1.0 Before we jump into the details, lets walk through an example MapReduce application to get a flavour for how they work. WordCount is a simple application that counts the number of occurrences of each word in a given input set. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim 1	Example Southwest Count 6 Systems and Services
	 Walk-through The WordCount application is quite straight-forward. <pre> public void map(Object key, Text value, Context context) throws IOException, InterruptedException { StringTokenizer itr = new StringTokenizer(value.toString()); while (itr.hasMoreTokens()) { word.set(itr.nextToken()); context.write(word, one); } } </pre> The Mapper implementation, via the map method, processes one line at a time, as provided by the specified TextInputFormat. It then splits the line into tokens separated by whitespaces, via the StringTokenizer, and emits a key-value pair of < <word>, 1>. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim 1	Example Southwest Count 6 Systems and Services
	For the given sample input the first map emits: <pre> < Hello, 1> < World, 1> < Bye, 1> < World, 1> </pre> The second map emits: <pre> < Hello, 1> < Hadoop, 1> < Goodbye, 1> < Hadoop, 1> </pre> Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim 1	Example Southwest Count 6 Systems and Services
	WordCount also specifies a combiner. Hence, the output of each map is passed through the local combiner (which is same as the Reducer as per the job configuration) for local aggregation, after being sorted on the <i>keys</i>. The output of the first map: <pre>< Bye, 1> < Hello, 1> < World, 2></pre> The output of the second map: <pre>< Goodbye, 1> < Hadoop, 2> < Hello, 1></pre> Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. On information and belief, MapReduce InputSplit length is measured in bytes and InputSplit has storage locations. MapReduce uses storage locations to place map tasks close InputSplit's data. Map tasks are generally processed in the order of the size of the splits so larger jobs are processed first. Sources: https://data-flair.training/blogs/hadoop-optimization/; https://data-flair.training/blogs/inputsplit-in-hadoop-mapreduce/. See also org.apache.hadoop.mapreduce.JobSubmitter.java at lines 589-606; org.apache.hadoop.mapreduce.v2.app.job.impl.JobImpl.java at lines 1544-1561.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim 1	Example Southwest Count 6 Systems and Services
(e) generating at least one output combining all of the subtask outputs and reflecting the processing of all of said subtasks.	On information and belief, the Southwest Count 6 Systems and Services practice generating at least one output combining all of the subtask outputs and reflecting the processing of all of said subtasks. On information and belief, intermediate values are grouped by the Hadoop framework and passed to a Reducer to determine the final output. 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. The Hadoop MapReduce framework spawns one map task for each InputSplit generated by the InputFormat for the job. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. All intermediate values associated with a given output key are subsequently grouped by the framework, and passed to the Reducer(s) to determine the final output. Users can control the grouping by specifying a Comparator via <code>Job.setGroupingComparatorClass(Class)</code>. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. The Mapper outputs are sorted and then partitioned per Reducer. The total number of partitions is the same as the number of reduce tasks for the job. Users can control which keys (and hence records) go to which Reducer by implementing a custom Partitioner. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. Data Compression Hadoop MapReduce provides facilities for the application-writer to specify compression for both intermediate map-outputs and the job-outputs i.e. output of the reduces. It also comes bundled with <code>CompressionCodec</code> implementation for the <code>zlib</code> compression algorithm. The <code>gzip</code>, <code>bzip2</code>, <code>snappy</code>, and <code>lz4</code> file format are also supported.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim 1	Example Southwest Count 6 Systems and Services
	Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. On information and belief, the Reducer reduces a set of intermediate values to a smaller output. Reducer Reducer reduces a set of intermediate values which share a key to a smaller set of values. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. The Reducer implementation, via the reduce method just sums up the values, which are the occurrence counts for each key (i.e. words in this example). Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim 1	Example Southwest Count 6 Systems and Services
	WordCount also specifies a combiner. Hence, the output of each map is passed through the local combiner (which is same as the Reducer as per the job configuration) for local aggregation, after being sorted on the <i>keys</i>. The output of the first map: <pre>< Bye, 1> < Hello, 1> < World, 2></pre> The output of the second map: <pre>< Goodbye, 1> < Hadoop, 2> < Hello, 1></pre> Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. Thus the output of the job is: <pre>< Bye, 1> < Goodbye, 1> < Hadoop, 2> < Hello, 2> < World, 2></pre>

U.S. Patent No. 7,257,582 (Claim 1)	
Claim 1	Example Southwest Count 6 Systems and Services
	Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html .

U.S. Patent No. 7,257,582 (Claim 2)	
Claim 2	Example Southwest Count 6 Systems and Services
2. The method defined in claim 1 wherein the automatic determination of file allocation and logical subdivision of records of said input file into said plurality of partitions in step (a) and the distribution of the description of all of said partitions in step (b) is carried out with at least one further processor in addition to the subtask processors.	On information and belief, the Southwest Count 6 Systems and Services practice the method of claim 1. See claim 1. On information and belief, the Southwest Count 6 Systems and Services further practice the automatic determination of file allocation and logical subdivision of records of said input file into said plurality of partitions in step (a) and the distribution of the description of all of said partitions in step (b) is carried out with at least one further processor in addition to the subtask processors. On information and belief, Hadoop's InputFormat describes the input-specification for a MapReduce job. The input file is logically split into multiple InputSplit instances that represent data that presents a byte oriented view of the input. A recordreader converts the byte oriented view of the data to a record oriented view and presents it to a mapper. Mappers run inside a compute node. Each input file can be subdivided into multiple partitions and carried out on additional subtask processors.

U.S. Patent No. 7,257,582 (Claim 2)	
Claim 2	Example Southwest Count 6 Systems and Services
	 Job Input InputFormat describes the input-specification for a MapReduce job. The MapReduce framework relies on the InputFormat of the job to: 1. Validate the input-specification of the job. 2. Split-up the input file(s) into logical InputSplit instances, each of which is then assigned to an individual Mapper. 3. Provide the RecordReader implementation used to glean input records from the logical InputSplit for processing by the Mapper. The default behavior of file-based InputFormat implementations, typically sub-classes of FileInputFormat, is to split the input into <i>logical</i> InputSplit instances based on the total size, in bytes, of the input files. However, the FileSystem blocksize of the input files is treated as an upper bound for input splits. A lower bound on the split size can be set via <code>mapreduce.input.fileinputformat.split.minsize</code>. Clearly, logical splits based on input-size is insufficient for many applications since record boundaries must be respected. In such cases, the application should implement a RecordReader, who is responsible for respecting record-boundaries and presents a record-oriented view of the logical InputSplit to the individual task. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. Typically the compute nodes and the storage nodes are the same, that is, the MapReduce framework and the Hadoop Distributed File System (see HDFS Architecture Guide) are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

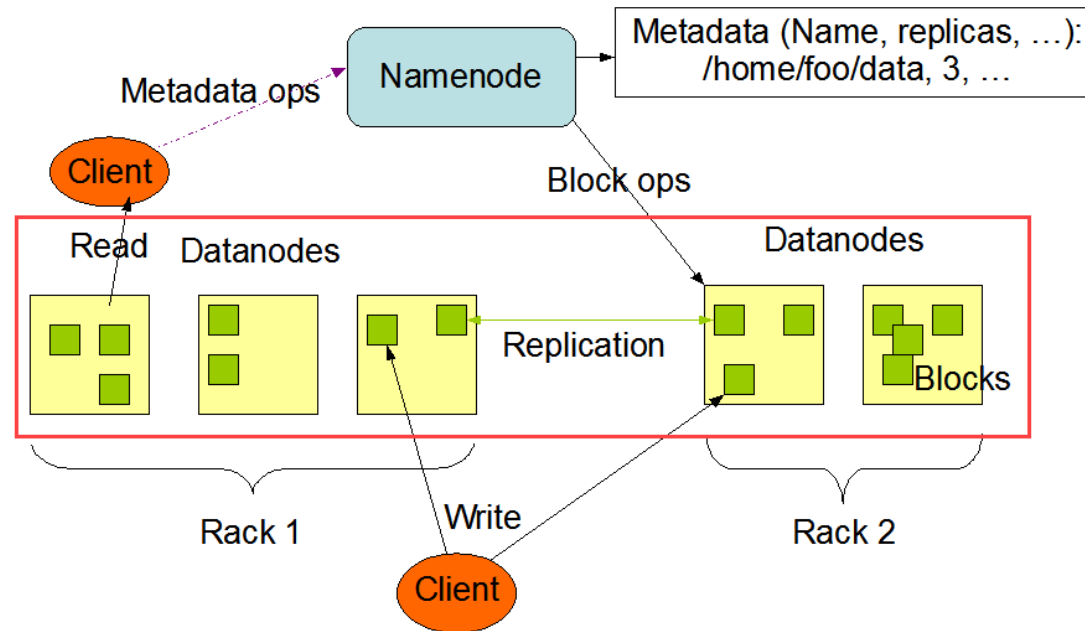
U.S. Patent No. 7,257,582 (Claim 2)	
Claim 2	Example Southwest Count 6 Systems and Services
	➡ Job Input <code>InputFormat</code> describes the input-specification for a MapReduce job. The MapReduce framework relies on the <code>InputFormat</code> of the job to: 1. Validate the input-specification of the job. 2. Split-up the input file(s) into logical <code>InputSplit</code> instances, each of which is then assigned to an individual Mapper. 3. Provide the <code>RecordReader</code> implementation used to glean input records from the logical <code>InputSplit</code> for processing by the Mapper. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. If <code>TextInputFormat</code> is the <code>InputFormat</code> for a given job, the framework detects input-files with the <code>.gz</code> extensions and automatically decompresses them using the appropriate <code>CompressionCodec</code>. However, it must be noted that compressed files with the above extensions cannot be <i>split</i> and each compressed file is processed in its entirety by a single mapper. InputSplit <code>InputSplit</code> represents the data to be processed by an individual Mapper. Typically <code>InputSplit</code> presents a byte-oriented view of the input, and it is the responsibility of <code>RecordReader</code> to process and present a record-oriented view. <code>FileSplit</code> is the default <code>InputSplit</code>. It sets <code>mapreduce.map.input.file</code> to the path of the input file for the logical split. RecordReader <code>RecordReader</code> reads <key, value> pairs from an <code>InputSplit</code>. Typically the <code>RecordReader</code> converts the byte-oriented view of the input, provided by the <code>InputSplit</code>, and presents a record-oriented to the Mapper implementations for processing. <code>RecordReader</code> thus assumes the responsibility of processing record boundaries and presents the tasks with keys and values. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

U.S. Patent No. 7,257,582 (Claim 2)

Claim 2

Example Southwest Count 6 Systems and Services

HDFS Architecture



Source: <https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html>.

U.S. Patent No. 7,257,582 (Claim 2)	
Claim 2	Example Southwest Count 6 Systems and Services
	How MapReduce works MapReduce helps chunk data processing projects into smaller pieces so they can run faster. Using one processor to analyze a huge file with terabytes or petabytes of data might, for example, take 10 hours. A MapReduce job can split that same data file into 10 tasks that run in parallel on 10 processors. This job might only take an hour or less to run. The data can be aggregated from the distributed computing nodes to produce a result. Source: https://www.ibm.com/think/topics/mapreduce.

U.S. Patent No. 7,257,582 (Claim 2)	
Claim 2	Example Southwest Count 6 Systems and Services
	3.1 Cluster Configuration MapReduce is commonly implemented on a cluster of commodity hardware, which is characterised by an assembly of cost-effective, widely available systems networked together to form a powerful collective computing resource. • Hardware Specifications: The individual nodes in a MapReduce cluster are typically equipped with components that optimise processing and storage capabilities without incurring high costs. Standard processors are used to perform computations efficiently, while adequate RAM supports rapid data access and manipulation during processing tasks. Additionally, these nodes often contain multiple hard drives, which not only provide substantial local data storage but also facilitate fast access speeds. Local storage is crucial for reducing data access times and for storing intermediate results during the MapReduce tasks, enhancing overall task execution speed. Source: https://medium.com/@zbabar/parallel-data-processing-using-mapreduce-5e23412de082.

U.S. Patent No. 7,257,582 (Claim 2)	
Claim 2	Example Southwest Count 6 Systems and Services
	5.2 Resource Allocation In MapReduce frameworks, administrators have the capability to configure the specific amount of memory and CPU resources allocated to each map and reduce tasks. This granular level of control enables optimization of resource usage tailored to the demands of specific tasks. By setting these parameters, administrators can prevent overutilization or underutilization of resources, which can lead to bottlenecks or wasted capacity, respectively. Furthermore, the MapReduce framework is designed to distribute tasks across the cluster in a manner that avoids overloading any single node while keeping other nodes from remaining idle. This balance is achieved through sophisticated scheduling algorithms that consider the current load and resource availability on each node. MapReduce ensures that all parts of the cluster contribute equally to the data processing efforts by evenly distributing tasks based on node capacity. Source: https://medium.com/@zbabar/parallel-data-processing-using-mapreduce-5e23412de082.

U.S. Patent No. 7,257,582 (Claim 5)	
Claim 5	Example Southwest Count 6 Systems and Services
5. The method defined in claim 1 wherein the output in step (e) is an accumulation of output records from said subtasks in an arbitrary order.	On information and belief, the Southwest Count 6 Systems and Services practice the method of claim 1. See claim 1. On information and belief, the Southwest Count 6 Systems and Services further practice the output in step (e) is an accumulation of output records from said subtasks in an arbitrary order. On information and belief, intermediate values are grouped by the Hadoop framework and passed to a Reducer to determine the final output, where the output is gathered based on subtasks in an arbitrary or

U.S. Patent No. 7,257,582 (Claim 5)	
Claim 5	Example Southwest Count 6 Systems and Services
	non-sequential order. 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. The Hadoop MapReduce framework spawns one map task for each InputSplit generated by the InputFormat for the job. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. All intermediate values associated with a given output key are subsequently grouped by the framework, and passed to the Reducer(s) to determine the final output. Users can control the grouping by specifying a Comparator via <code>Job.setGroupingComparatorClass(Class)</code>. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. The Mapper outputs are sorted and then partitioned per Reducer. The total number of partitions is the same as the number of reduce tasks for the job. Users can control which keys (and hence records) go to which Reducer by implementing a custom Partitioner. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. Data Compression Hadoop MapReduce provides facilities for the application-writer to specify compression for both intermediate map-outputs and the job-outputs i.e. output of the reduces. It also comes bundled with <code>CompressionCodec</code> implementation for the <code>zlib</code> compression algorithm. The <code>gzip</code>, <code>bzip2</code>, <code>snappy</code>, and <code>lz4</code> file format are also supported. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

U.S. Patent No. 7,257,582 (Claim 5)	
Claim 5	Example Southwest Count 6 Systems and Services
	On information and belief, the Reducer reduces a set of intermediate values to a smaller output. Reducer Reducer reduces a set of intermediate values which share a key to a smaller set of values. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. The Reducer implementation, via the reduce method just sums up the values, which are the occurrence counts for each key (i.e. words in this example). Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. WordCount also specifies a combiner. Hence, the output of each map is passed through the local combiner (which is same as the Reducer as per the job configuration) for local aggregation, after being sorted on the keys. The output of the first map: <pre>< Bye, 1> < Hello, 1> < World, 2></pre> The output of the second map: <pre>< Goodbye, 1> < Hadoop, 2> < Hello, 1></pre>

U.S. Patent No. 7,257,582 (Claim 5)	
Claim 5	Example Southwest Count 6 Systems and Services
	Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. Thus the output of the job is: <pre>< Bye, 1> < Goodbye, 1> < Hadoop, 2> < Hello, 2> < World, 2></pre> Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

U.S. Patent No. 7,257,582 (Claim 7)	
Claim 7	Example Southwest Count 6 Systems and Services
7. The method defined in claim 1 wherein said input file resides on a network-attached storage and is derived therefrom.	On information and belief, the Southwest Count 6 Systems and Services practice the method of claim 1. See claim 1. On information and belief, the Southwest Count 6 Systems and Services further practice the input file resides on a network-attached storage and is derived therefrom. On information and belief, Hadoop includes functionality for receiving input input-specification for a MapReduce job. The input file is logically split into multiple InputSplit instances.

U.S. Patent No. 7,257,582 (Claim 7)	
Claim 7	Example Southwest Count 6 Systems and Services
	 Job Input InputFormat describes the input-specification for a MapReduce job. The MapReduce framework relies on the InputFormat of the job to: 1. Validate the input-specification of the job. 2. Split-up the input file(s) into logical InputSplit instances, each of which is then assigned to an individual Mapper. 3. Provide the RecordReader implementation used to glean input records from the logical InputSplit for processing by the Mapper. The default behavior of file-based InputFormat implementations, typically sub-classes of FileInputFormat, is to split the input into <i>logical</i> InputSplit instances based on the total size, in bytes, of the input files. However, the FileSystem blocksize of the input files is treated as an upper bound for input splits. A lower bound on the split size can be set via <code>mapreduce.input.fileinputformat.split.minsize</code>. Clearly, logical splits based on input-size is insufficient for many applications since record boundaries must be respected. In such cases, the application should implement a RecordReader, who is responsible for respecting record-boundaries and presents a record-oriented view of the logical InputSplit to the individual task. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. On information and belief, the source of input to Hadoop can exist within network-storage, particularly for cloud uses as implemented by Southwest, including for streaming use cases. Class <code>FSDataInputStream</code> extends <code>DataInputStream</code> The core behavior of <code>FSDataInputStream</code> is defined by <code>java.io.DataInputStream</code>, with extensions that add key assumptions to the system. 1. The source is a local or remote filesystem. 2. The stream being read references a finite array of bytes. 3. The length of the data does not change during the read process. 4. The contents of the data does not change during the process. 5. The source file remains present during the read process. 6. Callers may use <code>Seekable.seek()</code> to offsets within the array of bytes, with future reads starting at this offset. 7. The cost of forward and backward seeks is low. 8. There is no requirement for the stream implementation to be thread-safe. 9. BUT, if a stream implements <code>PositionedReadable</code>, "positioned reads" MUST be thread-safe. Source: https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html.

U.S. Patent No. 7,257,582 (Claim 7)	
Claim 7	Example Southwest Count 6 Systems and Services
	How Streaming Works In the above example, both the mapper and the reducer are executables that read the input from stdin (line by line) and emit the output to stdout. The utility will create a Map/Reduce job, submit the job to an appropriate cluster, and monitor the progress of the job until it completes. Source: https://hadoop.apache.org/docs/r1.2.1/streaming.html.

**Exhibit 7 to
Intellectual Ventures I LLC's and Intellectual Ventures II LLC's
Preliminary Infringement Contentions**

**Infringement Claim Chart of
U.S. Patent No. 7,257,582 (“the ’582 Patent”)**

The Accused Systems and Services include without limitation Southwest systems and services that utilize Spark; all past, current, and future systems and services that operate in the same or substantially similar manner as the specifically identified systems and services; and all past, current, and future Southwest systems and services that have the same or substantially similar features as the specifically identified systems and services (“Example Southwest Count 6 Systems and Services” or “Southwest Systems and Services”).¹ On information and belief, Southwest’s Systems and Services directly infringe the asserted claims of the ’582 Patent, either literally or under the doctrine of equivalents, through at least using Spark in its various systems. In addition, Southwest directly infringes the ’582 Patent by testing the Southwest Systems and Services and its various systems that use Spark. On information and belief, Southwest, with knowledge at least since the time of filing the complaint, also indirectly infringes the ’582 Patent by inducing its employees, customers, and other third parties (such as vendors) to use Southwest Systems and Services, and provides documents that include instructions regarding how to use Southwest products and services that use Spark in the Southwest Systems and Services in an infringing manner. On information and belief, Southwest supplies its employees, customers, and other third parties with access to the Southwest Systems and Services through various products and services that allows its employees, customers, and other third parties to use the claimed method. On information and belief, Southwest provides instructions on how to use its various products and services that use Spark in the Southwest Systems and Services in an infringing manner, the use of which results in infringement of the ’582 Patent claims through performance of the claimed methods.

Consistent with this Court’s Standing Order Governing Proceedings (OGP), IV identifies Spark as used in Southwest’s systems and offerings as the Accused Instrumentality. Information regarding how specifically Southwest implements Spark, including for which products and offerings, is non-public. IV intends to obtain discovery regarding how the Spark is used in Southwest products and services and will update its infringement positions once discovery is completed.

IV does not intend this exemplary claim chart to be limiting, and IV reserves its rights to pursue other accused instrumentalities, patent claims, evidence, and infringement arguments in this case. Discovery has yet to begin, and this case is still in its initial stages. Accordingly, IV reserves the right to amend and/or supplement these contentions to the full extent allowed by the Court, including but not limited to, incorporating additional information, claims, theories, and / or accused products.

¹ For the avoidance of doubt, Plaintiffs do not accuse public clouds of Southwest if those services are provided by a cloud provider with a license to Plaintiffs’ patents that covers Southwest’s activities. IV will provide relevant license agreements for cloud providers in discovery. To the extent any of these licenses are relevant to Southwest’s activities, Plaintiffs will meet and confer with Southwest about the impact of such license(s).

On information and belief, the Southwest Systems and Services use Spark in Southwest's private cloud(s). For example, Southwest posts, or has posted, job opportunities that require familiarity with Spark technology.

- See <https://www.linkedin.com/in/meghanaa-malleboina/>, job profile of data engineer, stating that they utilized Pyspark for distributed data processing, leveraging RDD and DataFrame APIs for large-scale data transformations and implementing window functions for complex aggregations. (last accessed 10/8/24).
- See <https://www.linkedin.com/in/sudhir-saxena-45b9a524/>, job profile of senior data engineer stating that they used PySpark to create source to target mapping to design Data pipeline. (last accessed 10/8/24).
- See <https://www.linkedin.com/in/girishkumar81/>, job profile of senior Azure data engineer who uses Apache Spark for his position at Southwest. (last accessed 10/8/24).
- See <https://www.linkedin.com/in/amaranath-repati-a120bb22b/>, job profile of data engineer stating the use of Spark. (last accessed 10/8/24).
- See <https://www.linkedin.com/in/shreeja-srinivas-444961179/>, job profile of data engineer stating the use of Spark. (last accessed 10/8/24).

As another example, Southwest has stated that it is investing in cloud technology and has “moved about 50% of its technology” to the cloud and has indicated cloud migration is one of its areas of focus for 2024 and beyond. Source: <https://www.phocuswire.com/southwest-airlines-cio-tech-investment>.

On information and belief, other information confirms Southwest uses Spark technology.



A friend of mine recently had a data engineering interview at Southwest Airlines. The interview focused on real-world scenarios, especially working with large datasets. One question he faced was particularly challenging and required a good understanding of flight data analysis.

Source: <https://medium.com/towards-data-engineering/inside-a-southwest-airlines-data-engineering-interview-2f33ed964b5c>.²

² All sources cited in this document were publicly accessible as of the filing date of the Complaint.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	Example Southwest Count 6 Systems and Services
1. A method of effecting on a preexisting input file a computer-executable process comprised of a plurality of subtasks, the method comprising the steps of:	To the extent this preamble is limiting, on information and belief, the Southwest Count 6 Systems and Services practice a method of effecting on a preexisting input file a computer-executable process comprised of a plurality of subtasks. Spark includes a Cluster Mode where it connects to multiple cluster managers that allocate resources across applications. For example, Spark uses executors on nodes in the clusters to run tasks corresponding to an input file/data set that has been transformed into a resilient distributed dataset (RDD). Components Spark applications run as independent sets of processes on a cluster, coordinated by the SparkContext object in your main program (called the <i>driver program</i>). Specifically, to run on a cluster, the SparkContext can connect to several types of <i>cluster managers</i> (either Spark's own standalone cluster manager, Mesos, YARN or Kubernetes), which allocate resources across applications. Once connected, Spark acquires <i>executors</i> on nodes in the cluster, which are processes that run computations and store data for your application. Next, it sends your application code (defined by JAR or Python files passed to SparkContext) to the executors. Finally, SparkContext sends <i>tasks</i> to the executors to run. Source: https://spark.apache.org/docs/latest/cluster-overview.html.³

³ Annotations added unless otherwise noted.

U.S. Patent No. 7,257,582 (Claim 1)

Claim(s)	Example Southwest Count 6 Systems and Services
	<pre> graph LR subgraph DP [Driver Program] SC[SparkContext] end CM[Cluster Manager] subgraph WN1 [Worker Node] E1[Executor] C1[Cache] T1[Task] end subgraph WN2 [Worker Node] E2[Executor] C2[Cache] T2[Task] end SC <--> CM CM <--> E1 CM <--> E2 E1 <--> T1 E2 <--> T2 </pre> Source: https://spark.apache.org/docs/latest/cluster-overview.html. 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 on a cluster</i>. 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. Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html. 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: <i>parallelizing</i> 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. Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	Example Southwest Count 6 Systems and Services
(a) automatically determining file allocation and logically subdividing records of said input file into a plurality of partitions;	On information and belief, the Southwest Count 6 Systems and Services practice automatically determining file allocation and logically subdividing records of said input file into a plurality of partitions. Spark creates an RDD from datasets including files. This process includes forming a logical division of the datasets. An RDD results in an automatic file allocation across multiple nodes in a cluster, so that the nodes of the cluster can operate on the RDD in parallel. Resilient Distributed Datasets (RDDs) Spark revolves around the concept of a resilient distributed dataset (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 referencing a dataset in an external storage system, such as a shared filesystem, HDFS, HBase, or any data source offering a Hadoop InputFormat. Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html. 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 resilient distributed dataset (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. Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html. 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. 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. 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. Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	Example Southwest Count 6 Systems and Services
(b) distributing descriptions of all of said partitions to each of a plurality of subtask processors	On information and belief, the Southwest Count 6 Systems and Services practice distributing descriptions of all of said partitions to each of a plurality of subtask processors. Spark distributes partitions to clusters which includes multiple machines or nodes for processing. For example, Spark prepares jobs to perform work on the partitions. Jobs are divided into stages, and stages are divided into tasks, with one task assigned for each partition. 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. Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html. 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. 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. 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. Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html.

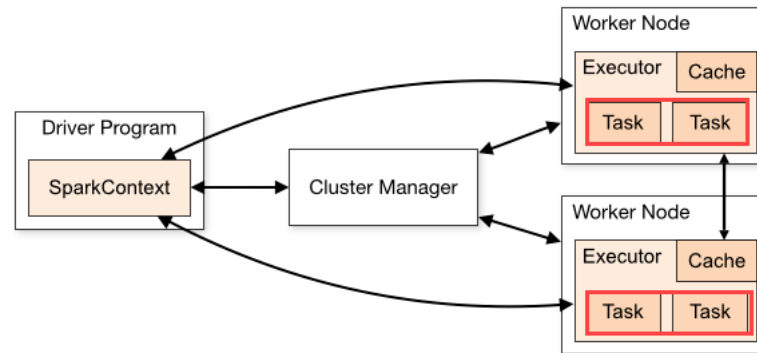
U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	Example Southwest Count 6 Systems and Services
	Apache Spark's Resilient Distributed Datasets (RDD) are a collection of various data that are so big in size, that they cannot fit into a single node and should be partitioned across various nodes. Apache Spark automatically partitions RDDs and distributes the partitions across different nodes. They are evaluated lazily (i.e, the execution will not start until an action is triggered which increases manageability, saves computation and thus increases optimization and speed) and the transformations are stored as directed acyclic graphs (DAG). So, every action on the RDD will make Apache Spark recompute the DAG. Source: https://www.talend.com/resources/intro-apache-spark-partitioning/.
(c) simultaneously executing at least a respective one of the subtasks of the computer-executable process in each of at least some of said processors on a respective one of the partitions with each subtask reading and processing the respective partition so as to process the respective partition and produce respective subtask output and;	On information and belief, the Southwest Count 6 Systems and Services practice simultaneously executing at least a respective one of the subtasks of the computer-executable process in each of at least some of said processors on a respective one of the partitions with each subtask reading and processing the respective partition so as to process the respective partition and produce respective subtask output. Spark worker nodes simultaneously execute at least one task relating to the partition. Components Spark applications run as independent sets of processes on a cluster, coordinated by the SparkContext object in your main program (called the <i>driver program</i>). Specifically, to run on a cluster, the SparkContext can connect to several types of <i>cluster managers</i> (either Spark's own standalone cluster manager, Mesos, YARN or Kubernetes), which allocate resources across applications. Once connected, Spark acquires <i>executors</i> on nodes in the cluster, which are processes that run computations and store data for your application. Next, it sends your application code (defined by JAR or Python files passed to SparkContext) to the executors. Finally, SparkContext sends tasks to the executors to run. Source: https://spark.apache.org/docs/latest/cluster-overview.html.⁴

⁴ Annotations added unless otherwise noted.

U.S. Patent No. 7,257,582 (Claim 1)

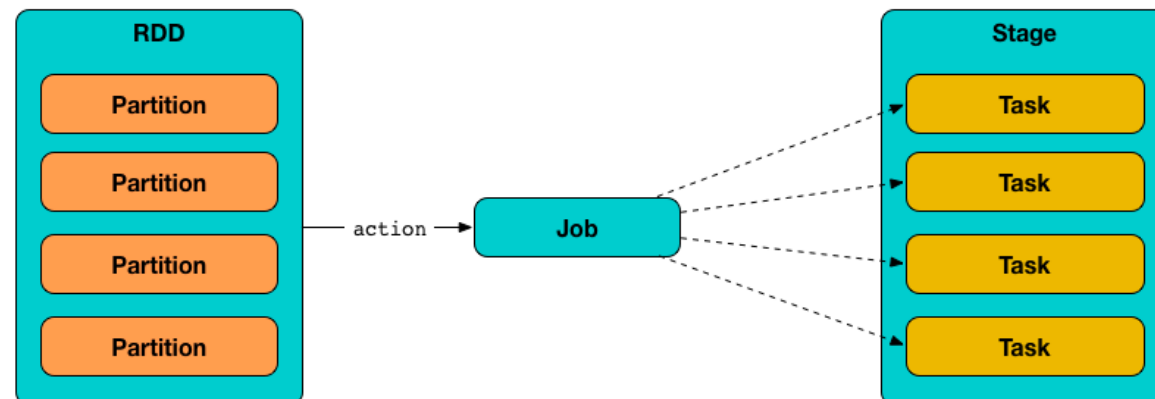
Claim(s)

Example Southwest Count 6 Systems and Services



Source: <https://spark.apache.org/docs/latest/cluster-overview.html>.

In Spark, each stage is built from transformations that can be done in a row, without the need for shuffle (narrow transformations). To recap, stages are created based on chunks of processing that can be done in a parallel manner, without shuffling things around again.



U.S. Patent No. 7,257,582 (Claim 1)

Claim(s)

Example Southwest Count 6 Systems and Services

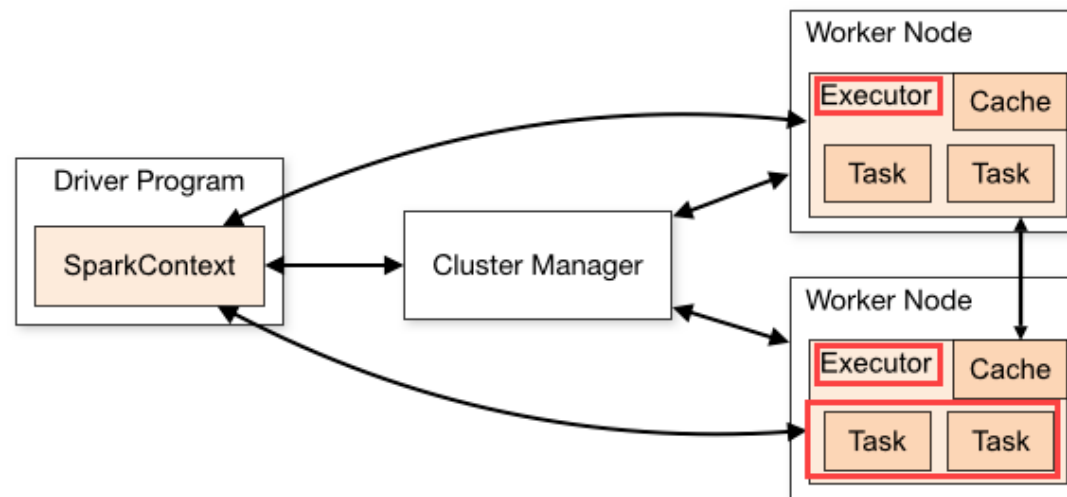
Source: <https://engineering.salesforce.com/how-to-optimize-your-apache-spark-application-with-partitions-257f2c1bb414/>.

Spark's tasks relating to each partition are being executed in parallel by the worker nodes.

There are several useful things to note about this architecture:

1. Each application gets its own executor processes, which stay up for the duration of the whole application and run tasks in multiple threads. This has the benefit of isolating applications from each other, on both the scheduling side (each driver schedules its own tasks) and executor side (tasks from different applications run in different JVMs). However, it also means that data cannot be shared across different Spark applications (instances of SparkContext) without writing it to an external storage system.

Source: <https://spark.apache.org/docs/latest/cluster-overview.html>.



U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	Example Southwest Count 6 Systems and Services
	In Spark, data is generally not distributed across partitions to be in the necessary place for a specific operation. During computations, a single task will operate on a single partition – thus, to organize all the data for a single <code>reduceByKey</code> reduce task to execute, Spark needs to perform an all-to-all operation. It must read from all partitions to find all the values for all keys, and then bring together values across partitions to compute the final result for each key – this is called the shuffle. Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html. The Shuffle is an expensive operation since it involves disk I/O, data serialization, and network I/O. To organize data for the shuffle, Spark generates sets of tasks – <i>map</i> tasks to organize the data, and a set of <i>reduce</i> tasks to aggregate it. This nomenclature comes from MapReduce and does not directly relate to Spark's map and reduce operations. Internally, results from individual map tasks are kept in memory until they can't fit. Then, these are sorted based on the target partition and written to a single file. On the reduce side, tasks read the relevant sorted blocks. Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html.
(d) thereafter repeating step (c) in at least some of the subtask processors each with another unprocessed partition on a first-come/first-served basis; and	On information and belief, the Southwest Count 6 Systems and Services practice repeating step (c) in at least some of the subtask processors each with another unprocessed partition on a first-come/first-served basis. Spark task output produced by each worker node is combined to produce processed output for the related partition. The simplest option, available on all cluster managers, is <i>static partitioning</i> of resources. With this approach, each application is given a maximum amount of resources it can use and holds onto them for its whole duration. This is the approach used in Spark's <i>standalone</i> and <i>YARN</i> modes, as well as the <i>coarse-grained Mesos mode</i>. Resource allocation can be configured as follows, based on the cluster type: • Standalone mode: By default, applications submitted to the standalone mode cluster will run in FIFO (first-in-first-out) order, and each application will try to use all available nodes. You can limit the number of nodes an application uses by setting the <code>spark.cores.max</code> configuration property in it, or change the default for applications that don't set this setting through <code>spark.deploy.defaultCores</code>. Finally, in addition to controlling cores, each application's <code>spark.executor.memory</code> setting controls its memory use. Source: https://spark.apache.org/docs/latest/job-scheduling.html.

U.S. Patent No. 7,257,582 (Claim 1)

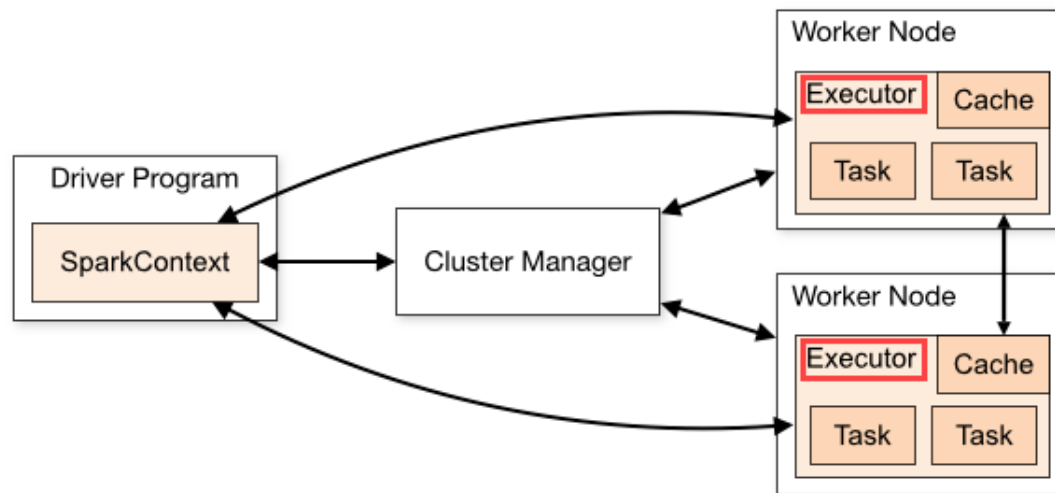
Claim(s)

Example Southwest Count 6 Systems and Services

1 Available resources in your cluster

Spark's official recommendation is that you have ~3x the number of partitions than available cores in cluster, to maximize parallelism along side with the overhead of spinning more executors. But this is not quite so straight forward. If our tasks are rather simple (taking less than a second), we might want to consider decreasing our partitions (to avoid the overhead), even if it means less than 3x. We should also take under consideration the memory of each executor, and make sure we are not exceeding it. If we do, we might want to create more partitions, even if it's more than 3x our available cores, so that each will be smaller, or increase the memory of our executors.

Source: <https://engineering.salesforce.com/how-to-optimize-your-apache-spark-application-with-partitions-257f2c1bb414/>.



Source: <https://spark.apache.org/docs/latest/cluster-overview.html>.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	Example Southwest Count 6 Systems and Services
	By default, Spark's scheduler runs jobs in FIFO fashion. Each job is divided into "stages" (e.g. map and reduce phases), and the first job gets priority on all available resources while its stages have tasks to launch, then the second job gets priority, etc. If the jobs at the head of the queue don't need to use the whole cluster, later jobs can start to run right away, but if the jobs at the head of the queue are large, then later jobs may be delayed significantly. Source: https://spark.apache.org/docs/latest/job-scheduling.html.
(e) generating at least one output combining all of the subtask outputs and reflecting the processing of all of said subtasks.	On information and belief, the Southwest Count 6 Systems and Services practice generating at least one output combining all of the subtask outputs and reflecting the processing of all of said subtasks. Spark task output produced by each worker node is combined to produce final processed output for the corresponding partition. In Spark, data is generally not distributed across partitions to be in the necessary place for a specific operation. During computations, a single task will operate on a single partition – thus, to organize all the data for a single reduceByKey reduce task to execute, Spark needs to perform an all-to-all operation. It must read from all partitions to find all the values for all keys, and then bring together values across partitions to compute the final result for each key – this is called the shuffle. Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html. The Shuffle is an expensive operation since it involves disk I/O, data serialization, and network I/O. To organize data for the shuffle, Spark generates sets of tasks – map tasks to organize the data, and a set of reduce tasks to aggregate it. This nomenclature comes from MapReduce and does not directly relate to Spark's map and reduce operations. Internally, results from individual map tasks are kept in memory until they can't fit. Then, these are sorted based on the target partition and written to a single file. On the reduce side, tasks read the relevant sorted blocks. Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html.

U.S. Patent No. 7,257,582 (Claim 2)	
Claim 2	Example Southwest Count 6 Systems and Services
2. The method defined in claim 1 wherein the automatic determination of file allocation and logical subdivision of records of said input file into said plurality of partitions in step (a) and the distribution of the description of all of said partitions in step (b) is carried out with at least one further processor in addition to the subtask processors.	On information and belief, the Southwest Count 6 Systems and Services practice the method of claim 1. See claim 1. On information and belief, the Southwest Count 6 Systems and Services further practice the automatic determination of file allocation and logical subdivision of records of said input file into said plurality of partitions in step (a) and the distribution of the description of all of said partitions in step (b) is carried out with at least one further processor in addition to the subtask processors. Spark distributes partitions to clusters which includes multiple machines or nodes for processing. For example, Spark prepares jobs to perform work on the partitions. Jobs are divided into stages, and stages are divided into tasks, with one task assigned for each partition. 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. Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html. 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. 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. 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. Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html.

U.S. Patent No. 7,257,582 (Claim 2)	
Claim 2	Example Southwest Count 6 Systems and Services
	Apache Spark's Resilient Distributed Datasets (RDD) are a collection of various data that are so big in size, that they cannot fit into a single node and should be partitioned across various nodes. Apache Spark automatically partitions RDDs and distributes the partitions across different nodes. They are evaluated lazily (i.e, the execution will not start until an action is triggered which increases manageability, saves computation and thus increases optimization and speed) and the transformations are stored as directed acyclic graphs (DAG). So, every action on the RDD will make Apache Spark recompute the DAG. Source: https://www.talend.com/resources/intro-apache-spark-partitioning/.

U.S. Patent No. 7,257,582 (Claim 5)	
Claim 5	Example Southwest Count 6 Systems and Services
5. The method defined in claim 1 wherein the output in step (e) is an accumulation of output records from said subtasks in an arbitrary order.	On information and belief, the Southwest Count 6 Systems and Services practice the method of claim 1. See claim 1. On information and belief, the Southwest Count 6 Systems and Services further practice the output in step (e) is an accumulation of output records from said subtasks in an arbitrary order. Spark task output produced by each worker node is combined to produce final processed output for the corresponding partition. In Spark, data is generally not distributed across partitions to be in the necessary place for a specific operation. During computations, a single task will operate on a single partition – thus, to organize all the data for a single reduceByKey reduce task to execute, Spark needs to perform an all-to-all operation. It must read from all partitions to find all the values for all keys, and then bring together values across partitions to compute the final result for each key – this is called the shuffle. Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html.

U.S. Patent No. 7,257,582 (Claim 5)	
Claim 5	Example Southwest Count 6 Systems and Services
	The Shuffle is an expensive operation since it involves disk I/O, data serialization, and network I/O. To organize data for the shuffle, Spark generates sets of tasks – <i>map</i> tasks to organize the data, and a set of <i>reduce</i> tasks to aggregate it. This nomenclature comes from MapReduce and does not directly relate to Spark's map and reduce operations. Internally, results from individual map tasks are kept in memory until they can't fit. Then, these are sorted based on the target partition and written to a single file. On the reduce side, tasks read the relevant sorted blocks. Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html.

U.S. Patent No. 7,257,582 (Claim 7)	
Claim 7	Example Southwest Count 6 Systems and Services
7. The method defined in claim 1 wherein said input file resides on a network-attached storage and is derived therefrom.	On information and belief, the Southwest Count 6 Systems and Services practice the method of claim 1. See claim 1. On information and belief, the Southwest Count 6 Systems and Services further practice the input file resides on a network-attached storage and is derived therefrom. On information and belief, Spark includes functionality for receiving input file/data sets that are then transformed into a resilient distributed dataset (RDD). These input file/data sets involve network storage, for example files stored in network memory or streaming applications involving input stored on a network storage. Components Spark applications run as independent sets of processes on a cluster, coordinated by the SparkContext object in your main program (called the <i>driver program</i>). Specifically, to run on a cluster, the SparkContext can connect to several types of <i>cluster managers</i> (either Spark's own standalone cluster manager, Mesos, YARN or Kubernetes), which allocate resources across applications. Once connected, Spark acquires <i>executors</i> on nodes in the cluster, which are processes that run computations and store data for your application. Next, it sends your application code (defined by JAR or Python files passed to SparkContext) to the executors. Finally, SparkContext sends tasks to the executors to run.

U.S. Patent No. 7,257,582 (Claim 7)	
Claim 7	Example Southwest Count 6 Systems and Services
	Source: https://spark.apache.org/docs/latest/cluster-overview.html.⁵ The diagram illustrates the Spark architecture. On the left, a box labeled 'Driver Program' contains 'SparkContext'. In the center is a 'Cluster Manager'. On the right are two 'Worker Node' boxes. Each Worker Node contains an 'Executor' and a 'Cache'. Inside the 'Executor' of each Worker Node are two 'Task' boxes, which are highlighted with red borders. Arrows show the flow of data and control: from 'SparkContext' to 'Cluster Manager', from 'Cluster Manager' to each 'Worker Node', and from each 'Task' back to 'SparkContext'. Source: https://spark.apache.org/docs/latest/cluster-overview.html. 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 on a cluster</i>. 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. Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html.

⁵ Annotations added unless otherwise noted.

U.S. Patent No. 7,257,582 (Claim 7)	
Claim 7	Example Southwest Count 6 Systems and Services
	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: <i>parallelizing</i> 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. Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html. Overview Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams. Data can be ingested from many sources like Kafka, Kinesis, or TCP sockets, and can be processed using complex algorithms expressed with high-level functions like map, reduce, join and window. Finally, processed data can be pushed out to filesystems, databases, and live dashboards. In fact, you can apply Spark's machine learning and graph processing algorithms on data streams.  <pre> graph LR subgraph Sources direction TB K[Kafka] F[Flume] H[HDFS/S3] Ki[Kinesis] T[Twitter] end Sources --> SS[Spark Streaming] SS --> Destinations subgraph Destinations direction TB HD[HDFS] DB[Databases] DA[Dashboards] end </pre> Source: https://spark.apache.org/docs/latest/streaming-programming-guide.html#overview.