
IN THE
UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE PATENT TRIAL AND APPEAL BOARD

DATABRICKS, INC.,

Petitioner

- vs. -

R2 SOLUTIONS LLC

Patent Owner

Patent No. 8,190,610
Issued: May 29, 2012
Inventors: Ali Dasdan, Hung-Chih Yang, Ruey-Lung Hsiao
Title: MapReduce for distributed database processing

**PETITION FOR *INTER PARTES* REVIEW
OF U.S. PATENT NO. 8,190,610**

Mail Stop Patent Board
Patent Trial and Appeal Board
P.O. Box 1450
Alexandria, VA 22313-1450

TABLE OF CONTENTS

	Page
I. Introduction.....	1
II. Requested Relief.....	1
III. Overview of the Subject Matter.....	2
A. Overview of MapReduce	2
B. Overview of the '610 Patent.....	5
C. Prosecution History	9
D. The Level of Ordinary Skill in the Art.....	9
E. Claim Construction	9
IV. Overview of the Prior Art.....	10
A. Pike	10
B. Chowdhuri	12
V. Grounds 1 and 2.....	15
A. Claim 1	15
[1a] “A method of processing data of a data set over a distributed system,”.....	15
[1b] “wherein the data set comprises a plurality of data groups,”	17
[1c] “the method comprising: partitioning...”	22
[1d] “providing each data partition...”	25
[1e] “to independently output a plurality of lists of values...,”.....	30
[1f] “wherein the data of a first data group has a different schema than the data of a second data group”	38
[1g] “and the data of the first data group is mapped differently...,”	44

Petition for *Inter Partes* Review of
U.S. Patent No. 8,190,610

[1h]	“wherein the different schema and corresponding different intermediate data have a key in common; and”	48
[1i]	“reducing...,”	49
[1j]	“wherein the mapping and reducing operations are performed by a distributed system”	52
B.	Claim 17	53
[17a]	“A computer system...”	53
[17b]	“partitioning...”	53
[17c]	“providing...”	53
[17d]	“to independently output...,”	53
[17e]	“wherein the data...”	53
[17f]	“and the data of the first data group...,”	53
[17g]	“wherein the different schema...”	54
[17h]	“reduce...”	54
C.	Claims 2, 18	54
D.	Claims 3, 19	55
E.	Claims, 4, 20	59
F.	Claims 5, 21	59
G.	Claims 6, 23	60
[a]	“wherein:...;”	60
[b]	“partition[ing]...”	61
H.	Claims 7, 24	62
[a]	“the intermediate data...; the reducing...;”	62
[b]	“partition[ing]...; and provid[ing]...”	62
I.	Claims 8, 25	64

Petition for *Inter Partes* Review of
U.S. Patent No. 8,190,610

J.	Claims 9, 26.....	65
	[a] “...at least some...;”	65
	[b] “...generating and providing...”	66
K.	Claims 10, 27 – “...processing the metadata.”	67
L.	Claims 11, 28 – “processing...”	68
M.	Claims 12, 22.....	68
	[a] “the intermediate data includes...” and “the reducing step...;”	68
	[b] “for at least one of the reducers, ...”	68
N.	Claims 13, 29.....	70
O.	Claims 14, 30.....	71
P.	Claims 15, 31.....	72
Q.	Claims 16, 32.....	72
R.	Claim 33	73
	[33a] “A map-reduce...”	73
	[33b] “for a first data set...”	73
	[33c] “partitioning...”	74
	[33d] “providing...”	74
	[33e] “to independently output...”	74
	[33f] “for a second data set...”	74
	[33g] “partitioning...”	74
	[33h] “providing...”	75
	[33i] “wherein the first schema differs from the second schema,”	77
	[33j] “wherein the first and second schema...;”	77

Petition for *Inter Partes* Review of
U.S. Patent No. 8,190,610

	[33k] “reducing...”	77
	[33l] “the output data set...”	78
	[33m] “wherein...”	79
S.	Claim 40	79
	[40a] “A computer system...”	79
	[40b]-[40l]	79
T.	Claims 34, 41	79
U.	Claims 35, 42	80
V.	Claims 36, 43	80
W.	Claims 37, 44	80
X.	Claims 38, 45	80
Y.	Claims 39, 46	81
VI.	Mandatory Notices Under 37 C.F.R. §42.8	81
A.	Real Party-in-Interest	81
B.	Related Matters	81
C.	Lead and Backup Counsel	84
D.	Service Information	84
VII.	Grounds for Standing	85
VIII.	Conclusion	85

TABLE OF EXHIBITS

Exhibit	Description
EX-1001	U.S. Patent No. 8,190,610 to Dasdan (“the ’610 patent”)
EX-1002	U.S. Patent No. 7,590,620 to Pike
EX-1003	U.S. Patent Publication No. 2006/0218123 to Chowdhuri
EX-1004	File History of U.S. Patent No. 8,190,610
EX-1005	Declaration of Jimmy Lin, Ph.D. from IPR2024-00303 (“Lin Declaration”)
EX-1006	Curriculum Vitae of Jimmy Lin, Ph.D.
EX-1007	Claim Construction Memorandum Opinion and Order, <i>R2 Solutions LLC v. Deezer S.A.</i> , 4:21-cv-00090, Dkt. 45 (E.D. Tex. Jan. 4, 2022)
EX-1008	Claim Construction Memorandum Opinion and Order, <i>R2 Solutions LLC v. American Airlines, Inc.</i> , 4:22-cv-00353, Dkt. 58 (E.D. Tex. Apr. 3, 2023)
EX-1009	Jeffrey Dean et al., MapReduce: Simplified Data Processing on Large Clusters, Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December 2004
EX-1010	[Reserved]
EX-1011	Jeffrey Dean et al., MapReduce: Simplified Data Processing on Large Clusters, available at https://research.google.com/archive/mapreduce-osdi04-slides/index.html
EX-1012	[Reserved]
EX-1013	[Reserved]
EX-1014	[Reserved]
EX-1015	[Reserved]

Petition for *Inter Partes* Review of
U.S. Patent No. 8,190,610

EX-1016	Kimberly Tripp, “SQL Server 2005: Partitioned Tables and Indexes,” SQLSkills.com (Sep. 24, 2004), available at https://www.sqlskills.com/resources/whitepapers/partitioning%20in%20sql%20server%202005%20beta%20ii.htm)
EX-1017	U.S. Patent No. 5,797,000 to Bhattacharya
EX-1018	U.S. Patent No. 7,020,661 to Cruanes
EX-1019	[Reserved]
EX-1020	Oracle Database SQL Reference 10g
EX-1021	“Join (SQL),” Wikipedia.org (Aug. 11, 2004)
EX-1022	Hung-chih Yang, et al., <i>Map-Reduce-Merge: Simplified Relational Data Processing on Large Clusters</i> , SIGMOD ’07: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data (2007)
EX-1023	[Reserved]
EX-1024	U.S. Patent No. 6,343,295 to MacLeod
EX-1025	U.S. Patent Publication No. 2005/0165865 to Farmer
EX-1026	Listing of Claims of U.S. Patent No. 8,190,610
EX-1027	Declaration of Jimmy Lin, Ph.D.

I. INTRODUCTION

Petitioner Databricks, Inc. (“Petitioner” or “Databricks”) requests *inter partes* review of claims 1-46 of U.S. Patent No. 7,590,610. This Petition raises grounds of unpatentability identical to those already raised in IPR2024-00303. It neither adds to nor alters any arguments the Board need consider and is accompanied by a timely motion for joinder under 35 U.S.C. § 315(c). The Board should institute review and cancel claims 1-46.

II. REQUESTED RELIEF

Petitioner requests review under 35 U.S.C. § 311 of the ’610 patent and cancellation of claims 1-46 as unpatentable in view of:

Prior Art
Pike (EX-1002); filed September 29, 2004; pre-AIA 35 U.S.C. § 102(e) prior art
Chowdhuri (EX-1003); filed June 2, 2005, published September 25, 2006; pre-AIA 35 U.S.C. §§ 102(a), (e) prior art
MacLeod (EX-1024); issued Jan. 29, 2002; pre-AIA 35 U.S.C. § 102(b) prior art

Unpatentability Grounds	
1	Pike renders obvious claims 1-46 under 35 U.S.C. § 103
2	Pike in view of Chowdhuri (“Pike-Chowdhuri”) renders obvious claims 1-46 under 35 U.S.C. § 103
3	Pike and Chowdhuri in view of MacLeod (“Pike-Chowdhuri-MacLeod”) renders obvious claims 3, 4, 9-11, 19, 26, 27, 37-39, 44-46 under 35 U.S.C. § 103

III. OVERVIEW OF THE SUBJECT MATTER

A. Overview of MapReduce

Well before the '610 patent, Google's Jeffrey Dean and Sanjay Ghemawat shared their programming methodology called “MapReduce” for processing “parallel computations over distributed (typically, very large) data sets.” '610 patent (EX-1001), 1:6-27; EX-1009; EX-1005 ¶¶35-36.¹ MapReduce maps a set of data into key-value pairs and reduces those pairs to a list of values. EX-1001, 1:6-27, Figs. 1-2.

¹ Submitted herewith is the Declaration of Jimmy Lin, Ph.D. adopting his declaration filed as Exhibit 1005 in IPR2024-00303. EX-1027.

Figures 1 and 2 of the '610 patent purportedly describe a “conventional” MapReduce architecture and implementation. *Id.*, 1:48-51; EX-1005 ¶37.

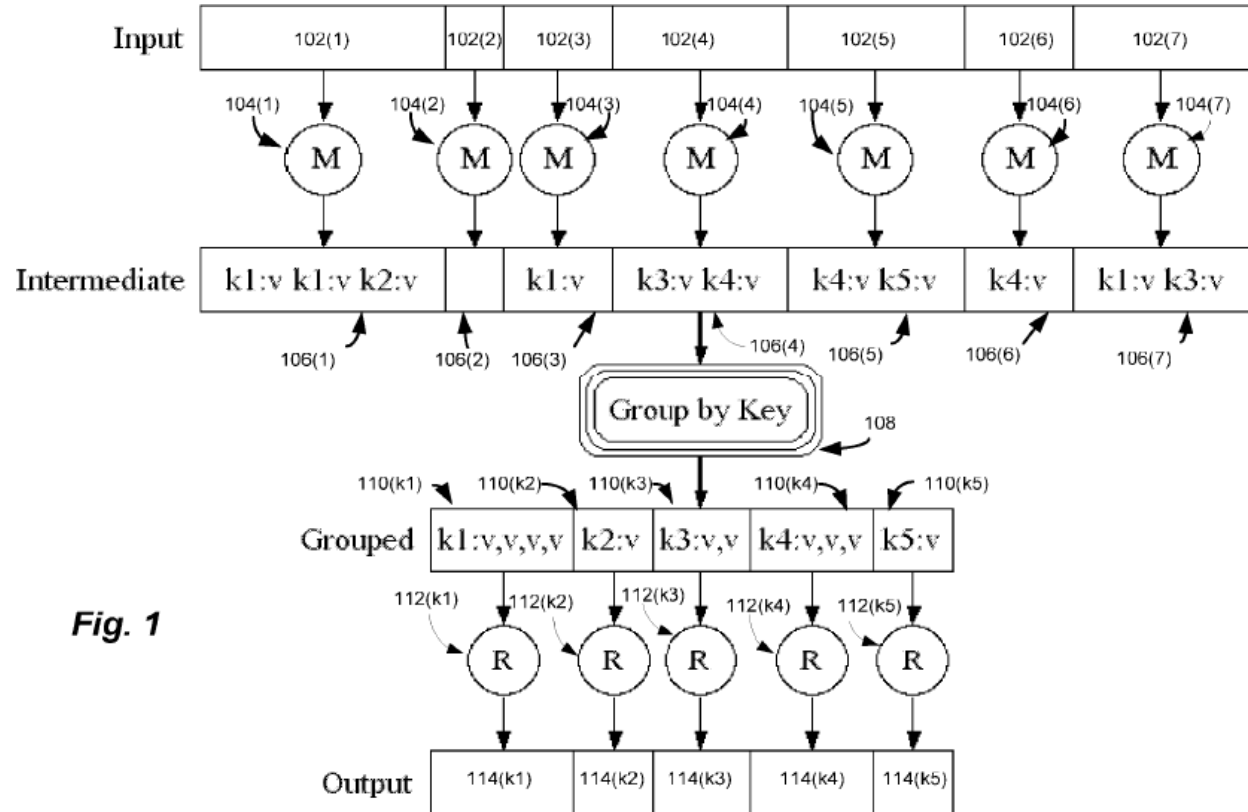
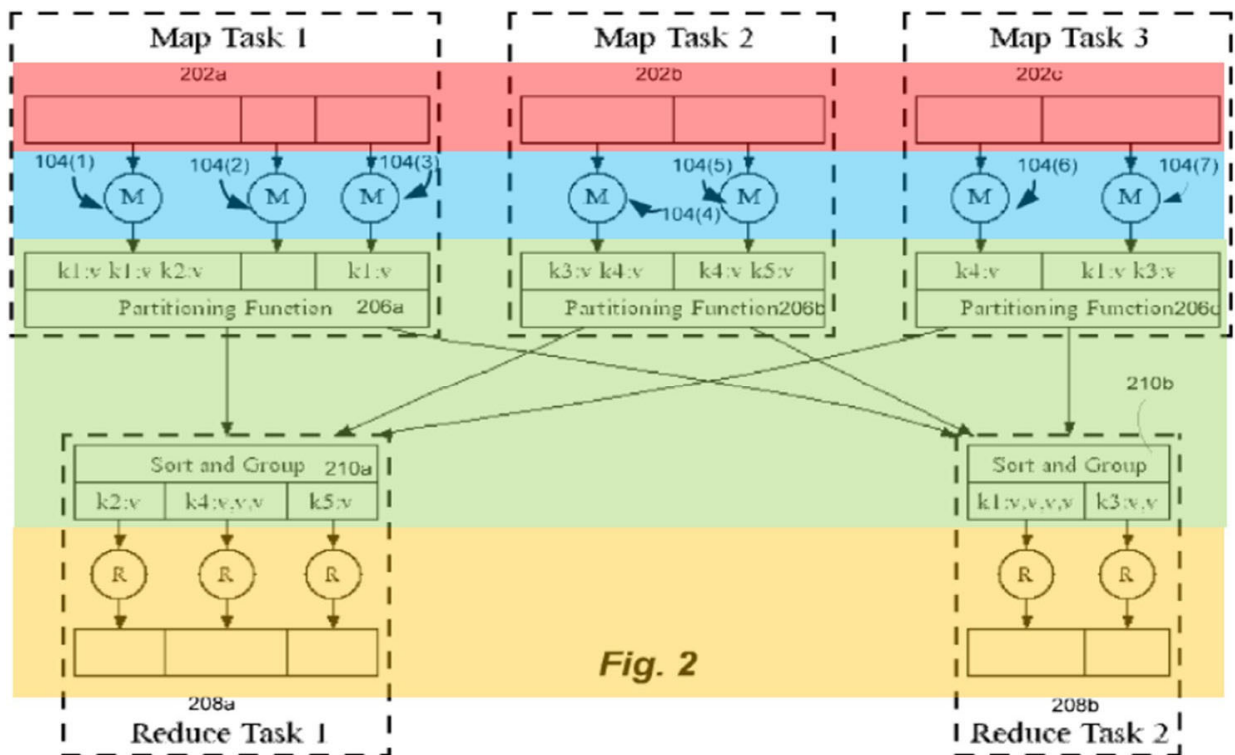


Fig. 1

EX-1001, Fig. 1. Input data 102 comprises keys k1-k5 and is divided into partitions 102(1)-102(7). *Id.*, 2:9-51. Each partition “is” provided to [a] corresponding invocation of [a] map function,” 104(1)-104(7), to generate intermediate data 106(1)-106(7). *Id.* The intermediate data is then provided to corresponding reduce functions 112(k1)-112(k5). *Id.* The reduce functions generate corresponding output data 114, sorted by keys 114(k1)-114(k5). *Id.* The map and reduce functions are

application-specific, dependent on use case, and “user-provided.” *Id.*, 1:17-27, 2:9-51; EX-1005 ¶¶38-39.

Figure 2 provides a parallel implementation of Figure 1, separated into three map tasks, 202a-202c, and two reduce tasks, 208a, 208b. EX-1001, 2:52-67. Figure 2 is annotated to illustrate the **partitioned input data**, the **mapping tasks**, the **intermediate data sorted by key-value pair**, and the **reducing tasks with output data**:



Id., Fig. 2 (annotated); EX-1005 ¶40.

Dean and Ghemawat’s publication on MapReduce provides real-world uses of MapReduce. In one example, MapReduce “count[s] the number of occurrences of each word in a large collection of documents.” EX-1009, 2. The map function

“emits each word plus an associated count of occurrences,” while the reduce function “sums together all counts emitted for a particular word.” *Id.* While that example concerned a generic “collection of documents,” other examples included a different or specific type of input, namely, webpages: listing webpages that link to a target webpage (i.e., a “Reverse Web-Link Graph”), summarizing the most important words from different webpages (i.e., a “Term-Vector per Host”), and identifying webpages that include a specified word (i.e., an “Inverted Index”). *Id.*, 2-3; EX-1005 ¶¶41-42.

One of Google’s “most significant uses of MapReduce” by 2004 was indexing webpages for Google’s web search service. EX-1009, 10-11. The indexing process used “as input a large set of documents that have been retrieved by [Google’s] crawling system, ... [t]he raw contents for these documents are more than 20 terabytes of data.” *Id.* These examples demonstrate that MapReduce could process diverse data sets and analyze large volumes of input data in an efficient manner. EX-1005 ¶¶43-45. Indeed, Dean and Ghemawat’s MapReduce publication placed no limitations on MapReduce’s input data or its schema. *Id.*; EX-1009, 2, 10-11.

B. Overview of the ’610 Patent

The ’610 patent purports to disclose “enhanced” MapReduce. EX-1001, Abstract, 1:31-44. The purported enhancement, however, merely characterizes the input data to MapReduce: “by treating an input data set as a plurality of grouped sets

of key/value pairs, the utility of the MapReduce programming methodology may be enhanced.” *Id.*, 1:66-2:8, 1:31-41, 2:9-10, 3:16-18, 3:35-41, Abstract. According to the patent, “data sets within the same group are characterized by the same schema; and data sets within different groups are characterized by different schemas.” *Id.*, 3:48-57. The ’610 patent thus specifies the data in each partition 102 of MapReduce shown in Figure 1. This was not new. EX-1005 ¶56; EX-1009, 2, 10-11.

When discussing whether MapReduce provided the ability to map and reduce data groups of different schema together, the ’610 patent only alleges that MapReduce could not “*efficiently* process data from heterogeneous sources,” EX-1001, 3:9-20,² and that it was “impractical,” *id.*; EX-1005 ¶57. The specification concedes that its “related ... datasets” are just like those of conventional MapReduce: “characterized by a common primary key.” *Compare* EX-1001, Abstract, 1:34-36, 2:2-4, 8:47-52, *with id.* Fig. 1, 2:12-18, 2:49-51; EX-1005 ¶58.

Figures 4 and 5 depict the purported enhancement applied to an Employee table 302 and a Department table 304. EX-1001, 3:65-4:22, 8:15-46, Figs. 4, 5. But these figures are indistinguishable from “conventional” MapReduce in Figures 1 and 2, as illustrated by the annotations below, showing the same **partitioned input data**, **mapping tasks**, **intermediate data sorted by key-value pair**, and **reducing tasks**

² Emphases added unless otherwise indicated.

with output data. Grouping input data and describing each group as having a different schema (organization) creates no meaningful distinction. EX-1005 ¶59.

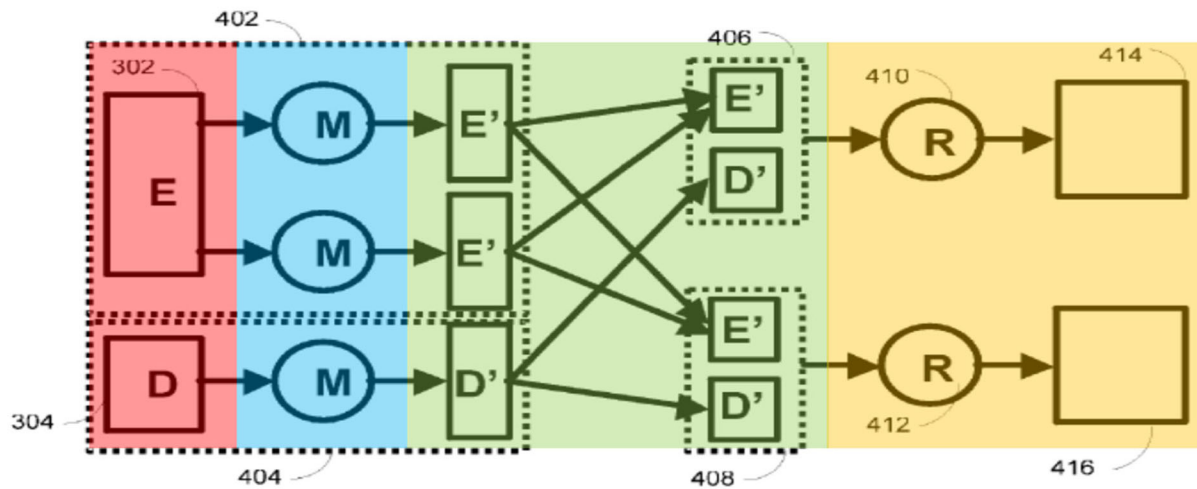
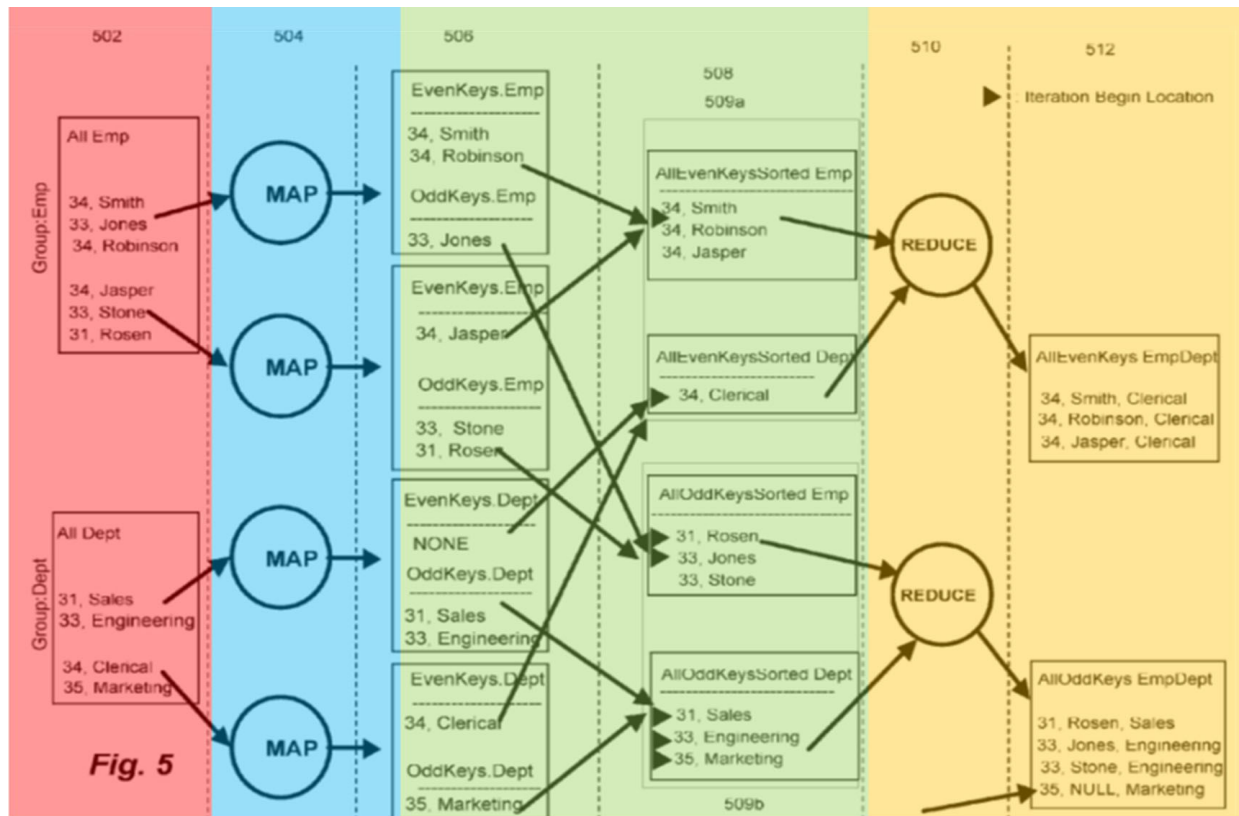


Fig. 4

Id., Fig. 4 (annotated).



Id., Fig. 5 (annotated).

The named inventors acknowledged that MapReduce could process data “from multiple sources.” EX-1022, 1029-30. They noted that developers would only have to encode “an additional data-source attribute.” *Id.* Using these identifiers, “[u]ser-defined logic can extract data-sources from values to identify their origins, then the entries from different sources can be merged.” *Id.*, 1032. But encoding data tags or identifiers to identify the source of data was a known MapReduce capability, and developers could easily encode input data this way before the ’610 patent. EX-1005 ¶60. It was also known in the art. *Id.* ¶¶61-63; EX-1024, Abstract, 5:30-42, 5:53-60, Fig. 6; EX-1025, Abstract, ¶¶5, 8, 26-27.

C. Prosecution History

The Office repeatedly rejected the application that would issue as the '610 patent. EX-1004, 202-39, 151-63, 109-23, 74-86. The Office withdrew its pre-*Alice* eligibility rejections when the applicant “clarif[ied] that the computer system comprises at least one processor and memory that are operable to perform such steps.” *Id.*, 102; EX-1005 ¶¶64. The Examiner did not consider Pike, Chowdhuri, or MacLeod. EX-1001, Cover.

D. The Level of Ordinary Skill in the Art

A POSITA would have had at least a bachelor’s degree in computer science or similar field, and approximately two years of industry or academic experience in a field related performing data analytics and/or related data processing tasks. EX-1005 ¶¶67-69; *In re GPAC Inc.*, 57 F.3d 1573, 1579 (Fed. Cir. 1995). Work experience can substitute for formal education and additional formal education can substitute for work experience. *Id.*

E. Claim Construction

In two cases not involving Petitioner, the Eastern District of Texas construed several terms to have their “plain meaning” and provided express constructions for two other terms:

<u>Term</u>	<u>Construction</u>
“a plurality of mapping functions that are each user-configurable”	“two or more mapping functions that are each configurable by a user”

“data group”	“a group of data and a mechanism for identifying data from that group”
--------------	--

EX-1007, 13-31; EX-1008, 10-19. The preambles of claims 1 and 17 were also found limiting. EX-1007, 19-23. For mapping the data groups “differently,” the court rejected arguments that the claims require different mapping *functions*. *Id.*, 16-18. The petition applies these constructions without acquiescing to their propriety.

IV. OVERVIEW OF THE PRIOR ART

A. Pike

Pike discloses worker processes (304, 308) in large-scale data processing system 300 that invoke mapping and reducing operations. Pike, 8:65-11:55, Fig. 3; EX-1005 ¶70-75. Inputs to the system include “input files 302,” which “can include a variety of data types,” including “text files” and “tables.” Pike, 9:56-10:7. The input files are partitioned into data blocks by worker processes 304 and provided to map tasks that “appl[y] [an] application-specific map() operator ... so as [to] generate intermediate data values.” *Id.* The intermediate data are sorted and assigned to reduce tasks by worker processes 308, which “merge[] or otherwise combine[] the sorted intermediate data values having the same key, and write[] the key and combined values” to output files 310 using “application-specific reduce() operator[s].” *Id.*, 11:6-55. Like the ’610 patent, as shown in annotated Figure 3,

Pike's process includes **partitioned input data**, **mapping tasks**, **intermediate data sorted by key-value pair**, and **reducing tasks with output data**:

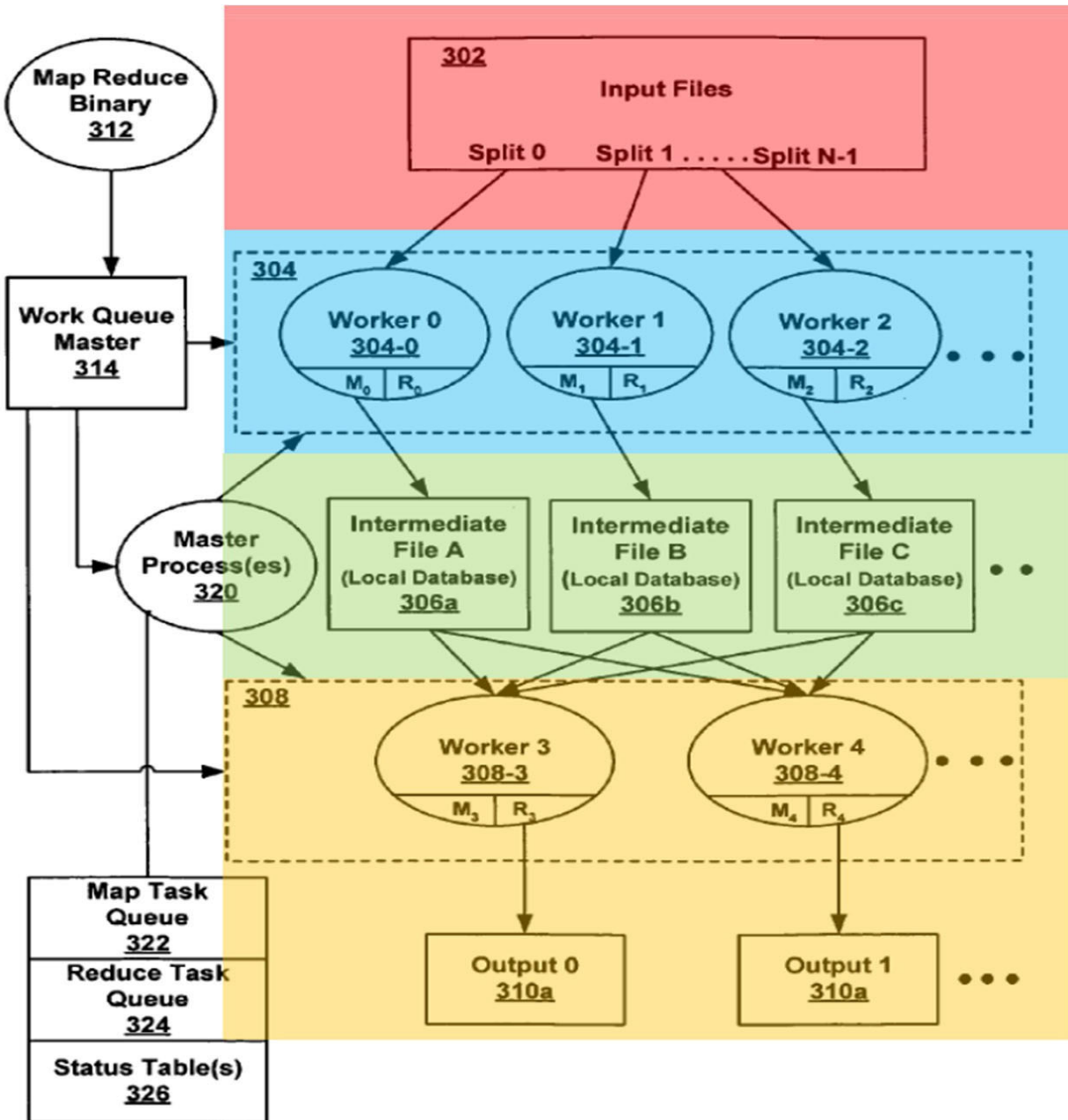


Figure 3

Id., Fig. 3 (annotated).

Pike also discloses tracking input files through the map and reduce phases:

Task Status Table					
Task ID	Status	Process	Input Files	Output Files	...
Map0000	Completed	P0000	12340	12341	...
Map0001	Completed	P0001	12344	12345	...
⋮	⋮				
Map0103	In Progress	P0010	12030	12031	...
Map0104	Failed	P0011	10101	12102	...
Map0103b	In Progress	P0010	12030	12031	...
⋮	⋮				
Red0000	In Progress	P0033	12340, ...	14000	...
Red0001	Waiting	-	-	-	...

Figure 7A

Id., Fig. 7A (tracking “input file 12340” through map and reduce tasks).

B. Chowdhuri

Chowdhuri discloses running “multiple instances of [] operators on different data sets,” where “large volumes of data are being scanned, joined, and sorted.” Chowdhuri ¶¶126-28. In one embodiment, Chowdhuri discloses using iterators—“self-contained software objects that accept a stream of rows from one, two, or multiple data sources”—to query data in a “serial operator tree 300.” *Id.* ¶¶129-31, Fig. 3A. The desired output is customer orders by state. *Id.* ¶¶126-27. In this query, “order and customer tables (data 308, 309) are scanned by two separate scan iterators, 301 and 302, respectively.” *Id.*

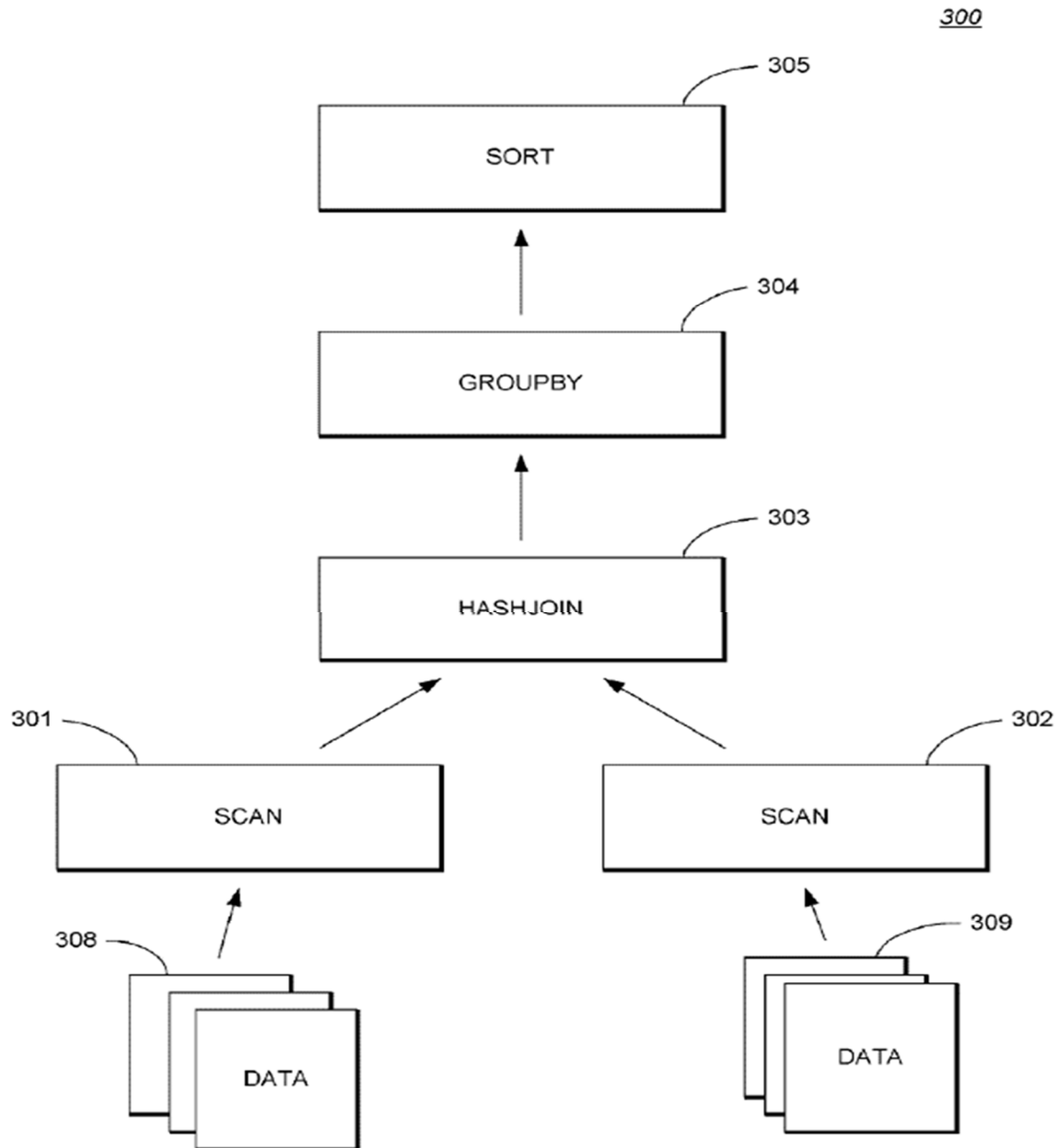


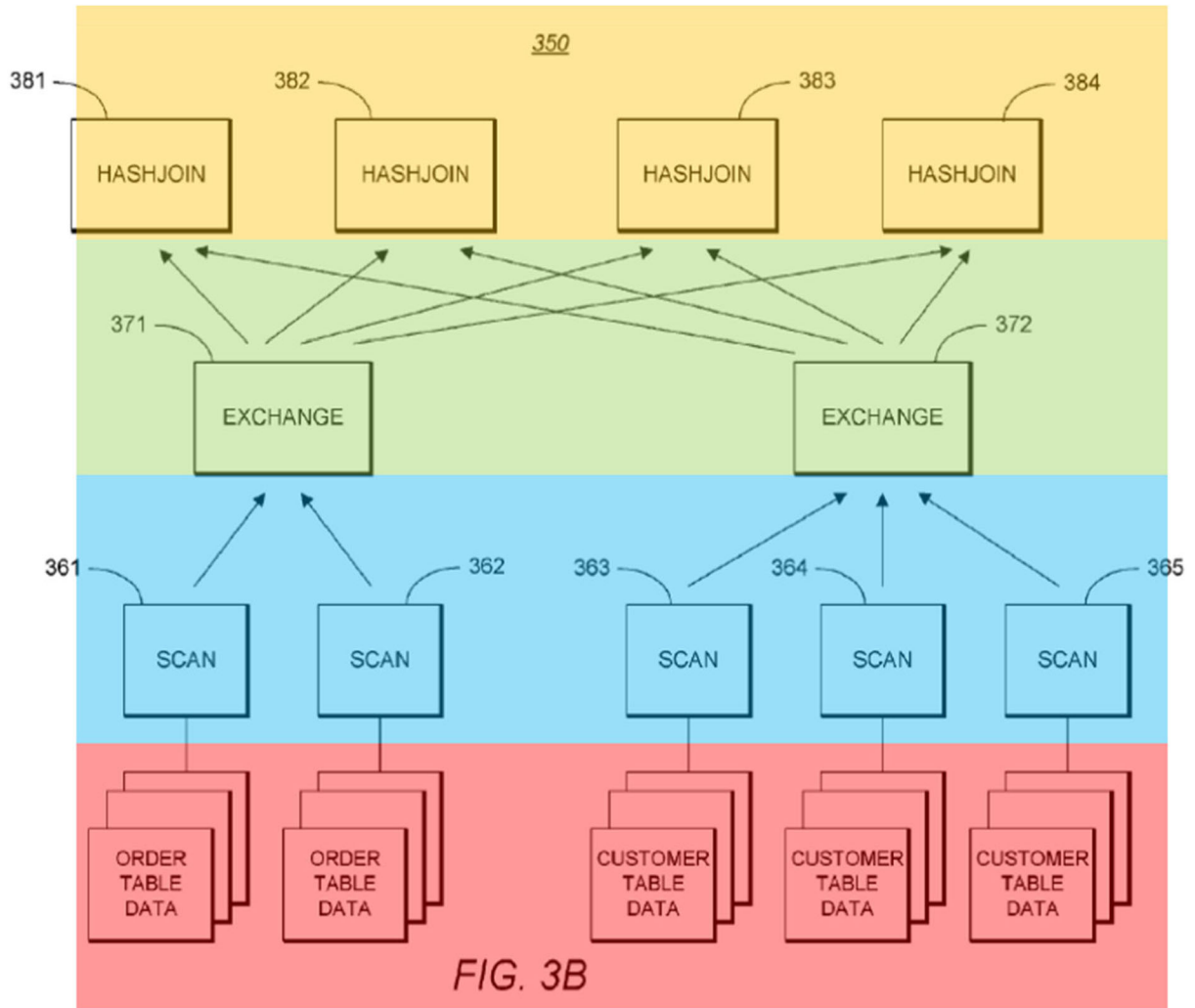
FIG. 3A

Id., Fig. 3A.

The scanning results are sent to a hash join iterator 303 to be matched by a customer, after which the “joined rows are then sent to the grouping iterator (GroupBy) 304 that performs a grouping operation” to group orders by state. *Id.*

¶¶126-27, 131. The sort iterator 305 then “orders for each of the states ... so that they appear in the order of the sum.” *Id.*

Chowdhuri discloses another embodiment using “exchange operators (iterators)” to “parallelize the serial query execution plan of FIG. 3A.” *Id.*, ¶135. The exchange operators “allow[] pipelining of intermediate results from one set of operators (i.e., producer tasks) and initiate[] another set of operators to process the next request (i.e., consumer tasks).” *Id.* ¶¶133-34. A POSITA would have understood that Chowdhuri’s iterators perform mapping and reducing tasks on data groups having different schema and track those input tables through the map and reduce phases. EX-1005 ¶¶76-80. Like the ’610 patent, Chowdhuri’s process also includes **partitioned input data**, **mapping tasks**, **intermediate data sorted by key-value pair**, and **reducing tasks** with output data:



Chowdhuri, Fig. 3B (annotated).

V. GROUNDS 1 AND 2

A. Claim 1

[1a] “A method of processing data of a data set over a distributed system,”³

Processing a data set over a distributed system was “conventional.” EX-1001, 1:6-27, 3:9-15; EX-1005 ¶83.

³ See EX-1026 for a full claim listing.

Pike also discloses this feature by disclosing “a framework for simplifying large-scale data processing” that is implemented “in a parallel and distributed processing environment” (**Grounds 1-2**). Pike, 1:10-26, 2:14-49. Pike discloses large-scale data processing system 300 that processes data over a “number of machines” and completes tasks faster by “distributing file reads over multiple local databases.” *Id.*, 8:65-9:54, 13:37-50. Pike additionally discloses exemplary computer system 400 for data processing system 300 that includes “one or more processing units (CPUs) 402,” memory 412, network or other communication interfaces/buses 410/414 for interconnecting these components, and “mass storage that is remotely located from ... central processing unit(s) 402.” *Id.*, 13:52-65, Fig. 4. Pike’s large-scale data processing system 300 discloses “processing data of a data set over a distributed system” because Pike discloses processing data over a system with distributed machines and remote storage. EX-1005 ¶¶83-85.

Chowdhuri also discloses this element by disclosing that its methods are “embodied in a multi-user computer system, such as a client/server system” (**Ground 2**). Chowdhuri ¶¶105-108. Chowdhuri discloses that its queries can be executed by dividing operations “into sub-tasks and executed in parallel across multiple storage units and CPUs.” *Id.* ¶¶128, 131. Chowdhuri further discloses exchange operators for “parallelizing a query [] to initiate as many iterators as possible and spread them across multiple disk and CPU resources.” *Id.* ¶¶132-35.

Chowdhuri's execution of iterators in a client/server system and across multiple CPUs discloses "processing data of a data set over a distributed system." EX-1005 ¶¶86-87. A POSITA would have found it obvious to implement the combined Pike-Chowdhuri system (described in greater detail below) using either of Pike's large-scale data processing system 300 or Chowdhuri's distributed system. *Id.* Doing so would have been nothing more than the simple substitution of one known element (one distributed computing system) for another to achieve the predictable result of distributing the processing of data over multiple machines to parallelize operations and reduce computation time. *Id.*

[1b] "wherein the data set comprises a plurality of data groups,"

The district court construed "data group" to mean "a group of data and a mechanism for identifying data from that group." EX-1007, 23-25; EX-1008, 8-9. Pike (**Ground 1**) and Pike-Chowdhuri (**Ground 2**) teach or render obvious this feature, under both its plain meaning and the district court's construction.

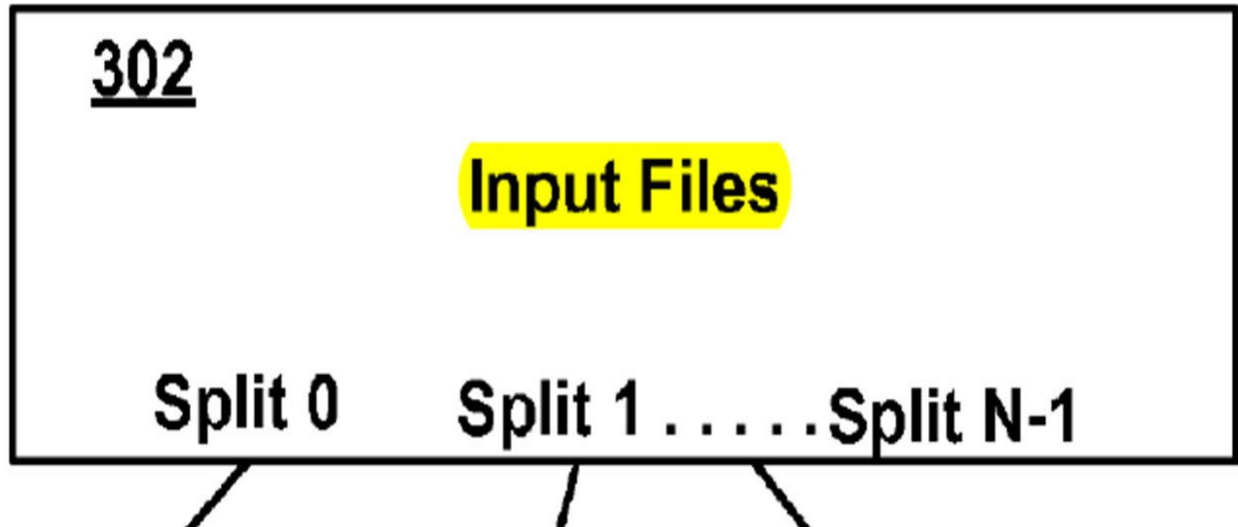
Pike discloses or renders obvious this feature by disclosing input data for system 300 that comprises "input files 302." Pike, 9:58-64, Fig. 3. Input files 302 include, among other things, "text files" and "tables." *Id.*, 10:1-4, 2:22-26. A POSITA would have recognized that Pike's input files teach or render obvious groups of data and a mechanism for identifying data from that group because input files include, for example, a file name. EX-1005 ¶¶88-91. A file name is an identifier

associated with the data in the file, and typically includes information about the content, type, and structure of the data in each file. *Id.* ¶91. This is a “mechanism for identifying data from [a data] group.” *Id.*

As an example, for an input file in Pike’s system, such as the “Employee table” as contemplated by the ’610 patent, a POSITA would have understood the input file would include, for example, a name (e.g., “Employee”) and an extension indicating its file type or structure (e.g., an extension such as “.csv” indicating a “comma separated file,” a common format for large-scale data processing). *Id.* ¶92. Other input files in Pike would have similar identifiers indicating the file content and structure. *Id.* Pike’s input files correspond to input tables in the ’610 patent because a POSITA would have understood that the Employee and Department tables of the ’610 patent would be stored as files and have similar identifiers. *See* EX-1001, 4:1-5:18, 7:18-23, 8:15-46, Figs. 4, 5; EX-1005 ¶92. Pike therefore teaches a data set comprising data groups.

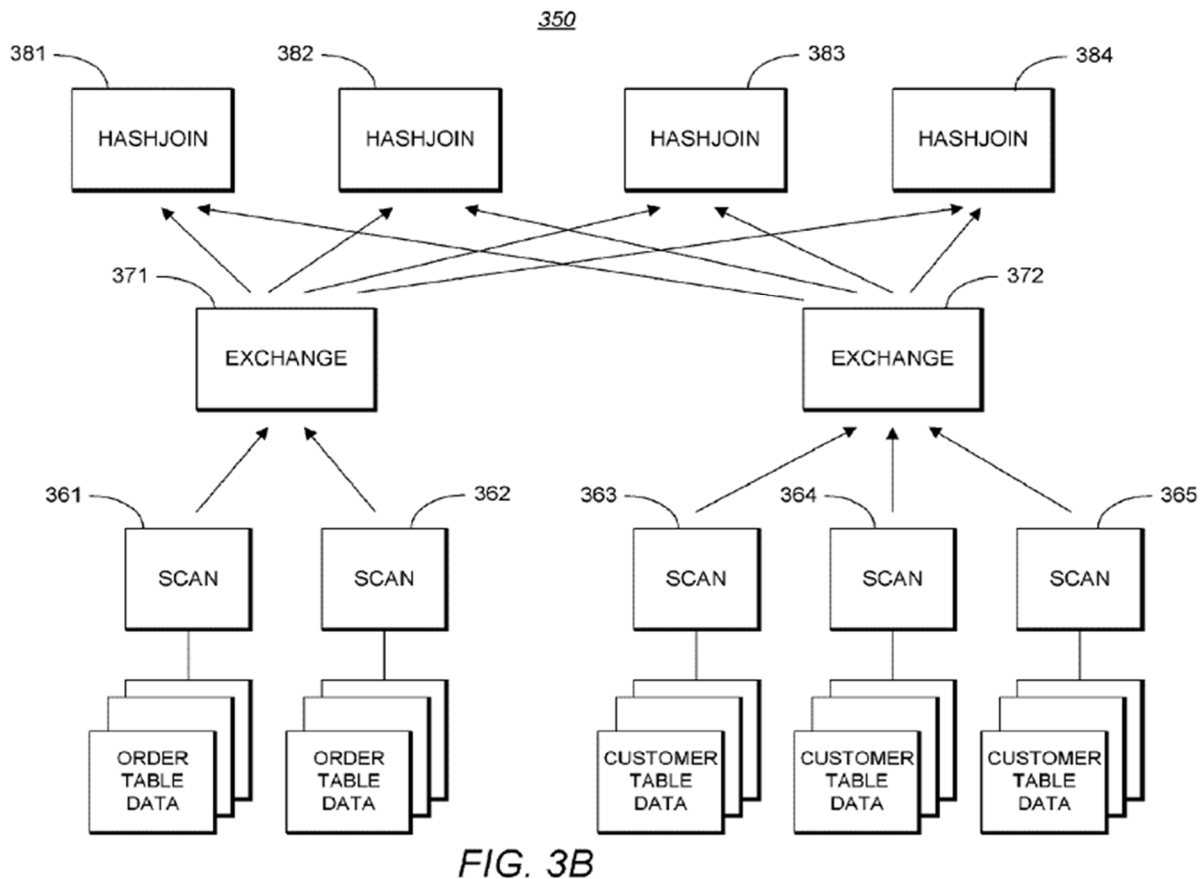
At minimum, a POSITA would have found it obvious to implement Pike’s input files and tables as data groups by associating an identifier of Pike’s input files with the file data because it would have allowed for locating that data group in Pike’s system and understanding the data in that file. EX-1005 ¶93. This would have enabled a MapReduce programmer to identify input files and write “application-

specific” map and reduce functions on those files. Pike, 10:1-7, 11:28-47, 14:12-27; EX-1005 ¶93.



Pike, Fig. 3 (annotated).

Chowdhuri also discloses this feature by disclosing “order” and “customer” tables (**Ground 2**). Chowdhuri ¶¶126-28, 131, 135, Figs. 3A-3B. Chowdhuri’s order and customer tables teach data groups because Chowdhuri’s table names provide a mechanism for identifying the data in the table (e.g., order and customer data in table format). *Id.*, ¶¶126-28, 131, 135; EX-1005 ¶94.



Id., Fig. 3B.

It would have been obvious to combine Pike with Chowdhuri to process input data that includes Chowdhuri's data groups. EX-1005 ¶95. Doing so would have involved nothing more than the simple substitution of Chowdhuri's input tables for Pike's input files, with predictable results. *Id.* A POSITA would have been motivated to do so, for example, based on Pike's suggestion of processing input "tables." Pike, 10:1-4, 2:22-26; EX-1005 ¶95.

A POSITA also would have been motivated to map and reduce Chowdhuri's tables in Pike's system 300 because Chowdhuri teaches and suggests performing

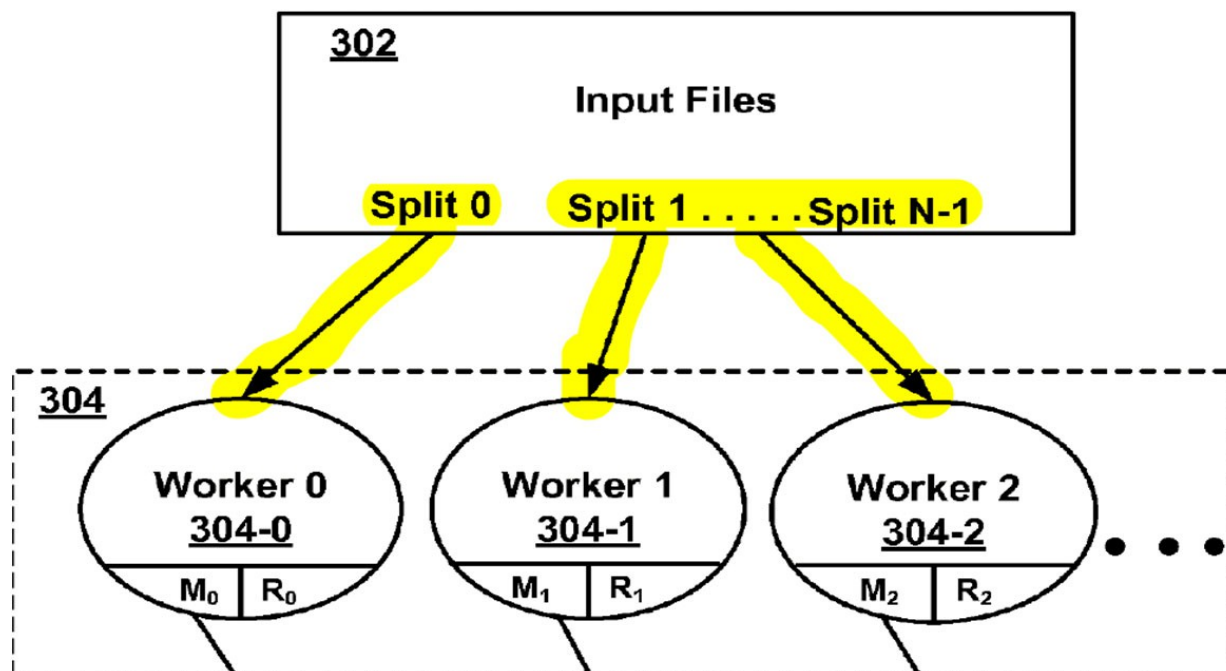
similar MapReduce operations on its data tables—scanning and merging its customer and order tables to identify orders from a specific “state.” EX-1005 ¶¶96; Chowdhuri ¶¶126-27, 131-35. A POSITA would have recognized that Chowdhuri’s scanning and hashjoin iterators correspond to Pike’s map and reduce tasks, and thus Chowdhuri’s tables and processing techniques were compatible with Pike’s system 300. EX-1005 ¶¶96. Indeed, Chowdhuri’s scan, hashjoin, GroupBy, and sort iterators correspond to the ’610 patent’s description of map and reduce functions. EX-1001, 1:20-27 (“[t]he map function iterates” and “[t]he reduce operation ... combines elements”), 5:19-39 (sorting), 2:64-66 (“a sort and group-by-key task”), 5:48-6:7 (group iterator), 6:34-7:16 (“hashtable” iterators), 10:1-7 (“the reducers include a sort, group-by-key and combine task”); EX-1005 ¶¶96.

A POSITA would have looked to Pike to map and merge Chowdhuri’s tables using Pike’s distributed system to, for example, leverage Pike’s distributed system and its fault tolerant task scheduling. EX-1005 ¶¶97; Pike, 4:28-53, 6:37-54, 6:55-7:25, 11:57-13:3. This would have been nothing more than combining known elements in the prior art to achieve predictable results. EX-1005 ¶¶97. The resulting combination would have processed Chowdhuri’s tables using, for example, Chowdhuri’s scanning, hashjoin, GroupBy, and sort iterators as map and reduce tasks in Pike, while employing Pike’s fault-tolerant architecture to do so. *Id.*

[1c] “the method comprising: partitioning...”

Conventional MapReduce partitioned its input data into key-value pairs before mapping. EX-1001, 2:19-52, Fig. 1; EX-1005 ¶98.

Pike discloses this by partitioning the data of its data groups into a plurality of data partitions that each have a plurality of key-value pairs (**Grounds 1-2**). For example, Pike discloses that its input files 302 are “[s]plit” via “[t]he division of input files 302 into data blocks 0, ... , N-1.” Pike, 9:58-10:19, 14:57-63, Fig. 3 (below). Pike’s data blocks are then “treated as key/value pairs” before being read by worker processes 304 to apply an “application-specific map() operator.” *Id.*, 9:58-10:19; EX-1005 ¶99.

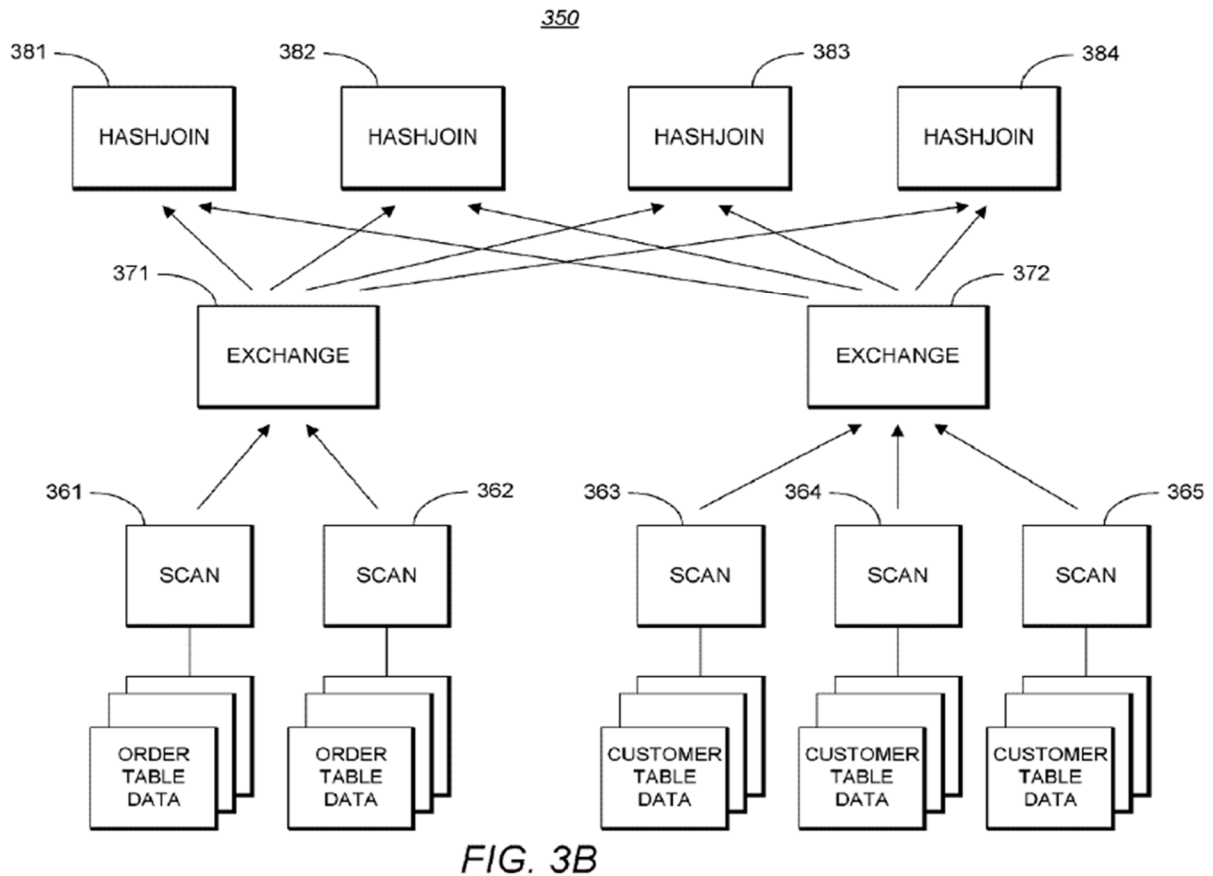


Pike, Fig. 3 (annotated).

Pike teaches or renders obvious partitioning the data “of each” data group (e.g., input file) into a plurality of data partitions because Pike discloses a user “set[ting] an optional flag, or specify[ing] a parameter, so as to control the size of the data blocks into which the input files are divided.” *Id.*, 9:58-64. From this, a POSITA would have understood that Pike teaches, for example, setting a data block size that is smaller than its input files, thereby partitioning each input file into multiple blocks. EX-1005 ¶100. Implementing Pike to partition each input file would have been obvious and well within the abilities of any POSITA or MapReduce programmer. *Id.* ¶101. Indeed, Pike’s “division of input files 302 into data blocks 0, ... , N-1” suggests doing so. Pike, 9:58-60. In the case of a small data set and/or one with large input files, for example, a POSITA would have been motivated to split each input file into several blocks to distribute the processing of input data across many processors. EX-1005 ¶¶101. This reduces computation time and takes advantage of Pike’s parallel processing capabilities. *Id.*

Chowdhuri also teaches these features (**Ground 2**). Chowdhuri discloses that its input tables have key-value pairs, including customers, orders, and “state” (i.e., state within the United States). Chowdhuri ¶¶126-27 (disclosing “group by state” and sorting “orders for each of the states”), ¶131 (“state codes”); EX-1005 ¶102. Chowdhuri further discloses that its system divides the “execution of each query operation ... into sub-tasks [to be] executed in parallel across multiple storage units

and CPUs.” Chowdhuri ¶¶128, 131-34. In Figure 3B, the data from the customer and order tables are each partitioned and scanned by individual scan iterators, and the output is provided to exchange operators that “affect parallelism.” *Id.* ¶¶132-35. From this, a POSITA would have understood that Chowdhuri partitions each of its input tables, where those partitions include key-value pairs. EX-1005 ¶102.



Chowdhuri, Fig. 3B.

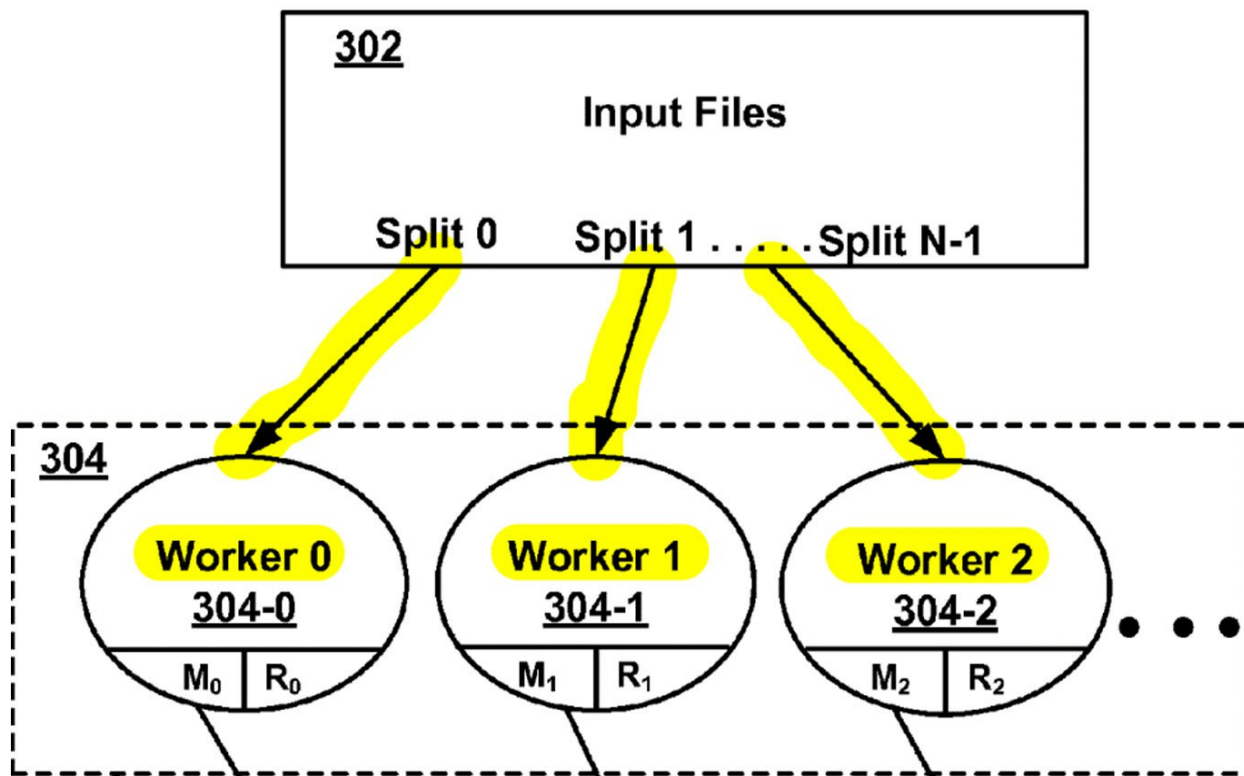
A POSITA would have been motivated to combine Chowdhuri’s partitioning with Pike’s large-scale data processing system 300, as discussed above in element [1b], to take advantage of Pike’s parallel processing system and fault tolerant task

scheduling. EX-1005 ¶103; Pike, 4:28-53, 6:55-7:25, 11:57-13:3. The combination would have been nothing more than combining known elements in the prior art to yield the predictable result of efficiently processing Chowdhuri's input tables by partitioning each table into a plurality of partitions and processing them in a distributed, fault-tolerant system. EX-1005 ¶103. The resulting combination would have processed Chowdhuri's tables using, for example, Chowdhuri's scanning, hashjoin, GroupBy, and sort iterators as map and reduce tasks in Pike, while employing Pike's fault-tolerant architecture to do so. *Id.*

[1d] “providing each data partition...”

Providing each data partition to a selected one of a plurality of mapping functions, where each map function is user-configurable, was conventional in MapReduce. EX-1001, 2:33-35, Fig. 1; EX-1005 ¶104.

Pike teaches these features (**Grounds 1-2**) by disclosing that each of its data blocks “are read by application independent worker processes 304-0, ... , 304-N-1, as shown in FIG. 3.” Pike, 9:65-10:27, 14:57-63, Fig. 3; EX-1005 ¶105.



Pike at Fig. 3 (annotated).

Pike discloses that “[e]ach of the worker processes 304 ... applies [an] application-specific map() operator to the *respective* input data block so as [to] generate intermediate data values.” *Id.*, 9:65-10:27. The output is “collected and written to one or more intermediate files 306.” *Id.* Pike’s application of a map task to each “respective input data block” teaches providing each data partition to a selected one of a plurality of mapping functions. *Id.*; EX-1005 ¶¶106-07. This is consistent with the ’610 patent, which discloses portions of its tables being provided to individual map tasks to generate intermediate data. EX-1001, 8:15-24, Fig. 5; EX-1005 ¶107.

Pike's map operators are "user-configurable." Pike discloses that "an application programmer can create a script or program using application software 422, which includes one or more [map, reduce, and partition] operators 424, 426 and 428." Pike, 14:35-39, 14:12-27, 4:28-53, 5:26-39, 8:34-45; EX-1005 ¶108. Pike also discloses that its map operators are "application-specific" and thus developed by programmers based on the use case. EX-1005 ¶108; Pike, 2:22-26, 2:31-36, 3:59-64, 4:28-53. Pike therefore discloses "two or more mapping functions that are each configurable by a user." EX-1007, 13-15; EX-1008, 8-9.

Chowdhuri also discloses these features (**Ground 2**). Chowdhuri discloses that its customer and order tables are partitioned for parallel processing and then provided to individual scanning iterators 361-365. Chowdhuri, ¶¶132-35, Fig. 3B. In one example, each scanning iterator operates on its own thread and scans portions of the customer and order tables to identify the customer and order state from each table. *Id.* ¶¶126-27, 132-35. This teaches "providing each data partition to a selected one of a plurality of mapping functions." EX-1005 ¶109.

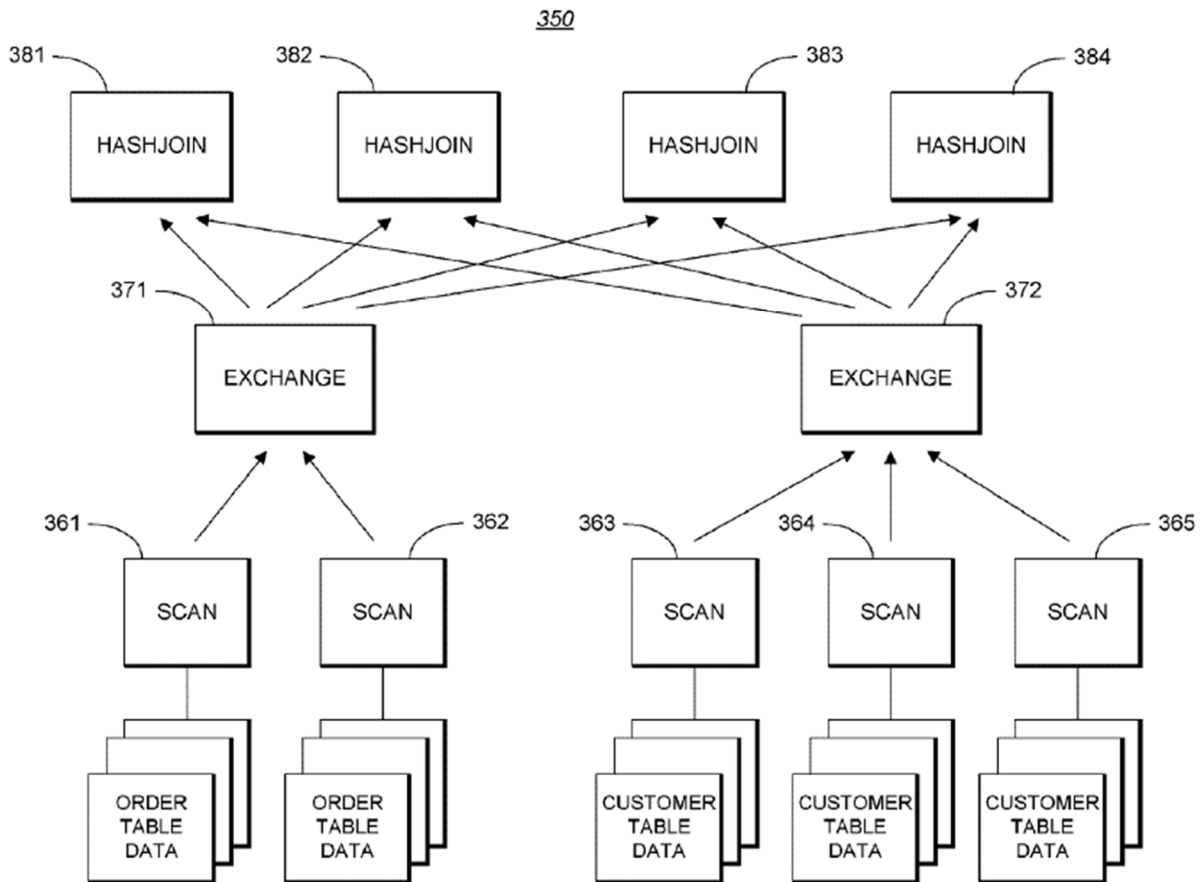


FIG. 3B

Chowdhuri, Fig. 3B.

As explained above, a POSITA would have understood that Chowdhuri’s scan iterators teach or suggest techniques that correspond to Pike’s map functions and those described in the ’610 patent. EX-1005 ¶110. Chowdhuri’s iterators are “self-contained software objects that accept a stream of rows from one, two, or multiple data sources.” Chowdhuri, ¶¶130-35, Figs. 3A-3B. The scan iterators “scan[] database rows from a database on disk” or “a table or index.” *Id.* ¶¶130, 61 (“Scanning rows from a table on disk can be the behavior of one type of iterator.”).

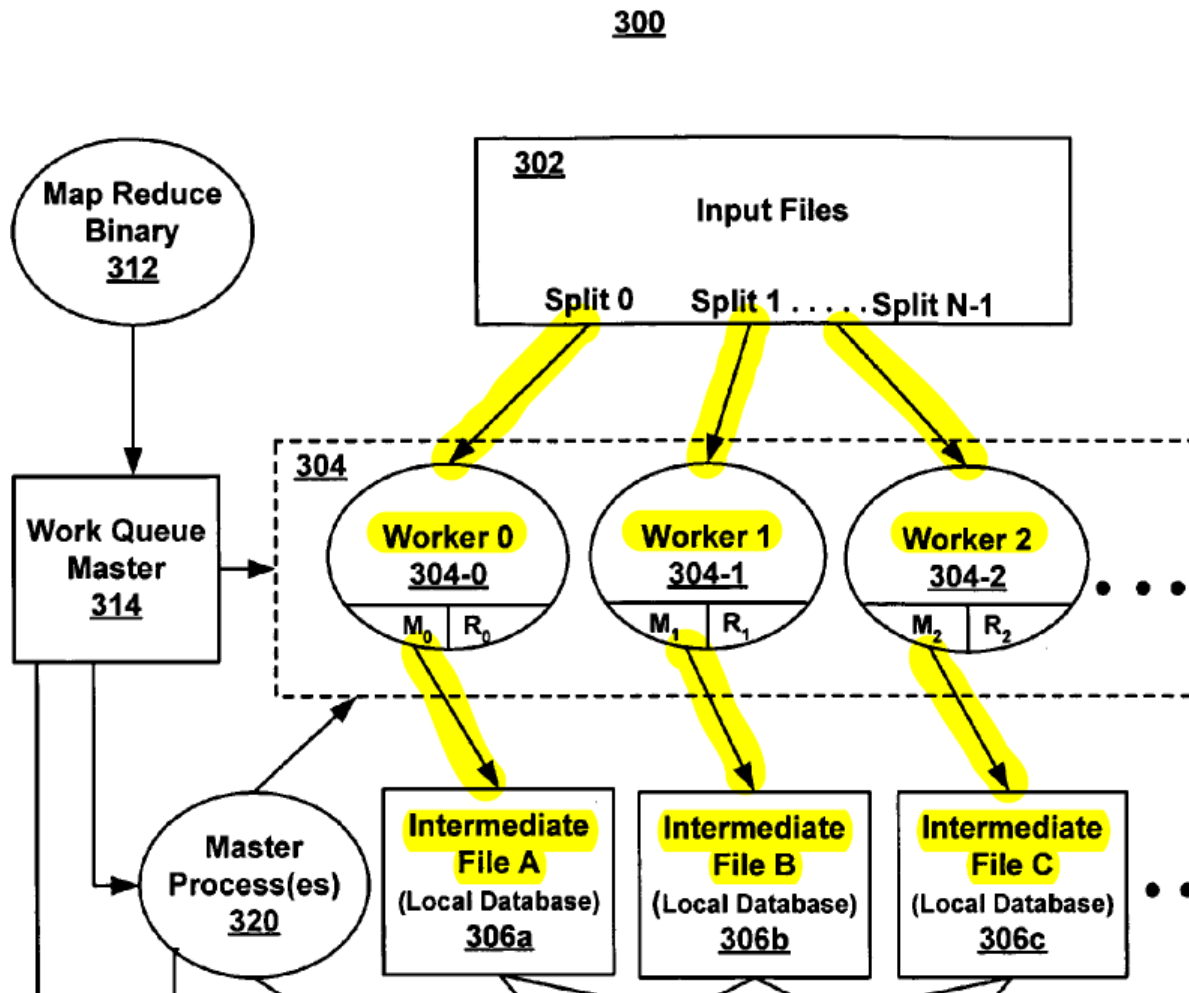
The scanning produces “intermediate results” that an exchange operator “pipelin[e]s ... from one set of operators (i.e., producer tasks) and initiates another set of operators to process the next request (i.e., consumer tasks).” *Id.* ¶134. This corresponds with the map functions of Pike and the ’610 patent. Pike, 7:29-58 (“[T]he map() operator may be required to process the input data one record at a time, proceeding monotonically from the first record to the last record in the data block being processed.”); EX-1001, 1:20-27 (“The map function iterates over a list of independent elements, performing an operation on each element as specified by the map function.”). Chowdhuri therefore teaches or renders obvious providing each data partition to a selected one of a plurality of mapping functions. EX-1005 ¶111. Chowdhuri’s scan iterators are user-configurable because Chowdhuri discloses user-defined pseudo code for scanning customer and order tables for customer and order state. Chowdhuri ¶¶126-27 (grouping data from tables “by state.”), ¶131 (grouping by “state codes”). The scan iterators are therefore configured for particular inputs. EX-1005 ¶111.

It would have been obvious to a POSITA to implement Chowdhuri’s scan iterators as Pike’s map operators in Pike’s system 300. EX-1005 ¶112. A POSITA would have done so to identify customer and order states based on Chowdhuri’s input tables. *Id.* Combining Chowdhuri’s scan iterators with Pike’s system 300 would have yielded predictable results in a system that maps Chowdhuri’s

partitioned input tables by scanning the data over a distributed system using multiple, parallel processes. *Id.* A POSITA would have been motivated to do so, as discussed above, to leverage Pike’s distributed system and fault tolerant task management. *Id.*

[1e] “to independently output a plurality of lists of values...,”

Pike discloses or at least renders this feature obvious (**Grounds 1-2**). Pike’s map functions “independently output[]” intermediate data because each worker process 304 invokes a map task to individually output intermediate data. EX-1005 ¶113. For example, Pike discloses that each worker process 304 “invokes a map thread to handle an assigned map task.” Pike, 8:65-9:19, 10:4-27, 14:57-15:6. In one embodiment, “each of the intermediate files 306 receives output from *only one* worker process 304, as shown in FIG. 3.” *Id.*, 10:13-15, Fig. 3. Figure 3 depicts each worker process 304 independently outputting intermediate data to intermediate files 306:

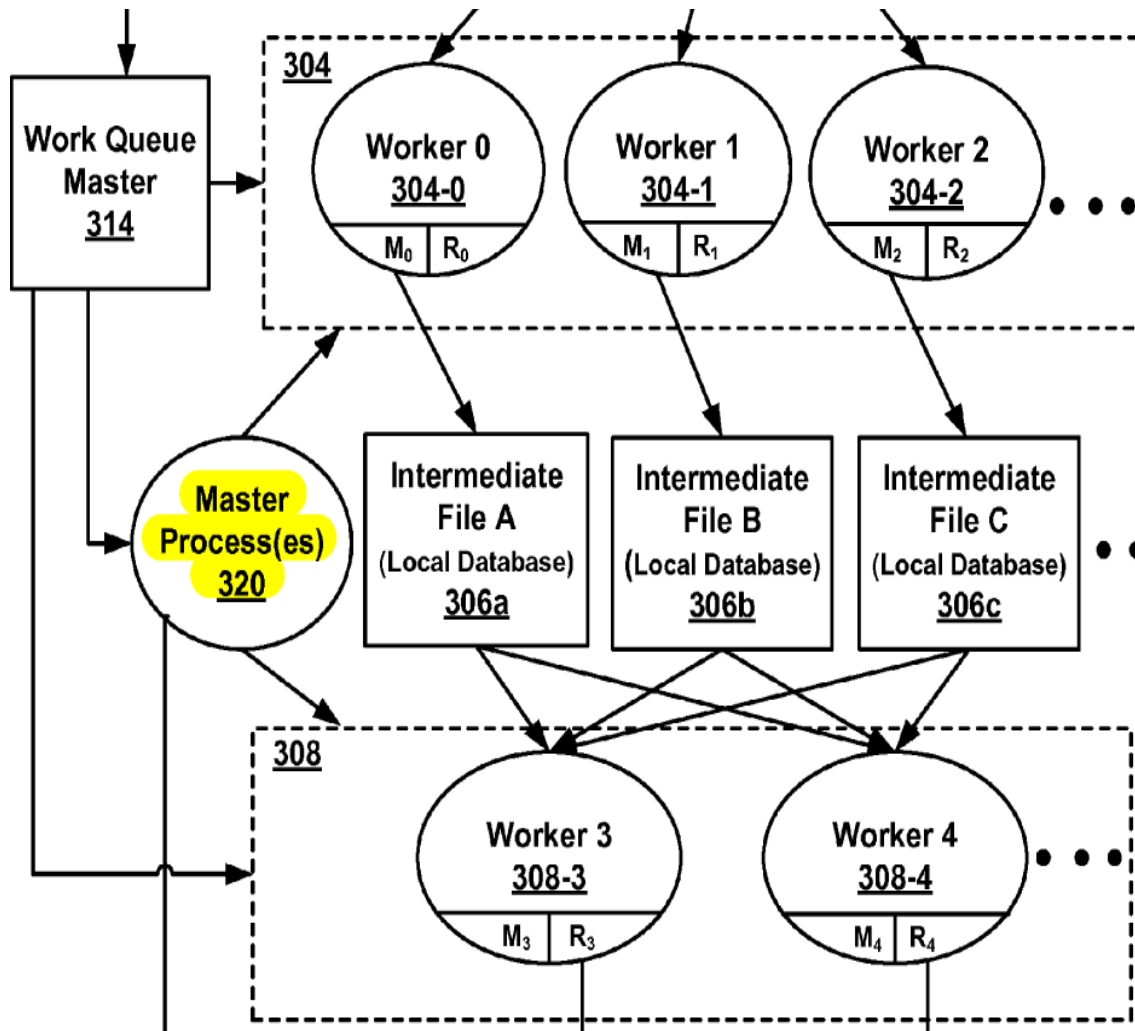


Id., Fig. 3 (annotated).

Pike also discloses outputting “a plurality of lists of values for each of a set of keys found in such map function’s corresponding data partition,” as claimed. For example, Pike explains that “[t]he input data blocks 0, ... , N-1, which may in some embodiments be treated as key/value pairs, are read by application independent worker processes 304-0, ... , 304-N-1.” *Id.*, 9:65-10:19. The worker processes 304 “appl[y] [an] application-specific map() operator to the respective input data block so as [to] generate intermediate data values.” *Id.* In one example, Pike discloses

“output[ting] a key/value pair for every word *w* in every document in [a] collection, where the key/value pair is $\langle w, 1 \rangle$.” *Id.*, 10:31-11:4, 14:60-63. The output values from the map operators are stored in intermediate files 306 that are read by worker process 308 “in accordance with the key of the key/value pairs in the intermediate data.” *Id.*, 11:28-31; EX-1005 ¶114.

Pike further discloses that the values stored in intermediate files 306 “form corresponding intermediate data for [a] data group and identifiable to that data group” by disclosing master process 320, which tracks the processing of each input file using identifiers through mapping and reducing. *Id.*, 13:9-21, Fig. 3 (item 320); EX-1005 ¶115. Pike explains that “master process 320 is responsible for assigning tasks to the worker processes 304 and 308 and for tracking their status and output.” Pike, 13:9-21.



Id., Fig. 3 (annotated).

Pike’s master process 320 “maintains status tables 326 for managing tasks, as described with respect to FIGS. 7A and 7B.” *Id.*, 13:7-17, Fig. 7A. In Figure 7A, map and reduce tasks are tracked according to “Input Files” (i.e., Pike’s “data groups”). *Id.*, Fig. 7A. According to Pike, “the input files field [of the task status table] may specify a portion of an input file (e.g., where the portion comprises a data block) to be processed by the task.” *Id.*, 6:55-7:12. As shown in Figure 7A, reproduced below, process P0000 (corresponding to Map task Map0000) is associated with input file

12340, whose status is “completed,” and process P0330 (corresponding to Reduce task Red0000) is associated with the same input file 12340. *Id.*, Fig. 7A. Pike therefore associates an “input file” identifier, which “specif[ies] a portion of an input file,” with an input file through the map and reduce tasks. EX-1005 ¶116.

Task Status Table					
Task ID	Status	Process	Input Files	Output Files	...
Map0000	Completed	P0000	12340	12341	...
Map0001	Completed	P0001	12344	12345	...
⋮	⋮				
Map0103	In Progress	P0010	12030	12031	...
Map0104	Failed	P0011	10101	12102	...
Map0103b	In Progress	P0010	12030	12031	...
⋮	⋮				
Red0000	In Progress	P0033	12340, ...	14000	...
Red0001	Waiting	-	-	-	...

Figure 7A

Pike, Fig. 7A (annotated).

Pike’s tracking input files through the map and reduce processes using input file identifiers, as shown in Figure 7A, and generating intermediate data that is identifiable to a particular input file (e.g., input file “12340”), teaches outputting “intermediate data for [a] data group and identifiable to that data group.” EX-1005 ¶117. The intermediate data for input file “12340” is stored in output file 12341. *Id.*;

Pike, Fig. 7A. A POSITA would have understood that, because input file “12340” specifies a portion of one of Pike’s input files, “12340” is an identifier that associates intermediate data (e.g., output file “12341”) with a particular input file. EX-1005 ¶117. This is indistinguishable from the ’610 patent, which describes using identifiers to associate intermediate data with its Employee or Department input tables. EX-1001, 4:1-22, Fig. 4; EX-1005 ¶117.

To the extent it is argued that Pike does not expressly disclose outputting corresponding intermediate data for that data group and identifiable to that data group (Pike does), it would have been obvious to include an identifier associated with Pike’s input files to track them through the map and reduce tasks. EX-1005 ¶118. Pike’s status tables 326 suggest doing so. *Id.* Data identifiers were not novel at the time of the ’610 patent. *Id.* Modifying Pike such that the input files of its task table are associated with and track data from particular input files through map and reduce processing would have been nothing more than applying a known technique of tracking data to Pike’s system 300 to yield a predictable result. *Id.* The addition would have involved basic encoding of an identifier for each input file and tracking those identifiers. *Id.*

A POSITA would have been motivated to track input files through the map and reduce process for several reasons, including allowing Pike’s system to track which output data from the map and reduce processes was derived from a particular

file. EX-1005 ¶119. This could later be used in a post-processing operation to rerun reduce tasks on a subset of intermediate data. *Id.* For example, particular input files could be omitted from the intermediate data before being sent to the reduce tasks. *Id.* This would improve efficiency of the system by enabling a data set to be selectively reduced after it is mapped, without requiring the desired subset of input files to be re-mapped. *Id.* Particular input files would instead be excised from the intermediate data before reducing. *Id.*

This feature also would have been obvious in view of Chowdhuri (**Ground 2**). In one example, Chowdhuri's scan iterators select a state from its customer table and the order amount from its order tables (i.e., "[S]elect state, [S]um(order_item)," "from customer c, order o"). Chowdhuri ¶¶126-27. The set of keys in Chowdhuri are "customer_id" in each table (i.e., "where o.customer_id=c.customer_id"). *Id.* The data from the scanning iterators is then joined by customer and grouped by state and ordered by order sum (i.e., "group by state" and "order by sum(order_item)"). *Id.* This discloses outputting "a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition." A POSITA would have understood that an exemplary Chowdhuri value from its customer table would be "Virginia" based on a customer key, and an exemplary value from its order table would be "\$25" based on a customer key. EX-1005 ¶120.

Identifiers “o” and “c” correspond to Chowdhuri’s order and customer tables, respectively, and identify the data from each table through the scan, join, and group processes. This teaches “form[ing] corresponding intermediate data for that data group and identifiable to that data group.” EX-1005 ¶121. A POSITA would have understood that the results of Chowdhuri’s scan iterator (e.g., map function) are identifiable to its input tables because the results are logically merged by customer state and customer order based on different input data to yield orders by state. *Id.*; Chowdhuri ¶¶126-27, 131, 135. The customer state and customer order are therefore identifiable to each input table. EX-1005 ¶121.

A POSITA would have found it obvious to combine Pike and Chowdhuri to implement Chowdhuri’s table identifiers through Pike’s map and reduce processes. EX-1005 ¶122. This would have enabled Pike to effectively merge Chowdhuri’s tables of different data based on a common key (e.g., by “customer_id”), as Chowdhuri suggests. *Id.* For larger data sets with multiple tables, tracking input tables as Chowdhuri suggests would have enabled alternative merging based on variables other than by customer. *Id.* Doing so further would have allowed the output data from Pike’s map and reduce processes to be associated with particular tables for subsequent postprocessing, as discussed above. *Id.* For example, this would allow postprocessing operations, such as omitting particular tables from the map/scan results for selective reducing/merging in the final result. *Id.* A POSITA

would have combined Chowdhuri and Pike as explained above to scan and merge these input tables in Pike, as Chowdhuri suggests. *Id.*

[1f] “wherein the data of a first data group has a different schema than the data of a second data group”

The '610 patent acknowledges that “[c]onventional MapReduce” could process data having “different schemas,” but alleges only that it could not do so “efficiently” or “[p]ractical[ly].” EX-1001, 3:9-20. This feature was therefore known in the art. EX-1005 ¶¶123-25. The inventors acknowledged that MapReduce could process data “from multiple sources”— including heterogenous sources having different schema—and track the data with a simple “data source attribute.” EX-1022, 1029-30. The '610 patent claims nothing more than a use-case of MapReduce that specifies data in different input data groups and tracking those groups. EX-1005 ¶125. As explained above, data identifiers were not new with the '610 patent. *Id.*

Pike (**Grounds 1-2**) and Pike-Chowdhuri (**Ground 2**) also render this feature obvious. Pike discloses that its “input files 302 can include a variety of data types typically used in data processing systems,” including “text files” and “tables.” Pike, 10:1-7, 2:17-41. In one example, Pike processes “a large collection of documents.” *Id.*, 10:31-48. Pike’s input text files and tables teach data of a first data group having a different schema than the data of a second data group. EX-1005 ¶126.

To the extent Pike does not expressly disclose this feature, a POSITA would have found it obvious to implement Pike to process a data set of two or more input

files (i.e., “data groups”) having different schema (**Grounds 1-2**). *Id.* ¶127. Implementing Pike to process input files of different schema would have been nothing more than implementing system 300 in a manner Pike suggests: to analyze “a variety of data types.” Pike, 10:1-7; EX-1005 ¶127.

Indeed, a POSITA would have recognized that Pike’s map and reduce operators are “application-specific,” meaning they are developed by programmers to be application dependent and specific to a given use case. EX-1005 ¶128; Pike, 2:22-26, 2:31-36, 3:59-64, 4:28-53 (“Generally, the map() operator specifies how input data is to be processed to produce intermediate data and the reduce() operator specifies how the intermediate data values are to be merged or otherwise combined.”). Pike discloses that MapReduce programmers need to know the input files, where to send output data from the map and reduce processes, and how to encode “two application-specific data processing operators, hereinafter referred to as map() and reduce(),” to process the specific input files. Pike, 4:33-40, 6:1-12, 10:4-7, 10:39-50, 11:36-38, 14:19-27. A programmer therefore codes the map and reduce functions to operate on particular input data, and those functions change for different applications and different input files. EX-1005 ¶128.

Analyzing input files of a different schema, including different type, format or content, would have been one known use-case and obvious implementation of Pike’s application-specific data processing operators. *Id.* ¶129. For example, a

POSITA would have found it obvious to map a variety of different tables or documents to merge them based on a common key, as Pike teaches and suggests. *Id.*; Pike, 10:31-11:4, 4:28-53.

Indeed, POSITAs knew about, were implementing, and were using MapReduce to analyze large data sets well before the '610 patent. EX-1005 ¶130; *see* Sections III.A, III.B. Given their education and training, POSITAs would have been able to write map and reduce functions and customize these functions for different input files, including files of different schema. EX-1005 ¶130. Mapping input files of different schema in Pike therefore would have been routine for a POSITA. *Id.* The result would be predictable: a system that analyzes a data set of input files having different type or format, partitions those input files, and tracks the files through the map and reduce phases to merge them according to key and/or input file. *Id.*

A POSITA would have been motivated to implement Pike in this way, for example, because it would have allowed selective merging based on particular keys and particular input files. *Id.* As discussed above in Section [1e], tracking each of diverse input files would have allowed for post-processing operations, including omitting particular input files from intermediate data to selectively reduce on a subset of the intermediate data (e.g., by removing particular inputs from the intermediate data based on input file). This would have improved Pike's efficiency

and reduced computation time by avoiding rerunning the map process on the subset of input files. *Id.* 131-32.

Pike-Chowdhuri (**Ground 2**) also renders this feature obvious. Chowdhuri discloses executing a data query using a series of iterators to scan an order table 308 and customer table 309. Chowdhuri ¶¶126-35, Figs. 3A-3B; EX-1005 ¶133.

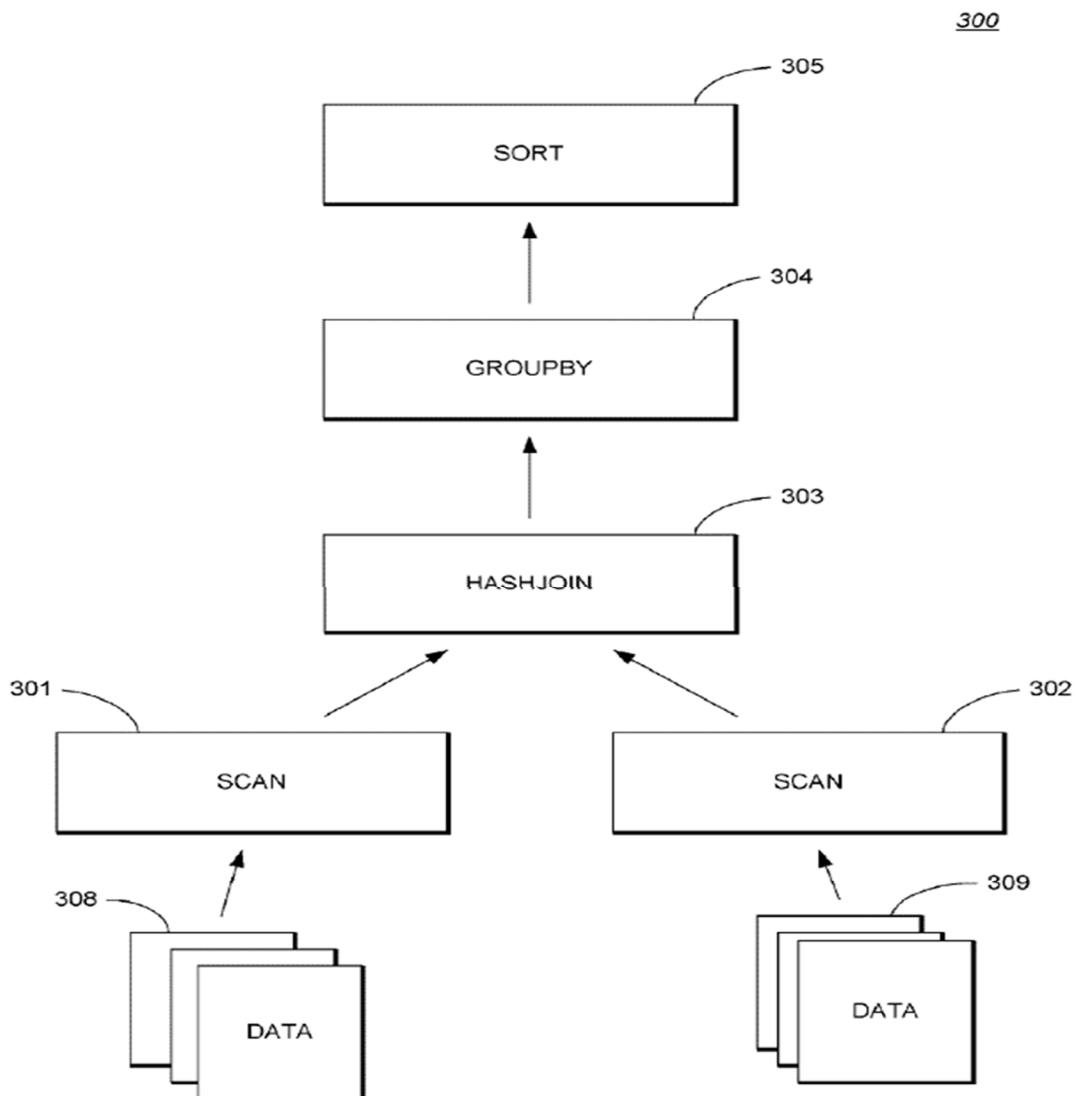


FIG. 3A

Chowdhuri, Fig. 3A.

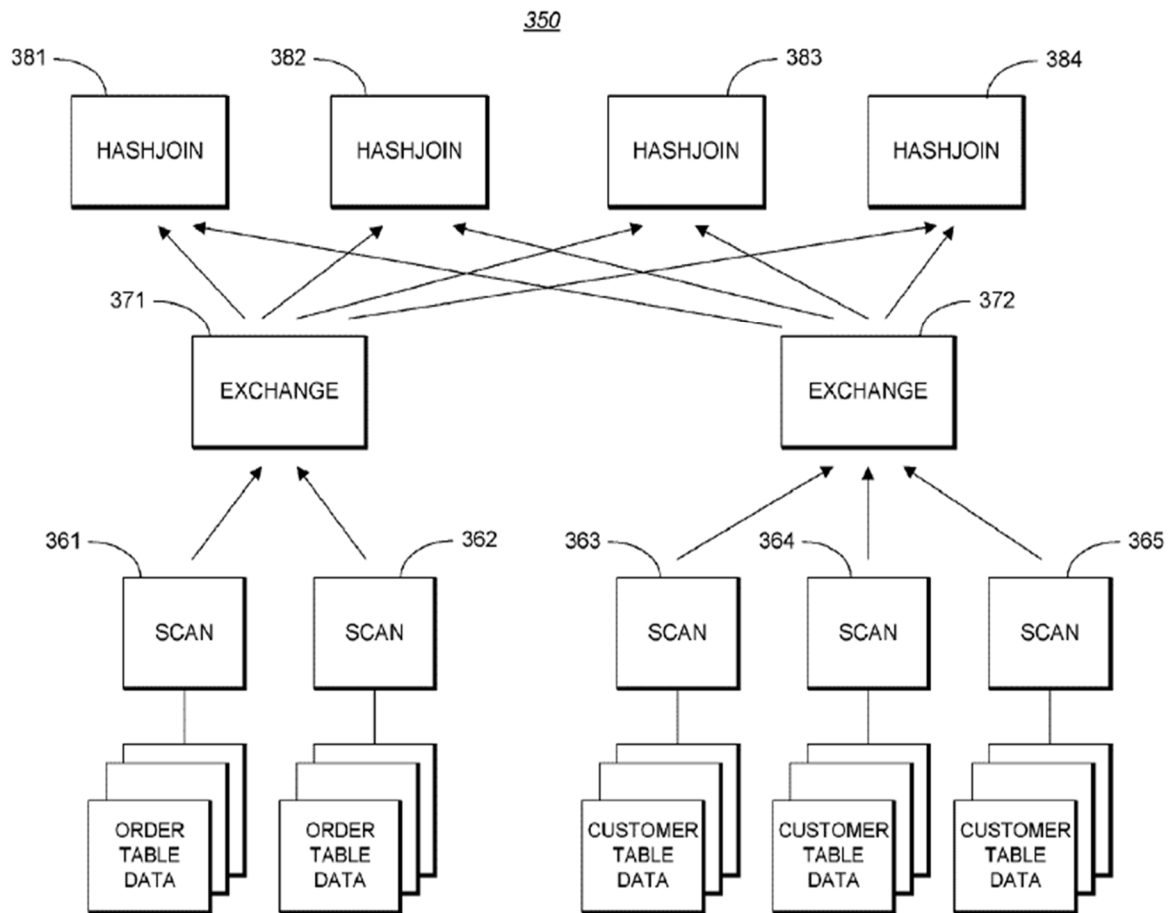


FIG. 3B

Id., Fig. 3B.

As discussed in elements [1b] and [1c], a POSITA would have understood that Chowdhuri's scan iterators are equivalent to Pike's map operators, and that Chowdhuri teaches scanning data groups of different schema to produce intermediate results. *Id.* ¶¶130-35, Figs. 3A-3B; EX-1005 ¶134. Chowdhuri's order and customer tables are data groups as discussed in element [1b]. These data groups have different schema in order data and customer data. Chowdhuri, ¶¶127, 131, 135, Figs. 3A-3B; EX-1005 ¶134. This is consistent with the '610 patent's employee and

department tables, which the patent contends are exemplary tables having different schema. EX-1001, 3:23-34, 3:65-4:13, Fig. 3.

A POSITA would have found it obvious to combine the teachings of Chowdhuri with Pike's system 300, as discussed above. EX-1005 ¶135. In Chowdhuri-Pike, Chowdhuri's order table and customer table would be partitioned and mapped in system 300 to output intermediate data identifiable to these tables and reduced by "state." Chowdhuri ¶¶126-27 (grouping data from tables "by state."), ¶131 (grouping by "state codes"). Combining Pike and Chowdhuri would have involved nothing more than combining two known techniques of mapping and reducing tables of data, with predictable results. EX-1005 ¶135. Pike and Chowdhuri disclose similar mapping and reducing operations, for similar purposes. *Id.* For example, Pike discloses "large-scale data processing in a parallel and distributed processing environment..." Pike, 1:20-26, 10:4-7, 2:17-41, 10:31-48. Chowdhuri similarly discloses "process[ing] large data sets for both local and distributed systems," and "efficiently partition[ing] data so that multiple subtasks or operations can be performed simultaneously." Chowdhuri ¶¶11, 14. Pike and Chowdhuri therefore suggest their systems are compatible.

A POSITA would have been motivated to combine Pike and Chowdhuri to join tables of different data or "schema," based on Chowdhuri's suggestion of doing so. EX-1005 ¶136. The result would have incorporated well-known table processing

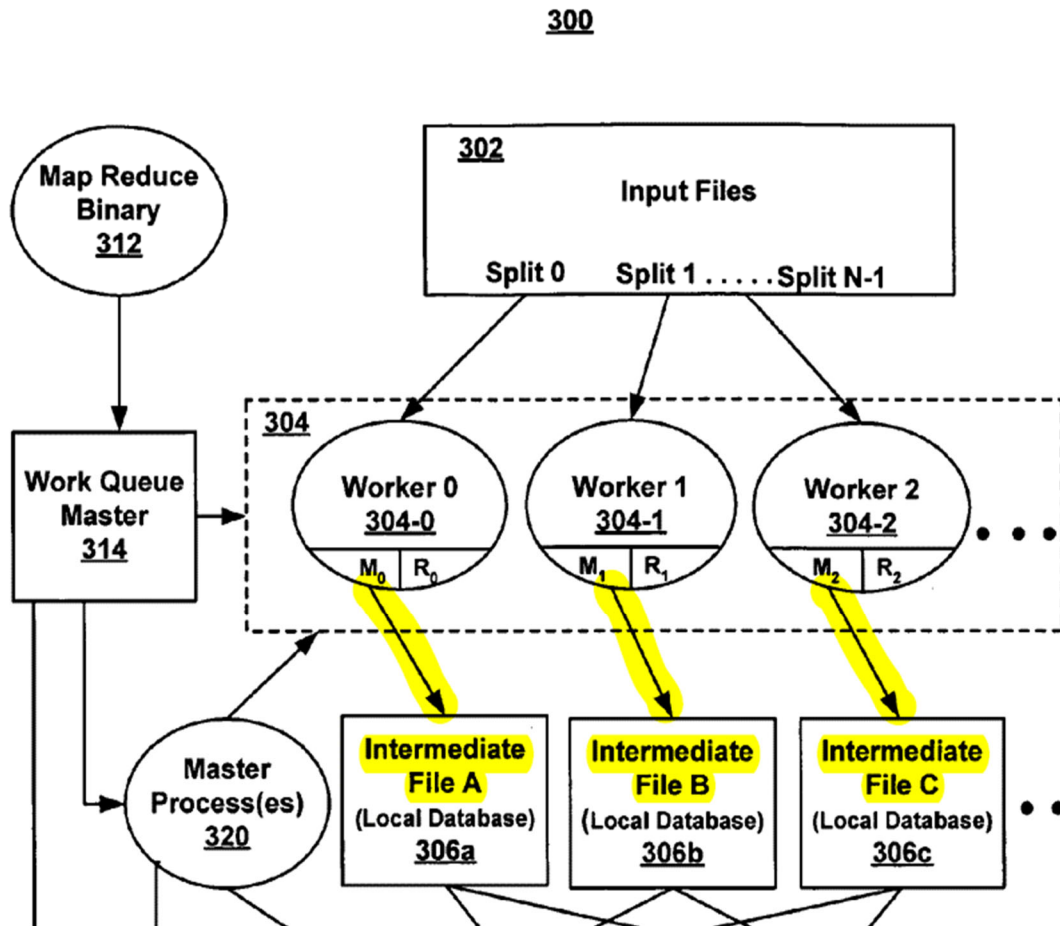
techniques in RDBMS and SQL queries, as taught by Chowdhuri, into MapReduce, based on Chowdhuri's suggestion of employing a similar set of serial map (e.g., scan) and reduce (e.g., merge) processes. *Id.* As discussed above, implementing Pike and Chowdhuri to track input table identifiers, as taught and suggested by Chowdhuri, would have enabled the output data from Pike's map and reduce processes to be associated with particular tables for subsequent postprocessing. *Id.* This would have allowed, for example, the omission of particular tables from the final merge result in the Pike-Chowdhuri system. *Id.* The predictable result would have been a MapReduce system that maps and reduces input tables and joins them according to prescribed keys, such as order "states," as Chowdhuri discloses. *Id.*

[1g] "and the data of the first data group is mapped differently..."

Patent Owner argued in district court that "mapped differently" does not require different map *functions* and the district court agreed. EX-1007, 16-18; EX-1005 ¶137.

Pike (**Grounds 1-2**) discloses or renders obvious this element. As shown in Figure 3 (below), Pike discloses or renders obvious mapping the data of its data groups "differently" by disclosing that each of its data blocks (e.g., "split[s]") are provided to a different worker process 304-0, 304-1, and 304-2, which in turn each invoke a map operator, M₀, M₁, and M₂, respectively. Pike, 9:10-12, 9:58-10:27, 10:4-13 ("Each of the worker processes 304 ... applies the application-specific map(

) operator to the respective input data block.”), 7:59-8:9 (“[T]he work queue master
[] ... create[s] a corresponding number of instances of the map process.”), 9:29-44;
EX-1005 ¶138.



Pike, Fig. 3 (annotated).

Pike additionally discloses that its worker processes 304 output “different lists of values” for “corresponding different intermediate data” by disclosing that each map operator “generate[s] intermediate data values” that are “collected and written to one or more intermediate files 306.” *Id.*, 10:4-19; EX-1005 ¶139. In certain embodiments, “each of the intermediate files 306 receives output from *only one*

worker process 304, as shown in FIG. 3.” Pike, 10:4-19, 14:57-15:6, Fig. 3. Pike’s disclosure of providing each data block (e.g., partition) to a different map operator, and outputting different intermediate data for each map operator to the intermediate files 306, is consistent with the disclosure of the ’610 patent, which depicts these features in Figures 4 and 5. EX-1001, Figs. 4, 5; EX-1005 ¶139.

To the extent “mapped differently” is construed to require invoking different map functions, Pike renders this feature obvious (**Grounds 1-2**). EX-1005 ¶140. As described in element [1f], Pike discloses that the map functions are application-specific and developed for specific use-cases. *See* Section V.A.[1f] (citing Pike, 2:22-26, 2:31-36, 3:59-64, 4:33-39, 6:1-12, 10:4-7, 10:39-50, 11:36-38, 14:19-27). Pike’s computer system 400 also includes application software 422, which includes “application-specific [map] functions for producing output.” Pike, 13:34-14:27, Fig. 4. This would have taught a POSITA that Pike contemplates multiple map functions in its application software, and that these map functions are invoked for processing particular data depending on the use-case, including invoking different map functions in the same map process (e.g., for different input files). EX-1005 ¶140. Indeed, Pike states this expressly by stating that “[g]enerally each ‘map process’ is a process configured (or configurable) to perform map *functions* and to execute an application-specific map operator.” Pike, 3:59-61.

Chowdhuri (**Ground 2**) also teaches mapping its data groups differently and outputting “different lists of values” for “corresponding different intermediate data.” Chowdhuri discloses scanning its customer and order tables with “two separate scan iterators” 301 and 302, or “clones” of these, which then produce output in the form of customer orders by state. Chowdhuri ¶¶126-28, 131-35, Fig. 3A, 3B. As shown in Figures 3A and 3B, each scan iterator individually outputs the results of the scan to be joined, grouped, and ordered by state and order amount. *Id.* As explained in elements [1b] and [1d], a POSITA would have understood that Chowdhuri’s scan iterator corresponds to the description of a map function in Pike and the ’610 patent. *See* Sections V.A.[1b], [1d]. A POSITA therefore would have understood that Chowdhuri’s scan iterators map its tables differently and output different lists of values into different intermediate data (e.g., customer and state data in one scan output, order and customer data in another). EX-1005 ¶141. Chowdhuri also renders these features obvious if “mapped differently” is construed to require invoking different map functions because the scan iterators 301 and 302 (and their clones) process each table using different functions—one looking for customer states and another looking for customer orders. EX-1005 ¶142. As explained in elements [1b]-[1f], a POSITA would have been motivated and found it obvious to combine Chowdhuri and Pike, with expected results. *Id.*; *See* Sections V.A.[1b]-[1f].

[1h] “wherein the different schema and corresponding different intermediate data have a key in common; and”

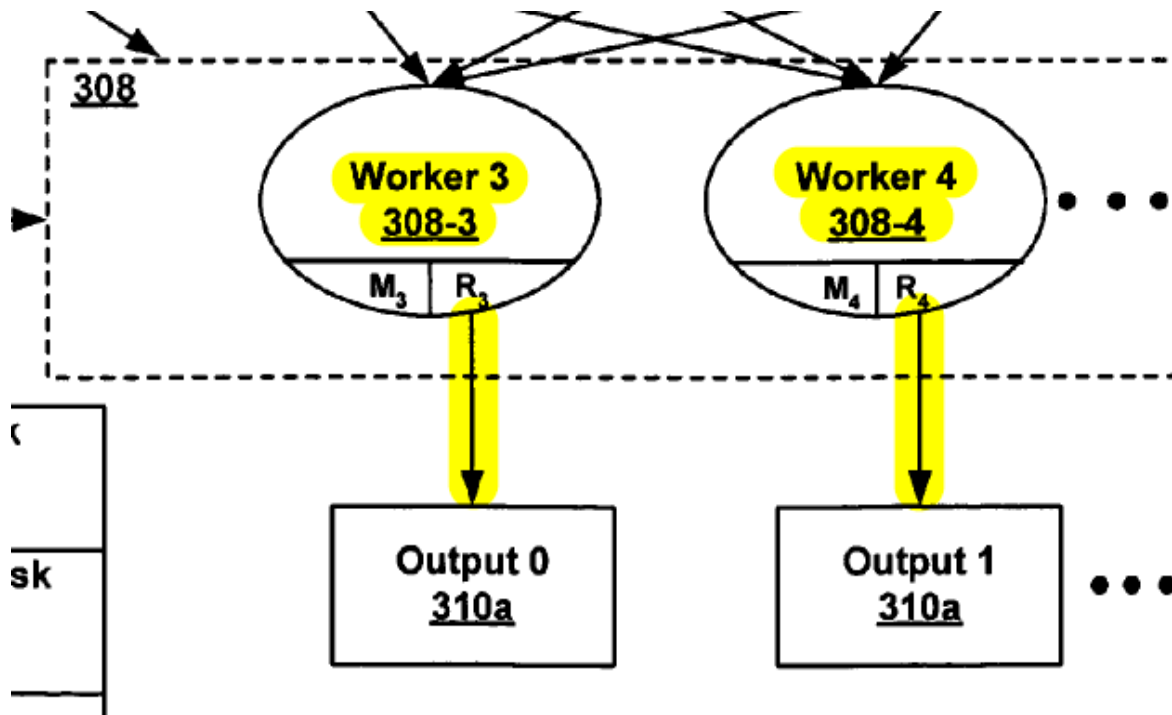
As explained above, Pike teaches or renders obvious “different schema” and “different intermediate data.” *See* Sections V.A.[1e], [1f], [1g]; EX-1005 ¶¶143-45. Pike additionally discloses or renders obvious having a key in common among them (**Grounds 1-2**). For example, Pike discloses that its worker process 308 invokes a reduce function that “merges or otherwise combines the sorted intermediate data values *having the same key*, and writes the key and combined values to one or more output files 310.” Pike, 11:28-38; 14:65-15:3. To reduce on “the same key,” the input data and intermediate data must have “a key in common.” EX-1005 ¶144. Having a common key among input data and intermediate data was also conventional in MapReduce. EX-1001, 1:17-27, 2:12-51 (common keys “k1 to k5”), Figs. 1-2.

Chowdhuri alternatively discloses this (**Ground 2**) by disclosing that its customer table and order table have a common key in the “customer_id,” which enables identifying customer “state” (i.e., within the United States) with customer orders. Chowdhuri ¶¶126-27 (“where o.customer_id=c.customer_id,” “group by state,” and sorting “orders for each of the states”), 131, 135; EX-1005 ¶145. As explained above, Chowdhuri’s customer and order tables have “different schema” and are mapped/scanned in the Pike-Chowdhuri system to result in “different intermediate data.” *See* Sections V.A.[1e], [1f], [1g]. For example, Pike’s mapping operators would map data blocks of customer table 308 and order table 309, and the

resulting key-value pairs would have the “customer_id” key in common. EX-1005 ¶145. This intermediate data would include the state for each customer and the order amount for each customer order so that the data could be reduced by Chowdhuri’s join, GroupBy, and sort iterators to merge orders by state, arranged by order amount. Chowdhuri ¶¶126-28, 131, 135. As explained in elements [1b]- [1f], a POSITA would have been motivated and found it obvious to combine Chowdhuri and Pike, with expected results. EX-1005 ¶45.

[1i] “reducing...,”

Pike discloses or renders these features obvious (**Grounds 1-2**). For example, Pike discloses that worker processes 308 invoke reduce operators to reduce the data from intermediate data files 306. Pike, 11:6-55, Fig. 3. Each worker process 308 “sorts the intermediate data values in the subset of the intermediate files read by that worker process *in accordance with the key* of the key/value pairs in the intermediate data.” *Id.* Each worker process 308 further “*merges or otherwise combines the sorted intermediate data values having the same key*, and writes the key and combined values to one or more output files 310.” *Id.*; EX-1005 ¶146. This teaches “reducing the intermediate data ... to at least one output data group, ... , so as to result in a merging of the corresponding different intermediate data based on the key in common.” EX-1005 ¶146.



Pike, Fig. 3 (annotated).

As explained in elements [1b] and [1f], Pike teaches or renders obvious “data groups.” *See* Sections V.A.[1b], [1f]. Pike’s worker processes 308 also render obvious processing intermediate data “for each data group in a manner that is defined to correspond to that data group” by disclosing that “[t]he merging or other combining operation performed on the sorted intermediate data is performed by an application-specific reduce() operator.” Pike, 11:28-55. A POSITA would have understood that Pike’s “application-specific” reducers are developed by programmers to be specific to a given use case and that processing the intermediate data based on the input file of Pike (or input table of Chowdhuri) was an obvious implementation of Pike’s application-specific data processing operators. EX-1005

¶148. This would have been well within the capabilities of a POSITA given the tracking of input files in Pike through the map and reduce phases. *See* Section V.A.[1e].

Pike-Chowdhuri (**Ground 2**) also renders this feature obvious by disclosing joining, grouping, and sorting by customer and order “state.” *See* Section V.A.[1h]. That is, Chowdhuri processes the results of the scanning iterators according to which table the data came from, processing the scanned results of the customer table to identify the state corresponding to each customer (i.e., “c.customer_id”), and processing the scanned results of the order table to identify the order amount by each customer (i.e., “o.customer_id”). Chowdhuri ¶¶126, 131, 135. Chowdhuri also explains that the hashjoin iterator “inherently has the notion of which data set to use for the build operation and what to use for the probe operation, normally designated by the left and the right operands respectively.” *Id.* ¶130. Chowdhuri’s reduce functions therefore understands the data set which they are operating on and process customer (“c”) data and order (“o”) data in a manner to correspond to Chowdhuri’s inputs: customer and order tables. EX-1005 ¶149.

As explained in element [1d], a POSITA would have understood that Chowdhuri’s hashjoin, GroupBy, and sort iterators teach or suggest techniques that correspond to Pike’s reduce functions. EX-1005 ¶150. Chowdhuri’s disclosure is indistinguishable from the reducing operations disclosed in the ’610 patent: “the

reduce function merges all intermediate data over all groups” and “[i]terators specific to a particular group may be employed to iterate over that group’s intermediate data.” EX-1001, 4:56-58, 5:36-59, 1:20-27 (“[t]he reduce operation ... combines elements”), 5:19-39 (sorting), 2:64-66 (“a sort and group-by-key task”), 5:48-6:7 (group iterator), 6:34-7:16 (“hashtable” iterators), 10:1-7 (“the reducers include a sort, group-by-key and combine task”); EX-1005 ¶150.

It would have been obvious to a POSITA to implement one or more of Chowdhuri’s hashjoin, GroupBy, and/or sort iterators as Pike’s reduce operators in Pike’s system 300. EX-1005 ¶151. A POSITA would have done so to merge data from Chowdhuri’s customer and order states, as Chowdhuri discloses and suggests. *Id.* Combining Chowdhuri’s hashjoin, GroupBy, and/or sort iterators with Pike’s system 300 would have yielded predictable result of a system that maps Chowdhuri’s partitioned input tables by scanning the data over a distributed system using multiple, parallel processes, and then reduces the scan outputs over the distributed system to affect a merging and sorting of customer and order data by state. *Id.* A POSITA would have been motivated to do so, as discussed above, to leverage Pike’s distributed system and fault tolerant task management. *Id.* ¶151-52.

[1j] “wherein the mapping and reducing operations are performed by a distributed system”

This feature was “conventional” in MapReduce. EX-1001, 1:6-27, 3:9-15. Pike’s large-scale data processing system 300 and computer system 400 also

disclose mapping and reducing operations being performed by a distributed system (**Grounds 1-2**). Section V.A.[1a]; Pike, 1:20-35, 2:43-49, 9:20-28, 13:37-50, 13:53-14:27; EX-1005 ¶153. So does Chowdhuri (**Ground 2**). Section V.A.[1a]; Chowdhuri ¶¶105-08, 128, 131-35; EX-1005 ¶153.

B. Claim 17

Pike and Chowdhuri disclose and render obvious claim 17 (**Grounds 1-2**) for reasons similar to those discussed for claim 1. Exemplary cross citations are provided below.

[17a] “A computer system...”

See Section V.A.[1a], V.A.[1j], V.A.[1b] (data groups); EX-1005 ¶154.

[17b] “partitioning...”

See Section V.A.[1c]; EX-1005 ¶155.

[17c] “providing...”

See Section V.A.[1d]; EX-1005 ¶156.

[17d] “to independently output...,”

See Section V.A.[1e]; EX-1005 ¶157.

[17e] “wherein the data...”

See Section V.A.[1f]; EX-1005 ¶158.

[17f] “and the data of the first data group...,”

See Section V.A.[1g]; EX-1005 ¶159.

[17g] “wherein the different schema...”

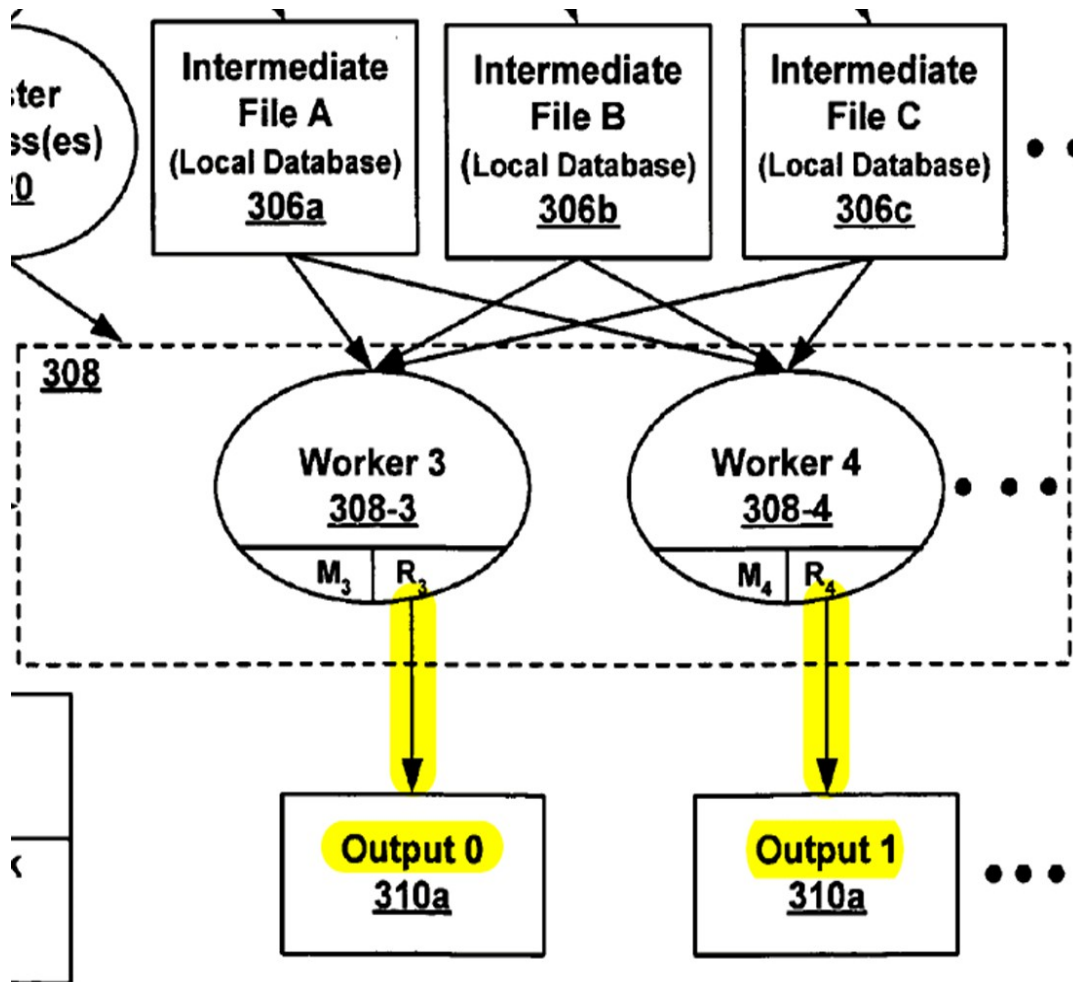
See Section V.A.[1h]; EX-1005 ¶160.

[17h] “reduce...”

See Section V.A.[1i]; EX-1005 ¶161.

C. Claims 2, 18

Pike (**Grounds 1-2**) discloses this feature because worker processes 308 reduces intermediate data 306 to “one *or more* output files 310.” Pike, 11:33-36, 11:6-55, 14:65-15:6, 6:65-7:12, Figs. 3, 7A; EX-1005 ¶162.



Pike, Fig. 3 (annotated). Chowdhuri alternatively discloses this (**Ground 2**) by disclosing more than one output from its scan iterators. Chowdhuri ¶¶132-35, Fig. 3B; EX-1005 ¶163. Pike and Chowdhuri’s output files are output data groups for the same reasons that Pike and Chowdhuri’s input files are “data groups.” See Section V.A.[1b].

D. Claims 3, 19

The patent states that “the operations can collect information about the data that they are processing” and “save the information as metadata.” EX-1001, 5:19-

32, 7:31-35. One example is saving *keys* associated with a partition in the intermediate data. *Id.*, 5:22-32; EX-1005 ¶164.

In the same manner, Pike (**Grounds 1-2**) discloses saving data associated with each partition's key in its intermediate files 306. EX-1005 ¶165. Worker processes 308 invoke reduce operators to “retriev[e] intermediate data values from the intermediate data files 306” and sort the same “in accordance with *the key*.” Pike, 11:8-14, 11:28-31. This enables worker process 308 to “merge[] or otherwise combine[] the sorted intermediate data values having the same key.” *Id.*, 11:33-36.

A POSITA would have understood Pike's processing each intermediate data files 306 in accordance with the key of that intermediate teaches or at least suggest storing keys “in each partition processed.” EX-1001, 5:22-24; EX-1005 ¶166. The key in the intermediate data is data about other data (e.g., data about the partition), which is encompassed by the plain meaning of “metadata.” EX-1001, 5:22-24; EX-1005 ¶166.

Pike's master process 320 tracking input files during its mapping and reducing phases, as well as associated task IDs, status, process IDs, and output files, also teaches “metadata ... [that] is identifiable to the data group.” Pike, 6:55-7:25, Fig. 7A. Master process 320 associates the input files with output files (which include intermediate files). *See* Section V.A.[1e]. This teaches metadata “identifiable to the data group.” EX-1005 ¶167.

Task Status Table					
Task ID	Status	Process	Input Files	Output Files	...
Map0000	Completed	P0000	12340	12341	...
Map0001	Completed	P0001	12344	12345	...
⋮	⋮				
Map0103	In Progress	P0010	12030	12031	...
Map0104	Failed	P0011	10101	12102	...
Map0103b	In Progress	P0010	12030	12031	...
⋮	⋮				
Red0000	In Progress	P0033	12340, ...	14000	...
Red0001	Waiting	-	-	-	...

Figure 7A

Pike, Fig. 7A (annotated).

To the extent argued otherwise, it would have been obvious to implement these features in Pike's intermediate data files 306 to include metadata about its keys. EX-1005 ¶168. Doing so would have been nothing more than the simple substitution of one known element (i.e., metadata) for another (i.e., intermediate data or status information in Pike's task status table) to obtain predictable results: storing this information as metadata in the intermediate data files or otherwise. *Id.* A POSITA would have known that using identifiers or metadata to track data in a system like Pike was well known. *Id.* (citing EX-1024, EX-1025). Worker process 308 would access the keys from metadata associated with the intermediate data to

reduce the intermediate data to output data, or master process 320 would access status information metadata to track and assign tasks. *Id.*

MacLeod also renders this feature obvious (**Ground 3**). MacLeod discloses a system “for tracking the lineage of data in a database.” EX-1024, Abstract, 1:12-59. In MacLeod, a “lineage identifier” traces the source of the data and can “act as a pointer to a detailed history files of operations that were performed on the data to transform it into its current form.” *Id.*, 1:60-2:23. The lineage identifier is “attached to data as it moves through systems thereby allowing the history and origin of data to be traced.” *Id.*, 6:54-7:14, 2:5-22, 7:46-61, Fig. 5.

A POSITA would understand that MacLeod’s lineage identifier teaches metadata that is identifiable to a data group because the identifier is data about data that identifies the origin of a file (e.g., a “data group” in Pike or Chowdhuri). A POSITA would have combined MacLeod’s lineage identifier with Pike and Chowdhuri to create data associated with intermediate data to track its lineage through the mapping and reducing. EX-1005 ¶¶169-70. A POSITA would have been motivated to do so, as discussed, to allow input files to be tracked and optionally removed from the reduce phase. *Id.*; *see* Section V.A.[1e] (“improv[ing] efficiency of the system.”). The result would have been predictable in attaching an identifier, or metadata, to the intermediate data for selective reducing. *Id.*

E. Claims, 4, 20

Pike (**Grounds 1-3**) and MacLeod (**Ground 3**) disclose or render obvious “intermediate data being stored in association with data that is identifiable to the data group” for the reasons it teaches or renders obvious “metadata.” *See* Section V.D; EX-1005 ¶171.

F. Claims 5, 21

Iterators are “useable within reduce functions *to access* intermediate data.” EX-1001, 3:58-62. Pike’s worker process 308 (**Grounds 1-2**) discloses an “iterator” that accesses data in intermediate data files 306 and applies a reduce operator to its values. Pike, 11:6-38 (“retrieving intermediate data values” and sorting and merging them “in accordance with the key”). Pike’s reducer operators are iterators because the inventors of Pike described them this way in the paper cited by the ’610 patent. EX-1009, 4 (“The reduce worker *iterates* over the sorted intermediate data...”); EX-1005 ¶¶172-73. Pike’s worker processes 308 “correspond[] to that data group” because they correspond to the input file, as explained in Section V.A.[1i]. EX-1005 ¶173.

Chowdhuri’s iterators also disclose this feature (**Ground 2**) by employing a hash join iterator, GroupBy iterator, and sort iterator for reducing the intermediate results of its scanning. Chowdhuri ¶¶129-35, 61, Figs. 3A-3B; *see also* Section V.A.[1f]; EX-1005 ¶174. It would have been obvious to a POSITA to combine

Chowdhuri and Pike as explained in Sections V.A.[1b]-[1f]. A POSITA would have been motivated to implement Pike's worker process 308 and reduce operators as an iterator because Pike suggests it, Chowdhuri discloses a mapping and reducing process using iterators, and as explained in Chowdhuri: "iterators ... can be easily duplicated for horizontal parallelism across CPU and I/O threads." Chowdhuri ¶131; EX-1005 ¶175. Implementing Pike's worker process 308 as an iterator would promote parallelism in system 300 and reduce run times. EX-1005 ¶175.

G. Claims 6, 23

[a] "wherein:...;"

Pike's intermediate data files 306 including key/value pairs discloses this feature (**Grounds 1-2**). Pike, 11:27-31, 7:34-52, 10:31-58, Fig. 3; EX-1005 ¶176. Chowdhuri also discloses this in the results of its scanning iterators, which include customer keys and values (customer state and order data). Chowdhuri ¶¶126-35, Figs. 3A-3B; EX-1005 ¶176.

Pike's "one or more worker processes 304 and 308" also discloses a reducing step carried out by a plurality of reducers (**Grounds 1-2**). Pike, 8:65-9:28, 11:6-55, 13:5-17, 14:65-15:6, Fig. 3; EX-1005 ¶177. Chowdhuri also discloses this in its hashjoin iterators 381-384, as well as its hashjoin, GroupBy, and sort iterators 303-305 (**Ground 2**). Chowdhuri ¶¶126-35, Figs. 3A-3B.

[b] “partition[ing]...”

Pike (**Grounds 1-2**) discloses this by its partition operator “specify[ing] that all intermediate data is to be directed to an intermediate file corresponding to the value of the first byte of the key.” Pike, 7:40-58, 14:9-27, 14:63-65, Fig. 4; EX-1005 ¶178. Pike further discloses that “[i]n some embodiments, each of the worker processes 308 reads a corresponding one of the intermediate files 306 produced by all or a subset of the worker processes 304.” Pike, 11:14-27. A POSITA would have understood that this teaches providing the intermediate data of each partition to a separate one of the reducers, as claimed. EX-1005 ¶179.

Pike at least renders this feature obvious. *Id.* ¶180. Pike discloses that the partition operator is application-specific and determined by the user and use-case. Pike, 14:12-27, 4:49-53. Implementing Pike so that each intermediate file includes only one group of key-value pairs, and providing each partition to a different reducer is one implementation a POSITA would have found obvious from Pike. EX-1005 ¶180. The ’610 patent also admits this feature was conventional. EX-1001, 4:63-67 (“similar to what is done by the conventional MapReduce”), 2:61-64 (partition functions 206a, 206b, 206c), Fig. 2.

Chowdhuri teaches or at least renders this feature obvious (**Ground 2**). Chowdhuri discloses that the customer and order table may be “repartitioned” after the scanning operation to employ “multiple devices” to join and then group via the

hashjoin and GroupBy iterators. Chowdhuri ¶128, Figs. 3A-3B. Chowdhuri also discloses employing exchange operators that “immediately repartition[] the data and redistribute[] it to [a] hash join operator,” including “redistribut[ing] of data to multiple servers on a cluster for processing the data.” *Id.* ¶¶132-34, Fig. 3B. A POSITA would have understood that this teaches or at least renders obvious providing intermediate data of each partition to a separate one of the reducers, including individual hashjoin iterators. EX-1005 ¶181.

H. Claims 7, 24

[a] “the intermediate data...; the reducing...”

See Section V.G.[a]; EX-1005 ¶182.

[b] “partition[ing]...; and provid[ing]...”

Claims 7 and 24 do not require “key/value pairs of the intermediate data” to *each* be provided to “more than one of the partitions” (i.e., providing the same key/value pair to more than one partition). The claims only recite providing at least some of the key/value pairs to more than one partition (e.g., providing two key/value pairs to two partitions). EX-1005 ¶184.

Pike’s partition operator discloses partitioning the intermediate data into a plurality of partitions and providing each partition to a reducer. *See* Section V.G.[b]; EX-1005 ¶183. Pike also discloses providing some of the key/value pairs to more than one partition by portioning by “a modulo function,” where the application

programmer “specifies the modulus.” Pike, 7:52-55. A POSITA would have understood that dividing the intermediate data based on a modulus, rather than dividing them by key, teaches providing key/value pairs of the intermediate data to multiple partitions. EX-1005 ¶185. This at least suggests providing some of the key/value pairs to multiple partitions and therefore renders this feature obvious. *Id.*

To the extent the same key/value pair must be sent to multiple partitions, this would have been an obvious implementation of MapReduce in view of Pike (**Grounds 1-2**). To provide a particular key-value pair to more than one partition, the map operation would simply emit the same key/value pair multiple times. EX-1005 ¶186. Repeating the same process twice is not new and would yield expected results: emitting the same key/value pair more than once, to provide each to a separate partition for reduction. *Id.* This is a straightforward application of a map function in “conventional MapReduce” that a POSITA would have found obvious to implement. *Id.*

This was also obvious based on Chowdhuri’s repartitioning and exchange operators 371, 372, as discussed in Section V.G.[b] (**Ground 2**). Chowdhuri ¶¶128-35, Fig. 3B; EX-1005 ¶187. Chowdhuri further discloses a “replicated join,” where “each tuple from a given partition on the outer operand will be sent to *every partition* of the inner operand, which means that if the same tuple encounters a match in more than one partition, only one of them needs to be selected.” Chowdhuri ¶227.

This teaches or at least suggests providing key/value pairs to more than one partition. EX-1005 ¶188. Implementing these teachings into Pike's system 300 would have been nothing more than the simple substitution of known features for predictable results: providing all or some of Pike's key/values to more than one partition. *Id.*

I. Claims 8, 25

Claims 8 and 25 do not require all of the key-value pairs to be provided to *each* of the partitions. Accordingly, Pike teaches this feature (**Grounds 1-2**) by disclosing that all of its key/value pairs are provided to its partitions. *See* Section V.G.[b]; EX-1005 ¶189. Chowdhuri alternatively renders it obvious (**Ground 2**). *See id.*

To the extent each key/value pair must be provided to each partition, Pike renders this feature obvious (**Grounds 1-2**). A POSITA would have understood that the partition operators of Pike are application-specific and that it would have been obvious to provide each key/value pair to each partition. EX-1005 ¶190. A POSITA would have been motivated to do so to increase fault tolerance. *Id.* Separate processors or machines would process the same key/value pairs and ensure the pairs are processed even if individual processors fail in the process. *Id.* Pike also depicts the data from its intermediate data files 306 being distributed to each worker process 308. Pike, Fig. 3. This renders obvious providing each key-value pair in the input data to all reducers. EX-1005 ¶191. A POSITA would have implemented Pike to

provide all key-value pairs to the reducers to account for all key-value pairs in the input data sets. *Id.* Indeed, Pike suggests this by disclosing that its reduce tasks may provide statistics about the key-value pairs in the input data sets. Pike, 11:38-55. The most accurate statistics account for all available data in the data sets, including all of the key-value pairs. EX-1005 ¶191.

J. Claims 9, 26

[a] “...at least some...;”

Pike discloses or renders obvious this feature, particularly in view of the conventional art and knowledge of a POSITA, as admitted by the '610 patent (**Grounds 1-2**). EX-1005 ¶192. Pike discloses a sort and combine task by worker process 308 “sort[ing] the intermediate data values” by key, Pike, 11:28-38, 12:7-17, 14:65-15:6, 9:45-49, and “merg[ing] or otherwise combin[ing] the sorted intermediate data values having the same key,” *id.*, 11:28-38. While Pike does not expressly disclose that its worker processes 308 include a “group-by-key,” the '610 patent admits this feature was “conventional” in MapReduce. EX-1001, 2:9-67, Fig. 1 (item 108 “Group-by-Key”), 2:64-67, Fig. 2 (item 210a, 210b). A POSITA would have known that Pike’s worker process 308 includes this MapReduce feature, or a POSITA would have known that it was conventional and found it obvious to implement in Pike. EX-1005 ¶193. A POSITA would have been motivated to include

a group-by-key reducer to make the processing of the output data “more efficient and/or convenient,” as the ’610 patent concedes. *Id.*; EX-1001, 2:64-67.

Pike-Chowdhuri (**Ground 2**) also renders this feature obvious. As explained in claim 1, Chowdhuri discloses reducing data using a hash join iterator 303, GroupBy iterator 304, and sort iterator 305. *See* Section V.A.[1f]; Chowdhuri, ¶¶127, 130. A POSITA would have combined Pike and Chowdhuri to implement Pike’s worker processes 308 as Chowdhuri’s iterators as explained in Sections V.A.[1a]-[1f]. EX-1005 ¶194.

[b] “...generating and providing...”

Pike’s storing “keys” in its intermediate data files 306 and its status table, teaches providing metadata during at least the mapping phase (**Grounds 1-2**). *See* Section V.D. So does MacLeod (**Ground 3**). *See* Section V.D; EX-1005 ¶195.

Pike also renders obvious providing metadata during the sorting and combining by disclosing that “[e]ach worker process 308 sorts the intermediate data values in the subset of the intermediate files,” Pike, 11:28-31, and “merges or otherwise combines the sorted intermediate data values” by *key*, *id.*, 11:33-38. EX-1005 ¶196.

Pike also renders this feature obvious by its master process 320 tracking associated input files during its mapping and reducing phases, as explained in Section V.D. *See also* Pike, 6:55-7:25, Fig. 7A; EX-1005 ¶197. Master process 320

associates the input files (e.g., “data groups”) with output files, including intermediate files. *See* Section V.A.[1e].

A POSITA would have understood that to sort and merge by key, Pike’s keys or task scheduling data would be “provid[ed]” to combining and sorting operations. EX-1005 ¶198. The metadata may also include MacLeod’s identifier that would identify the “data group” associated with the intermediate data for subsequent grouping or sorting. EX-1005 ¶199.

K. Claims 10, 27 – “...processing the metadata.”

Pike discloses this feature by the reducers sorting the values in the intermediate data files by key (**Grounds 1-3**). As explained for claims 3, 9, 19, and 26, Pike’s storing “keys” in its intermediate data files 306 teaches generating and providing metadata during the mapping and reducing. *See* Sections V.D, V.J.[b]; EX-1005 ¶200. During reducing, the worker process 308 “sorts the intermediate data values in the subset of the intermediate files read by that worker process in accordance with the key of the key/value pairs in the intermediate data.” Pike, 11:28-31. This teaches processing the intermediate data by key. EX-1005 ¶200. Pike’s merging the sorted intermediate data values by key also discloses this metadata processing. *Id.*; Pike, 11:33-38. Pike’s master process 320 processing status data when reducing is complete also teaches processing the metadata. Pike, 11:49-55; EX-1005 ¶200; *see also* Sections V.D. (citing Pike, 6:55-7:12, Fig. 7A), V.J.[b].

L. Claims 11, 28 – “processing...”

As explained in claims 5 and 21, Pike’s worker process 308 (**Grounds 1-3**), and Pike’s worker process 308 in view of Chowdhuri’s iterators (**Grounds 2-3**) disclose processing the intermediate data for each group in a manner that is defined to correspond to that data group by employing an iterator for each data group that corresponds to that data group. *See* Section V.F; EX-1005 ¶201.

As explained in claims 9, 10, 26, and 27, Pike (**Grounds 1-3**) further discloses providing metadata to the processing of the reducing step by providing keys of the intermediate data to the worker process 308 for sorting and merging, and providing status information associated with the reducing to master process 320. *See* Sections V.J.[b], V.K; EX-1005 ¶202.

M. Claims 12, 22

[a] “the intermediate data includes...” and “the reducing step...;”

See Section V.G.[a] (claim 6); EX-1005 ¶203.

[b] “for at least one of the reducers, ...”

Pike’s worker process 308 (**Grounds 1-2**), and Pike’s worker process 308 in view of Chowdhuri’s iterators (**Ground 2**), disclose processing the intermediate data for each group in a manner that is defined to correspond to that data group by employing an iterator for each data group. *See* Section V.F. Pike renders obvious

and Chowdhuri discloses that the data groups (i.e., input files) may have different schema. *See* Section V.A.[1f].

A POSITA would have understood that Pike also renders obvious using two iterators to sort different keys from intermediate data associated with two input files having different schema. EX-1005 ¶204. Pike discloses that its worker processes 308 “sort[] the intermediate data values in the subset of the intermediate files read by that worker process in accordance with the key of the key/value pairs in the intermediate data,” and that “sorting of the key/value pairs is an application independent function of the reduce threads in the worker processes 308.” Pike, 11:28-33. This suggests that Pike’s sorting is performed on multiple keys across the intermediate data. EX-1005 ¶205-06. Sorting multiple keys in MapReduce was also conventional, as the ’610 patent concedes in its “sort and group-by-key” tasks 210a, 201b. EX-1001, 2:9-67, 2:64-67, Fig. 2.

Chowdhuri (**Ground 2**) also discloses this in its hashjoin iterator, which joins Chowdhuri’s customer and order tables based on iterators specific to those inputs: based on customer state and order sum. Chowdhuri ¶¶126-35; EX-1005 ¶207. Indeed, Chowdhuri explains that the hashjoin iterator “inherently has the notion of which data set to use for the build operation and what to use for the probe operation.” Chowdhuri ¶130.

A POSITA would have found it obvious to implement Pike’s sort process, or Chowdhuri’s hashjoin iterators, to operate on different keys corresponding to different data groups. EX-1005 ¶208. Doing so would have been a straightforward combination of known elements in the prior art based on the suggestions of Pike and the teachings in Chowdhuri, where Chowdhuri discloses selectively joining data based on input data. *Id.* A POSITA would have been motivated to combine Pike and Chowdhuri for this feature, as discussed above, to selectively merge data from different input files, as Chowdhuri suggests. *Id.* The results would have been predictable. *Id.*

N. Claims 13, 29

The ’610 patent states that data that is not intermediate data “can come from data local to [a] reduce function or data shared by other reduce functions.” EX- 1001, 5:66-6:7. A POSITA would have understood that this data includes data received from sources other than map functions, including data provided by other reduce functions. EX-1005 ¶209.

Pike (**Grounds 1-2**) discloses this feature. Pike’s worker processes 308 receive information from master process 320 that “informs the worker processes 308 *where* to find intermediate data files 306 and schedul[ing] read requests for retrieving intermediate data values from the intermediate data files 306,” teaches processing data that is not intermediate data. Pike, 11:8-27. The data is received from

the master process 320 and not from a map function, and therefore is not intermediate data. EX-1005 ¶210. It is also processed by worker process 308 to locate intermediate data to reduce. *Id.*

Chowdhuri's GroupBy and sort iterators 304, 305 also teach processing data that is not intermediate data because these receive data from another reducing function: the hash join iterator 303 (**Ground 2**). Chowdhuri ¶¶127-35, Fig. 3A. As explained, a POSITA would have understood that Chowdhuri's GroupBy and sort iterators are reducing functions. *See* Sections V.A.[1f], V.B.[17e]. A POSITA would have understood these iterators receive data from another hash join reducing function, and that the received data is not intermediate data. EX-1005 ¶211.

O. Claims 14, 30

Pike-Chowdhuri (**Ground 2**) discloses these features. EX-1005 ¶212. Chowdhuri discloses a plurality of reducers as explained in claim 6, 13, 23, and 29. *See* Sections V.G.[a], V.N. As explained in claims 13 and 29, Chowdhuri discloses processing data that is not intermediate data and that is associated with another reducer by disclosing serial reduce functions: hashjoin, GroupBy, and sort iterators. *See id.*; Chowdhuri ¶¶127-35, Fig. 3A, 3B; *see also id.* ¶135 (“(Note that the GroupBy and Sort operators depicted in FIG. 3A are not included in FIG. 3B to simplify the diagram.).”). A POSITA would have combined Pike and Chowdhuri. *See* Sections V.A.[1b]-[1i]; EX-1005 ¶212.

P. Claims 15, 31

Pike (**Grounds 1-2**) discloses a plurality of reducers as explained in claims 6 and 23. *See* Section V.G.[a]. Pike also discloses processing data that is not intermediate data by “a particular reducer” and “associated with that reducer,” as explained in claims 13 and 29, by Pike’s worker processes 308 receiving information from master process 320 that “informs the worker processes 308 where to find intermediate data files 306 and schedul[ing] read requests for retrieving intermediate data values from the intermediate data files 306.” *See* Section V.N. The data is received from the master process 320 and not from a map function, and therefore is not intermediate data. EX-1005 ¶213. The data is also processed by worker process 308, which invokes a reduce function, and is data associated with that reduce function (e.g., where to locate intermediate data to reduce). *Id.*

Q. Claims 16, 32

Pike (**Grounds 1-2**) discloses this feature by its worker processes 308 invoking reducing operators to “merge[] or otherwise combine[]” intermediate data files 306. Pike, 4:40-44, 11:33-41. Merging or combining intermediate data across Pike’s input files (e.g., Pike’s “data groups”) teaches “relating the data among the plurality of data groups.” EX-1005 ¶214. This is consistent with the ’610 patent, which discloses “merg[ing] all intermediate data over all groups,” including data from employee and department tables. EX-1001, 4:50-59. Chowdhuri also discloses

this feature (**Ground 2**) by disclosing that its hashjoin, GroupBy, and sort iterators combine data from its customer and order tables. Chowdhuri ¶¶126-35; EX-1005 ¶214.

R. Claim 33

[33a] “A map-reduce...”

Pike’s system 300 discloses a “map-reduce method” because it maps and reduces data. Pike, 2:12-39, 2:50-3:35, 6:55-7:25, 8:65-11:55; EX-1005 ¶215. System 300 processes data from a plurality of groups over a computer system. *See* Sections V.A.[1a]-[1b], [1j]. Pike (**Grounds 1-2**) and Pike-Chowdhuri (**Ground 2**) renders obvious data groups of “different schema.” *See* Section V.A.[1f]. A POSITA would have combined Pike and Chowdhuri. *See* Sections V.A.[1a]-[1b].

[33b] “for a first data set...”

Pike (**Grounds 1-2**) and Pike-Chowdhuri (**Ground 2**) render this feature obvious. *See* Sections V.A.[1b], [1f]. A POSITA would have understood that both Pike and Chowdhuri disclose a “data set” with key-value pairs “belong[ing] to a ... data group.” EX-1005 ¶216; Section V.A.[1c]. Pike discloses input files. *See* Section V.A.[1b]. Chowdhuri discloses input tables. *See* Section V.A.[1f]. Pike also suggests and renders obvious input files of different schema, while Chowdhuri discloses this expressly. *See* Sections V.A.[1b], V.A.[1f]; EX-1005 ¶216. A POSITA would have combined Pike and Chowdhuri. *See* Sections V.A.[1a]-[1b].

[33c] “partitioning...”

Pike and Pike-Chowdhuri disclose and render this feature obvious. *See* Section V.A.[1c]. It was also conventional. *See id.*; EX-1005 ¶217.

[33d] “providing...”

Pike and Pike-Chowdhuri disclose and render this feature obvious. *See* Section V.A.[1d]. It was also conventional. *See id.*; EX-1005 ¶218.

[33e] “to independently output...”

Pike and Pike-Chowdhuri disclose and render this feature obvious. *See* Section V.A.[1e]. Pike and Pike-Chowdhuri also disclose that the intermediate data have a first set of resulting key-value pairs. *See* Section V.G.[a]; EX-1005 ¶219.

[33f] “for a second data set...”

Pike and Pike-Chowdhuri disclose and render this feature obvious. *See* Sections V.A.[1b]-[1c], [1f], V.R.[33b]. Chowdhuri’s order table and customer table are separate data groups with different key-value pairs and different schema. *See* Sections V.A.[1b], [1f]; EX-1005 ¶220.

[33g] “partitioning...”

This element recites partitioning the *first* data set (again). Whether this is understood to be a second partitioning of the first data set, or a single partition of the first data set into a plurality of data partitions, Pike’s splitting its input files (**Grounds 1-2**) or Chowdhuri’s splitting its input tables (**Ground 2**) renders this feature obvious. *See* Sections V.A.[1c], V.R.[33c]; EX-1005 ¶221.

For example, it would have been obvious to split a portion of Pike's input files or Chowdhuri's customer table into a first set of data blocks, and provide those data blocks to a first set of mapping operators to form first intermediate data, and then split another portion of Pike's input files or Chowdhuri's customer table to provide data blocks to a second set of mapping operators to form second intermediate data. EX-1005 ¶222. Pike suggests this by disclosing splitting an input into N-1 data blocks that can each be provided to different mapping functions. Pike, 9:65-10:15, Fig. 3.

To the extent it is argued that this feature requires partitioning the second data set, Pike's splitting of its input files, or splitting Chowdhuri's tables into data blocks in the Pike-Chowdhuri system, renders this feature obvious. *See* Sections V.R.[33c], [33f]; V.A.[1c]; EX-1005 ¶223.

[33h] “providing...”

Pike and Pike-Chowdhuri render this feature obvious. *See* Sections V.A.[1d]-[1e], V.G.[a](claim 6); EX-1005 ¶224.

Forming “differing” sets of resulting key-value pairs is satisfied by mapping functions that each output a different number of key-value pairs. EX-1005 ¶225. Finding a particular word once in one partition, for example, and two times in another partition, satisfies this limitation. *Id.* Pike discloses or at least suggests this by applying an “application-specific map() operator to [a] respective input data

block so as [to] generate intermediate data values,” and mapping “every word *w* in every document in the collection, where the key/value pair is $\langle w, 1 \rangle$.” Pike, 9:56-10:15, 10:31-43. Pike’s mapping operators would output particular numbers of the same key for different data blocks, which would differ between blocks. EX-1005 ¶225.

To the extent this element requires mapping using different keys, Pike (**Grounds 1-2**) discloses or renders this obvious by mapping different words in a document. Pike, 9:56-10:15, 10:31-43. Mapping different words teaches or at least suggests forming first and second intermediate data having differing first and second key-value pairs, where different words are different keys. Pike’s outputting intermediate files 304 also teaches forming first and second intermediate data sets. *Id.*, 9:65-10:19, Fig. 3; EX-1005 ¶226.

Chowdhuri (**Ground 2**) also teaches this by disclosing its scan iterators map customer and order tables based on `customer_id`, order amount, and state keys. Chowdhuri ¶¶126-35; EX-1005 ¶227. Chowdhuri’s outputting a plurality of intermediate results from its scanning iterators also teaches forming first and second intermediate data sets. Chowdhuri ¶135, Fig. 3B; EX-1005 ¶227. A POSITA would have combined Pike and Chowdhuri with predictable results. *See* Sections V.A.[1b]-[1h].

[33i] “wherein the first schema differs from the second schema,”

Pike (**Grounds 1-2**) and Pike-Chowdhuri (**Ground 2**) render this feature obvious. *See* Sections V.R.[33a], V.A.[1f]. Chowdhuri’s order table and customer table are separate data groups with different key-value pairs and different schema. *See* Sections V.A.[1b], V.A.[1f]; EX-1005 ¶228.

[33j] “wherein the first and second schema...;”

Pike (**Grounds 1-2**) and Pike-Chowdhuri (**Ground 2**) render this feature obvious. *See* Section V.A.[1h]. For example, Pike’s mapping operators forming resulting key-value pairs based on a single word teaches this feature. *See* Section V.R.[33h]; EX-1005 ¶229.

Chowdhuri alternatively discloses this by disclosing a customer and order table and its resulting intermediate data having a common key: “customer_id.” Chowdhuri ¶¶126-27, 131; *see* Sections V.A.[1f], [1h]. Pike’s mapping operators in the Pike-Chowdhuri system would map data blocks of the customer table (or data blocks in both customer and order tables), and the resulting key-value pairs would have a “customer_id” in common. EX-1005 ¶230.

[33k] “reducing...”

Pike (**Grounds 1-2**) discloses this feature as explained in Section V.A.[1i]. Pike and Chowdhuri (**Ground 2**) also disclose reducing by iterating on a key from the intermediate data. *See* Section V.F. *See also* Pike, 11:6-38; Chowdhuri

¶¶129-35; EX-1005 ¶231. In the Pike-Chowdhuri system, a POSITA would have understood that iterating by a specific word or by customer, state, or order amount teaches iterating on at least one key from the first and second intermediate data set. *See* Section V.R.[33h]; EX-1005 ¶231.

[33l] “the output data set...”

Pike (**Grounds 1-2**) and Pike-Chowdhuri (**Ground 2**) render this feature obvious.

For example, a POSITA would have understood that Pike’s word count in a collection of documents output from its reducing would have a different format—or “schema”—than the input documents. EX-1005 ¶232. A POSITA at least would have found this obvious based on a POSITA’s knowledge of MapReduce, which describes counting words in documents that are webpages. EX-1009, 1-3; EX-1005 ¶232. A POSITA would have recognized that a word count would have a different schema than a website. EX-1005 ¶232.

The Pike-Chowdhuri system further renders this feature obvious by outputting a merged list of customers and orders based on a common customer to provide orders by state. *See* Section V.A.[1i], V.R.[33h]. Chowdhuri’s merged customer-order list discloses an output with a different schema relative to the input tables, which would contain other fields than just customer names, orders, and states. EX-1005 ¶233.

This is consistent with the '610 patent's merging tables into a combined department-employee table. EX-1001, 3:23-34, Figs. 4, 5.

[33m] “wherein...”

Pike and Pike-Chowdhuri disclose and render this feature obvious as explained in Section V.A.[1j]. EX-1005 ¶234.

S. Claim 40

[40a] “A computer system...”

Pike (**Grounds 1-2**) and Pike-Chowdhuri (**Ground 2**) a computer system with a plurality of computing devices with at least one processor and memory. *See* Sections V.A.[1a], V.B.[17a]. Pike (**Grounds 1-2**) and Pike-Chowdhuri (**Ground 2**) disclose and render obvious processing data of a data set that comprises a plurality of data groups. *See* Sections V.A.[1a]-[1b]. A POSITA would have combined Pike and Chowdhuri. *See id.*; EX-1005 ¶235.

[40b]-[40l]

Elements [40b]-[40l] have counterparts in elements [33b]-[33l] and would have been obvious in view of Pike or Pike-Chowdhuri as discussed above regarding those counterpart elements. *See* Sections V.R.[33b]-[33i], [33k]-[33m]; EX-1005 ¶236-46.

T. Claims 34, 41

Reducing the intermediate data based on the common key of the intermediate data to merge the first and second intermediate data satisfies this limitation.

Pike (**Grounds 1-2**) and Pike-Chowdhuri (**Ground 2**) render this feature obvious. *See* Sections V.A.[1i], V.R.[33j], V.R.[33k]. EX-1005 ¶247.

U. Claims 35, 42

Pike (**Grounds 1-3**) and Pike-Chowdhuri (**Ground 2**) disclose this as explained in claims 7 and 24. *See* Section V.H; EX-1005 ¶248.

V. Claims 36, 43

These claims do not recite providing all key-value pairs to *each* reducer, only that the key-value pairs are all provided to reducers. Accordingly, Pike (**Grounds 1-2**) and Chowdhuri (**Ground 2**) teach this feature by disclosing that all key/value pairs are provided to partitions, which are reduced by worker processes 308 in Pike and hashjoin iterators in Chowdhuri. *See* Sections V.G.[b], V.I; EX-1005 ¶249.

Even if all key-values must be provided to each partition (and thus each reducer), Pike (**Grounds 1-2**) and Chowdhuri (**Ground 2**) render this feature obvious. *See* Section V.I; EX-1005 ¶250-51.

W. Claims 37, 44

Pike (**Grounds 1-3**) and Chowdhuri (**Grounds 2-3**) teach and render this feature obvious. *See* Section V.J; EX-1005 ¶252.

X. Claims 38, 45

Pike (**Grounds 1-3**) and Chowdhuri (**Grounds 2-3**) teach and render this feature obvious. *See* Section V.K; EX-1005 ¶253.

Y. Claims 39, 46

Pike (**Grounds 1-3**) and Chowdhuri (**Grounds 2-3**) teach and render this feature obvious. *See* Section V.L; EX-1005 ¶254.

VI. MANDATORY NOTICES UNDER 37 C.F.R. §42.8

A. Real Party-in-Interest

The real party-in-interest for this Petition is Databricks, Inc. 37 C.F.R. §42.8.

B. Related Matters

To the best knowledge of Petitioner, the '610 patent has been involved in the following judicial matters. Only the first three matters below are pending. The remaining matters have been terminated, including the *American Airlines* and *FedEx* IPR proceedings which were both terminated prior to any decision on institution.

The '610 Patent is currently the subject of *Cloudera, Inc. v. R2 Solutions LLC*, Inter Partes Review No. IPR2024-00303 (P.T.A.B.) (filed Dec. 18, 2023). This Petition is substantively identical to the petition submitted in IPR2024-00303 and Petitioner has concurrently filed a motion to join that proceeding.

Name	No.	Court	Filed
<i>R2 Solutions LLC v. Databricks, Inc.</i>	4:23-cv-01147	E.D. Tex.	Dec. 28, 2023
<i>Cloudera, Inc. v. R2 Solutions LLC</i>	IPR2024-00303	PTAB	Dec. 18, 2023
<i>R2 Solutions LLC v. Cloudera, Inc.</i>	1-23-cv-01205	W.D. Tex.	Oct. 5, 2023

Petition for *Inter Partes* Review of
U.S. Patent No. 8,190,610

<i>American Airlines, Inc. v. R2 Solutions LLC</i>	IPR2023-00689	PTAB	Mar. 7, 2023
<i>R2 Solutions LLC v. Hilton Domestic Operating Co. Inc.</i>	3-22-cv-02761	N.D. Tex.	Dec. 12, 2022
<i>FedEx Corporate Services, Inc. v. R2 Solutions LLC</i>	IPR2022-01405	PTAB	Aug. 15, 2022
<i>Allegiant Travel Company v. R2 Solutions LLC</i>	2-22-cv-00828	D. Nev.	May. 24, 2022
<i>R2 Solutions LLC v. American Airlines, Inc.</i>	4-22-cv-00353	E.D. Tex.	Apr. 28, 2022
<i>R2 Solutions LLC v. CVS Health Corporation et al</i>	4-22-cv-00354	E.D. Tex.	Apr. 28, 2022
<i>R2 Solutions LLC v. Hilton Worldwide Holdings Inc. et al</i>	4-22-cv-00356	E.D. Tex.	Apr. 28, 2022
<i>R2 Solutions LLC v. Citigroup Inc.</i>	4-22-cv-00357	E.D. Tex.	Apr. 28, 2022
<i>R2 Solutions LLC v. FedEx Corporate Services, Inc.</i>	4-21-cv-00940	E.D. Tex.	Nov. 29, 2021
<i>R2 Solutions LLC v. State Farm Mutual Automobile Insurance Company</i>	4-21-cv-00941	E.D. Tex.	Nov. 29, 2021
<i>R2 Solutions LLC v. Booking.com BV</i>	4-21-cv-00942	E.D. Tex.	Nov. 29, 2021
<i>R2 Solutions LLC v. Booking.com Transport Limited</i>	4-21-cv-00943	E.D. Tex.	Nov. 29, 2021
<i>R2 Solutions LLC v. Agoda Company Pte. Ltd.</i>	4-21-cv-00944	E.D. Tex.	Nov. 29, 2021

Petition for *Inter Partes* Review of
U.S. Patent No. 8,190,610

<i>R2 Solutions LLC v. Expedia Group, Inc.</i>	6-21-cv-00628	W.D. Tex.	Jun. 17, 2021
<i>R2 Solutions LLC v. iHeartMedia, Inc.</i>	6-21-cv-00552	W.D. Tex.	Jun. 01, 2021
<i>R2 Solutions LLC v. JPMorgan Chase & Co.</i>	4-21-cv-00174	E.D. Tex.	Mar. 02, 2021
<i>R2 Solutions LLC v. The Charles Schwab Corporation</i>	4-21-cv-00122	E.D. Tex.	Feb. 09, 2021
<i>R2 Solutions LLC v. Fidelity Brokerage Services LLC</i>	4-21-cv-00123	E.D. Tex.	Feb. 09, 2021
<i>R2 Solutions LLC v. Deezer SA</i>	4-21-cv-00090	E.D. Tex.	Jan. 29, 2021
<i>R2 Solutions LLC v. Walmart Inc.</i>	4-21-cv-00091	E.D. Tex.	Jan. 29, 2021
<i>R2 Solutions LLC v. Target Corporation</i>	4-21-cv-00092	E.D. Tex.	Jan. 29, 2021
<i>R2 Solutions LLC v. Workday, Inc.</i>	4-21-cv-00093	E.D. Tex.	Jan. 29, 2021

C. Lead and Backup Counsel

Petitioner provides the following designation and service information for lead and back-up counsel. 37 C.F.R. § 42.8(b)(3) and (b)(4).

Lead Counsel	Back-Up Counsel
Dargaye Churnet (Reg. No. 71,288) dchurnet@fenwick.com Fenwick & West LLP 555 California Street, Fl. 12 San Francisco, CA 91404 Tel: (415) 875-2300 Fax: (415) 281-1350	Michael Sacksteder (<i>pro hac vice</i> forthcoming) msacksteder@fenwick.com Fenwick & West LLP 555 California Street, Fl. 12 San Francisco, CA 91404 Tel: (415) 875-2300 Fax: (415) 281-1350 Vigen Salmastlian (<i>pro hac vice</i> forthcoming) vsalmastlian@fenwick.com Fenwick & West LLP 801 California Street Mountain View, CA 94041 Tel: (650) 988-8500 Fax: (650) 938-5200

D. Service Information

Please address all correspondence to lead and backup counsel at the addresses shown above. Petitioner also consents to electronic service by e-mail at r2vdatabricks@fenwick.com. A Power of Attorney is being filed concurrently with the designation of counsel in accordance with 37 C.F.R. § 42.10(b).

VII. GROUNDS FOR STANDING

Petitioner certifies that the '610 patent is available for *inter partes* review, and Petitioner is not barred or estopped from requesting *inter partes* review to challenge the claims on the grounds identified herein. Petitioner filed this Petition within one year of service of the original complaint against Petitioner in the district court litigation.

VIII. CONCLUSION

Petitioner requests institution of *inter partes* review and cancellation of all claims as unpatentable.

Dated: March 6, 2024

Respectfully submitted,

/Dargaye Churnet/

Dargaye Churnet

Reg. No. 71,288

Attorneys for Petitioner

Databricks, Inc.

CERTIFICATION OF WORD COUNT

The undersigned certifies pursuant to 37 C.F.R. §42.24 that the foregoing Petition for *Inter Partes* Review, excluding any table of contents, mandatory notices under 37 C.F.R. §42.8, certificates of service or word count, or appendix of exhibits, contains 13,899 words according to the word-processing program used to prepare this document (Microsoft Word).

Dated: March 24, 2024

/Dargaye Churnet/

Dargaye Churnet

Reg. No. 71,288

Attorneys for Petitioner

Databricks, Inc.

.

**CERTIFICATE OF SERVICE ON PATENT OWNER
UNDER 37 C.F.R. § 42.105**

I hereby certify, pursuant to 37 C.F.R. §§ 42.6(e) and 42.105, that a complete copy of this Petition for *Inter Partes* Review, including all exhibits, is being served via U.S. Express Mail on March 24, 2024, upon Patent Owner by serving the correspondence address of record with the USPTO as follows:

Weaver Austin Villeneuve & Sampson
P.O. Box 70250
Oakland, CA 94612-0250

Further, a courtesy copy of this Petition for *Inter Partes* Review, including all exhibits, is being sent via Federal Express to Patent Owner's litigation counsel:

Edward R. Nelson, III
Brent N. Bumgardner
Carder W. Brooks
Christopher G. Granaghan
John P. Murphy
Nelson Bumgardner Conroy PC
3131 W. 7th Street, Suite 300
Fort Worth, TX 76107
ed@nelbum.com
brent@nelbum.com
carder@nelbum.com
chris@nelbum.com
murphy@nelbum.com

Dated: March 24, 2024

/Dargaye Churnet/

Dargaye Churnet
Reg. No. 71,288

Attorneys for Petitioner
Databricks, Inc.