

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

Cloudera, Inc.,
Petitioner,

v.

R2 Solutions LLC,
Patent Owner.

Case No. IPR2024-00303

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

**DECLARATION OF JIMMY LIN, PH.D.
IN SUPPORT OF PETITION FOR INTER PARTES REVIEW OF
U.S. PATENT NO. 8,190,610**

TABLE OF CONTENTS

I.	Introduction.....	1
II.	Qualifications and Background	1
A.	Education and Qualifications	2
III.	Materials Considered	5
IV.	Legal Standards	6
A.	Claim Construction	7
B.	Obviousness Under 35 U.S.C. § 103.....	8
C.	Secondary Considerations of Non-Obviousness.....	10
V.	The '610 Patent.....	12
A.	Overview of MapReduce	12
B.	State of the Art of Relational Processing	18
C.	Overview of the '610 Patent.....	26
D.	Summary of the Prosecution History of the '610 Patent	31
E.	Claim Construction of Terms in the '610 Patent	32
VI.	Person of Ordinary Skill in the Art.....	33
VII.	The Prior Art.....	34
A.	Overview of Pike.....	34
B.	Overview of Chowdhuri.....	38
C.	Overview of MacLeod.....	41
VIII.	Opinions on Unpatentability.....	42
A.	Pike (Ground 1) and Pike in View of Chowdhuri (Ground 2) Render Claims 1-46 Obvious; Pike and Chowdhuri in view of	

MacLeod (Ground 3) Renders Claims 3, 4, 9-11, 19, 26, 27, 37-39 Obvious	43
1. Claim 1	43
a. [1a] “A method of processing data of a data set over a distributed system,”	43
b. [1b] “wherein the data set comprises a plurality of data groups,”	45
c. [1c] “the method comprising: partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and”	51
d. [1d] “providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable”	54
e. [1e] “to independently output a plurality of lists of values for each of a set of keys found in such map function’s corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,”	59
f. [1f] “wherein the data of a first data group has a different schema than the data of a second data group,”	67
g. [1g] “and the data of the first data group is mapped differently than the data of the second data group so that different lists of values are output for the corresponding different intermediate data,”	75
h. [1h] “wherein the different schema and corresponding different intermediate data have a key in common; and”	79
i. [1i] “reducing the intermediate data for the data groups to at least one output data group, including processing the intermediate data for each data	

	group in a manner that is defined to correspond to that data group, so as to result in a merging of the corresponding different intermediate data based on the key in common,”	81
j.	[1j] “wherein the mapping and reducing operations are performed by a distributed system.”	85
2.	Claim 17	86
a.	[17a] “A computer system including a plurality of computing devices, the computer system configured to process data of a data set, wherein the data set comprises a plurality of data groups, the computer system comprises at least one processor and memory that are operable to perform the following operations:”	86
b.	[17b] “partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and”	86
c.	[17c] “providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable”	87
d.	[17d] “to independently output a plurality of lists of values for each of a set of keys found in such map function’s corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,”	87
e.	[17e] “wherein the data of a first data group has a different schema than the data of a second data group”	87
f.	[17f] “and the data of the first data group is mapped differently than the data of the second data group so that different lists of values are output for the corresponding different intermediate data,”	87

g.	[17g] “wherein the different schema and corresponding different intermediate data have a key in common; and,”	88
h.	[17h] “reduce the intermediate data for the data groups to at least one output data group, including processing the intermediate data for each data group in a manner that is defined to correspond to that data group so as to result in a merging of the corresponding different intermediate data based on the key in common.”	88
3.	Claims 2 and 18 — “wherein: the at least one output data group is a plurality of output data groups.”	88
4.	Claims 3 and 19 — “wherein: corresponding intermediate data for a data group being identifiable to the data group includes the corresponding intermediate data being stored such that metadata associated with a manner of storage is identifiable to the data group.”	89
5.	Claims 4 and 20 — “store[] [the corresponding intermediate data] in association with data that is identifiable to the data group.”	93
6.	Claims 5 and 21 — “process[ing] the intermediate data” “includes, for each data group,” “employ[ing] an iterator that corresponds to that data group.”	93
7.	Claims 6 and 23.....	95
a.	[a] “wherein: the intermediate data includes a plurality of grouped sets of key/value pairs” and reducing via “a plurality of reducers;”	95
b.	[b] “partition[ing] the intermediate data into a plurality of partitions, [to provide] each key/value pair of the intermediate data . . . to a separate one of the partitions; and provid[ing] the intermediate data of each partition to a separate one of the reducers.”	96

8.	Claims 7 and 24.....	98
a.	[a] “the intermediate data includes a plurality of grouped sets of key/value pairs; the reducing step is carried out by a plurality of reducers;”	98
b.	[b] “partition[ing] the intermediate data into a plurality of partitions, [to provide] at least some of the key/value pairs of the intermediate data . . . to more than one of the partitions; and provid[ing] the intermediate data of each partition to a separate one of the reducers.”	98
9.	Claims 8 and 25 — “The method of claim 7, wherein: in the partitioning step all of the key/value pairs of the intermediate data are provided to all of the partitions.”	101
10.	Claims 9 and 26.....	103
a.	[a] “wherein: at least some of the reducers include a sort, group-by-key and combine task;”	103
b.	[b] “generating and providing metadata for at least some of the mapping, partitioning, combining, grouping and sorting.”	104
11.	Claims 10 and 27 — “wherein: the reducing [step] includes processing the metadata.”	105
12.	Claims 11 and 28 — “wherein: . . . process[ing] the intermediate data for each data group in a manner that is defined to correspond to that data group includes, for each data group, . . . employ[ing] an iterator that corresponds to that data group, wherein the iterator . . . provid[es] the associated metadata to the [processing of the] reducing [step].”	106
13.	Claims 12 and 22.....	107
a.	[b] “wherein: the intermediate data includes a plurality of grouped sets of key/value pairs” and reducing step via “a plurality of reducers;”	107

- b. [b] “for at least one of the reducers, the iterator corresponding to a particular data group, for that reducer, operates according to a different key of a different schema than the iterator corresponding to another particular data group, for that reducer.” 107
- 14. Claims 13 and 29 — “wherein: the intermediate data processing step of the reducing step further comprises processing data that is not intermediate data.” 110
- 15. Claims 14 and 30 — wherein reducing is carried out via “a plurality of reducers,” and “the data that is not intermediate data, for a particular reducer, includes data that is associated with another reducer.” 111
- 16. Claims 15 and 31 — wherein reducing is carried out via “a plurality of reducers,” and “the data that is not intermediate data, for a particular reducer, includes data that is associated with that reducer.” 112
- 17. Claims 16 and 32 — reducing by “relating the data among the plurality of data groups.” 112
- 18. Claim 33 113
 - a. [33a] “A map-reduce method of processing data from a plurality of groups having different schema over a computer system, the method comprising:” 113
 - b. [33b] “for a first data set having a plurality of first key-value pairs, wherein such first data set belongs to a first data group and the first key-value pairs have a first schema” 114
 - c. [33c] “partitioning the first data set into a plurality of data partitions and” 114
 - d. [33d] “providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable” 115

e.	[33e] “to independently output a plurality of lists of values for each of a set of keys found in such map function’s corresponding data partition to form a first intermediate data set having a first set of resulting key-value pairs;”	115
f.	[33f] “for a second data set having a plurality of second key-value pairs, wherein such second data set belongs to a second data group and the second key-value pairs have a second schema,”	115
g.	[33g] “partitioning the first data set into a plurality of data partitions and”	116
h.	[33h] “providing each data partition to a selected one of a plurality of mapping functions to form a second intermediate data set having a second set of resulting key-value pairs differing from the first set of resulting key-value pairs,”	117
i.	[33i] “wherein the first schema differs from the second schema,”	118
j.	[33j] “wherein the first and second schema and the first and second set of resulting key-value pairs have a key in common; and”	119
k.	[33k] “reducing the first and the second intermediate data set together so as to form an output data set, wherein the reducing is accomplished by iterating on at least one key from the first and second intermediate data set, and”	119
l.	[33l] “the output data set has a different schema than the first and second schema,”	120
m.	[33m] “wherein the mapping and reducing operations are performed by a distributed system.”	121
19.	Claim 40	121

- a. [40a] “A computer system including a plurality of computing devices, the computer system configured to process data of a data set, wherein the data set comprises a plurality of data groups, the computer system comprises at least one processor and memory that are operable to perform the following operations:” 121
- b. [40b] “for a first data set having a plurality of first key-value pairs, wherein such first data set belongs to a first data group and the first key-value pairs have a first schema,” 122
- c. [40c] “partitioning the first data set into a plurality of data partitions and” 122
- d. [40d] “providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable” 122
- e. [40e] “to independently output a plurality of lists of values for each of a set of keys found in such map function’s corresponding data partition to form a first intermediate data set having a first set of resulting key value pairs;” 123
- f. [40f] “for a second data set having a plurality of second key-value pairs, wherein such second data set belongs to a second data group and the second key-value pairs have a second schema,” 123
- g. [40g] “partitioning the first data set into a plurality of data partitions and” 123
- h. [40h] “providing each data partition to a selected one of a plurality of mapping functions to form a second intermediate data set having a second set of resulting key-value pairs differing from the first set of resulting key-value pairs,” 124
- i. [40i] “wherein the first schema differs from the second schema; and” 124

- j. [40j] “reducing the first and the second intermediate data set together so as to form an output data set, wherein the reducing is accomplished by iterating on at least one key from the first and second intermediate data set” 124
- k. [40k] “and the output data set has a different schema than the first and second schema,” 124
- l. [40l] “wherein the first and second schema and the first and second set of resulting key-value pairs have a key in common.” 125
- 20. Claims 34 and 41 — reducing by “iterating on at least one key from each of the first and second intermediate data set so that the output data set is a merging of a portion of the first and second intermediate data set.” 125
- 21. Claims 35 and 42 — “wherein the reducing is accomplished by a plurality of reducers,” “partitioning the first or second intermediate data set into a plurality of partitions, at least some of the key-value pairs of the first or second intermediate data being provided to more than one of the partitions,” and “providing the partitioned intermediate data set of each partition to a separate one of the reducers.” 125
- 22. Claims 36 and 43 — “wherein all of the key-value pairs of the first and second intermediate data set are provided to all of the reducers.” 126
- 23. Claims 37 and 44 — “wherein at least some of the reducers include a sort, group-by-key and combine task,” and “generating and providing metadata for at least some of the mapping, partitioning, combining, grouping and sorting.” 128
- 24. Claims 38 and 45 — “wherein the reducing includes processing the metadata.” 128

25.	Claims 39 and 46 — “wherein iterating includes providing the associated metadata to the processing of the reducing step.”	128
IX.	Conclusion	129

I. INTRODUCTION

1. I, Dr. Jimmy Lin, submit this declaration to state my opinions on the matters described below.

2. I have been retained by Petitioner, Cloudera, Inc. (“Petitioner” or “Cloudera”), as an independent expert in this proceeding before the United States Patent and Trademark Office. Although I am being compensated at my usual and customary rate of \$1,100 USD per hour, no part of my compensation depends on the outcome of this proceeding, and I have no other interest in this proceeding.

3. I understand that this proceeding involves U.S. Patent No. 8,190,610 (“the ’610 patent,” EX-1001), and I have been asked to provide my opinions as to the patentability of the claims of the ’610 patent.

4. I have been asked to consider the validity of certain claims of the ’610 patent based on certain prior art references. I have also been asked to consider the state of the art and prior art available as of October 5, 2006, which is the earliest claimed priority date on the face of the ’610 patent. Based on the combination of prior art references discussed in this declaration, it is my opinion that claims 1-46 of the ’610 patent are unpatentable for the reasons provided below.

II. QUALIFICATIONS AND BACKGROUND

5. I believe that I am well qualified to serve as a technical expert in this matter based upon my educational and work experience, which I summarize below.

I understand that my curriculum vitae, which includes a more detailed summary of my background, experience, and publications, is being submitted as EX-1006.

A. Education and Qualifications

6. I am currently a Professor and the David R. Cheriton Chair in the David R. Cheriton School of Computer Science at the University of Waterloo, Ontario, Canada. I have held this position since 2015. Mclean's has ranked the David R. Cheriton School of Computer Science as the top-ranked computer science program in Canada. I am also a Fellow of the Association for Computing Machinery, recognized for contributions to question answering, information retrieval, and natural language processing since 2023.

7. From 2004 to 2015, I held the positions of Assistant Professor, Associate Professor, and Professor at the University of Maryland, College Park, Maryland, USA. My primary appointments were with the College of Information Studies and the Institute of Advanced Computer Studies, with additional appointments in the Department of Computer Science.

8. I received a Ph.D. in Electrical Engineering and Computer Science from the Massachusetts Institute of Technology in 2004. I received a Master of Engineering and Bachelor of Science, also from the Massachusetts Institute of Technology, in 2001 and 2000, respectively.

9. I currently serve as the Chief Technology Officer for Primal Fusion, Inc., a startup based in Waterloo, Ontario, Canada. From 2018 to 2021, I served as the Chief Scientist of RSVP.ai, another startup based in Waterloo, Ontario, Canada. From 2010 to 2015, I was a research scientist at Twitter. From 2005 to 2009, I held various visiting positions at the National Library of Medicine, the National Institutes of Health.

10. I am presently the co-director of the Waterloo Artificial Intelligence (AI) Institute, which has the mission to cohere, advance, promote, and exploit cross-disciplinary research at the frontiers of artificial intelligence and its applications at the University of Waterloo. The Institute spans all six Faculties across campus and encompasses a large, vibrant, multidisciplinary community comprising over 230+ faculty. The University of Waterloo was the second-ranked institution for AI in Canada and the top institution for applied AI in 2022, according to CSRankings, a widely respected and cited ranking of computer science institutions around the world.

11. For nearly twenty-five years, my research has been driven by the quest to develop techniques and build tools that connect users to relevant information. Beyond text, I have explored speech, images, graphs, relational data, semi-structured logs, and more, targeting diverse groups of users, ranging from casual information seekers to data scientists, historians, legal scholars, medical

doctors, and other domain experts. My work mostly lies at the intersection of information retrieval and natural language processing, with a focus on two fundamental challenges: those of understanding and scale. As of December 2023, I have had an *h*-index of 91 and my papers collectively have been cited over 31,000 times, according to Google Scholar. I have received research funding from a multitude of sources, including governmental agencies such as the U.S. National Science Foundation and Natural Sciences and Engineering Research Council of Canada, as well as corporations, including Amazon, Yahoo!, IBM, and Google.

12. In 2007, Google and IBM introduced the Academic Cloud Computing Initiative (ACCI). This project aimed to help faculty and students address the challenges of “web-scale” computing by making computer clusters available for research and education. These clusters ran Hadoop, which is the open-source implementation of MapReduce. Six universities were involved at the outset: Berkeley, Carnegie Mellon, MIT, Stanford, University of Maryland, and University of Washington. I was the lead faculty member at Maryland for this project, and in this capacity I was among the first to teach “big data” courses on MapReduce and related technologies. I have taught numerous iterations and variations of such courses since then.

13. In 2010, I co-authored a textbook entitled “Data-Intensive Text Processing with MapReduce” with Morgan & Claypool Publishers. This textbook

is among the best sellers in the publisher's "Synthesis Lectures on Human Language Technologies" series and has been used extensively around the world for undergraduate and graduate courses on MapReduce and related "big data" technologies.

14. From around 2009 to 2010, I served as a consultant for Cloudera, which was then a startup founded around commercializing Hadoop MapReduce and other related "big data" technologies.

15. In my capacity as a research scientist at Twitter, I worked on Hadoop-based MapReduce infrastructure for data analytics and data science. The infrastructure that I helped build during that time managed tens of petabytes of data across tens of thousands of servers in multiple datacenters. In addition, I contributed to many user-facing products and services at Twitter, including real-time search, real-time query suggestion, expert search, and graph-based recommendations. These products and services are used by hundreds of millions of people around the world.

III. MATERIALS CONSIDERED

16. In forming my opinions, I have reviewed the following documents:

Exhibit	Description
EX-1001	U.S. Patent No. 8,190,610 ("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-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., <i>MapReduce: Simplified Data Processing on Large Clusters</i> , Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December 2004.
EX-1011	Jeffrey Dean et al., <i>MapReduce: Simplified Data Processing on Large Clusters</i> , available at https://research.google.com/archive/mapreduce-osdi04-slides/index.html .
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-1020	Oracle Database SQL Reference 10g
EX-1021	"Join (SQL)," Wikipedia.org (Aug. 11, 2004)
EX-1022	Hung-chih Yang, et al., "Map-Reduce-Merge: Simplified Relational Data Processing on Large Clusters," SIGMOD '07
EX-1024	U.S. Patent No. 6,343,295 to MacLeod
EX-1025	U.S. Patent Publication No. 2005/0165865 to Farmer

IV. LEGAL STANDARDS

17. In forming my opinions and considering the subject matter of the '610 patent and its claims in light of the prior art, I am relying on certain legal principles that counsel in this case explained to me. My understanding of these concepts is summarized below.

18. I understand that the claims define the invention. I also understand that an unpatentability analysis is a two-step process. First, the claims of the patent

are construed to determine their meaning and scope. Second, after the claims are construed, the content of the prior art is compared to the construed claims.

19. I understand that an invention, as defined by the claims of the patent, must not be anticipated by or rendered obvious by the prior art.

A. Claim Construction

20. I understand that the United States Patent and Trademark Office interprets claim terms in an *inter partes* review proceeding under the same claim construction standard that is used in a United States federal court. I understand that under this standard, the meaning of claim terms is considered from the viewpoint of one of ordinary skill in the art at the time of the alleged invention.

21. I have been informed that claim terms are generally given their ordinary and customary meaning as understood by one of ordinary skill in the art in light of the specification and the prosecution history pertaining to the patent. I understand, however, that claim terms are generally not limited by the embodiments described in the specification.

22. I understand that in addition to the claims, specification, and prosecution history, other evidence may be considered to ascertain the meaning of claim terms, including textbooks, encyclopedias, articles, and dictionaries. I have been informed that this other evidence is often less significant and less reliable than the claims, specification, and prosecution history.

B. Obviousness Under 35 U.S.C. § 103

23. I understand that a patent claim is invalid as obvious if the claimed invention would have been obvious to a person of ordinary skill in the art at the time the claimed invention was made. This means that even if all of the elements of the claim cannot be found in a single prior art reference that would anticipate the claim, a person of ordinary skill in the field who knows about all the prior art would have come up with the claimed invention. I understand that in an obviousness determination, the person of ordinary skill in the art is presumed to have knowledge of all material prior art. I understand that whether a claim is obvious is based upon the determination of several factual issues.

24. In considering obviousness, I understand that one must determine the scope and content of the prior art. I understand that, in order to be considered as prior art to a patent being considered, a prior art reference must be reasonably related to the claimed invention of that patent. A reference is reasonably related if it is in the same field as the claimed invention or is from another field to which a person of ordinary skill in the art would look to solve a known problem.

25. I understand that one must determine what differences, if any, existed between the claimed invention and the prior art.

26. I understand that a patent claim composed of several elements is not proved obvious merely by demonstrating that each of its elements was

independently known in the prior art. In evaluating whether such a claim would have been obvious, one may consider whether a reason has been identified that would have prompted a person of ordinary skill in the art to combine the elements or concepts from the prior art in the same way as in the claimed invention. There is no single way to define the line between true inventiveness on the one hand (which is patentable) and the application of common sense and ordinary skill to solve a problem on the other hand (which is not patentable). For example, market forces or other design incentives may be what produced a change, rather than true inventiveness.

27. I understand that one may consider whether (1) the change was merely the predictable result of using prior art elements according to their known functions, or whether it was the result of true inventiveness; (2) there is some teaching or suggestion in the prior art to make the modification or combination of elements claimed in the patent; (3) the innovation applies a known technique that had been used to improve a similar device or method in a similar way; (4) the claimed invention would have been obvious to try, meaning that the claimed innovation was one of a relatively small number of possible approaches to the problem with a reasonable expectation of success by those of ordinary skill in the art; (5) the invention merely substituted one known element for another known element in order to obtain predictable results; (6) the invention merely applies a

known technique to a known device, method, or product to yield predictable results; or (7) known work in the field may have prompted variations of use of the same inventions in the same or different fields due to market forces or design incentives that would have been predictable to a person of ordinary skill in the art.

28. I understand that one must be careful not to determine obviousness using the benefit of hindsight; many true inventions might seem obvious after the fact. I understand that the determination should be based on the position of a person of ordinary skill in the field at the time the claimed invention was made and should not consider what is known today or what is learned from the teaching of the patent.

C. Secondary Considerations of Non-Obviousness

29. I understand that certain secondary considerations may be considered in evaluating obviousness in order to prevent hindsight bias. These secondary considerations include the commercial success of products that practice the patent, long-felt need for the patented technology, failure by others to solve the problem addressed by the patent, initial skepticism by others in the industry, industry recognition and praise of the patented products, and efforts by others to copy the patented technology.

30. I have been informed that these secondary considerations of non-obviousness must be balanced against the strength of the *prima facie* case of

obviousness. Where the invention represents no more than the predictable use of prior art elements according to their established functions, secondary considerations are inadequate to establish non-obviousness.

31. I have also been informed that the patentee must establish a nexus between any secondary consideration factors and the claimed invention. Where a secondary consideration factor results from something other than what is both claimed and novel in the patent, there is no nexus to the merits of the claimed invention.

32. I have been informed that, for example, the nexus requirement as it applies to the evidence of commercial success requires that the patentee must show that the sales of a product that allegedly practices the claimed invention must be a direct result of the unique characteristics of the claimed invention, as opposed to other economic and commercial factors that are unrelated to the patented invention. I understand that this means if the commercial success is due to an element that existed in the prior art or that is not claimed by the patent, then no nexus exists.

33. I have been further informed that, for example, the nexus requirement as it applies to the long-felt need element must likewise be supported by evidence. Where the differences between the prior art and the claimed invention are minimal,

it cannot be said that any long-felt need was met by the patented invention or that the patent solved any technological problems that were unaddressed by others.

34. I have been informed that evidence of industry praise must also show a nexus with the claimed invention, and that self-serving statements by the patentee regarding the merits of the claimed invention are generally not given weight.

V. THE '610 PATENT

A. Overview of MapReduce

35. MapReduce is a programming methodology for processing big data sets in a parallel, distributed computing environment. In 2004, Jeffrey Dean and Sanjay Ghemawat of Google disclosed MapReduce in a paper presented at the 6th Symposium on Operating Systems Design and Implementation (OSDI). EX-1009. The '610 patent cites this paper and an accompanying presentation by Jeffrey Dean, '610 patent (EX-1001), 1:4-27, and acknowledges that MapReduce was a known programming methodology used for processing “parallel computations over distributed (typically, very large) data sets,” *id.*, 1:6-27.

36. MapReduce involves two primary steps: processing by a “map” function, and then processing by a “reduce” function. The map function performs an operation on each element (key-value pairs) of a data set to generate intermediate data. *See id.*, 1:6-27. The reduce function combines the intermediate data into smaller data sets or lists of values. *Id.* The map function and reduce

processes may be performed by different computers, as the '610 patent acknowledges, enabling “the use of massive clusters of commodity computers . . . for processing large sets of data in parallel.” *Id.*, 3:9-12.

37. Figures 1 and 2 of the '610 patent depict the features of what the '610 patent refers to as “conventional” MapReduce. *Id.*, 1:48-51, 2:9-3:8, Figs. 1, 2.

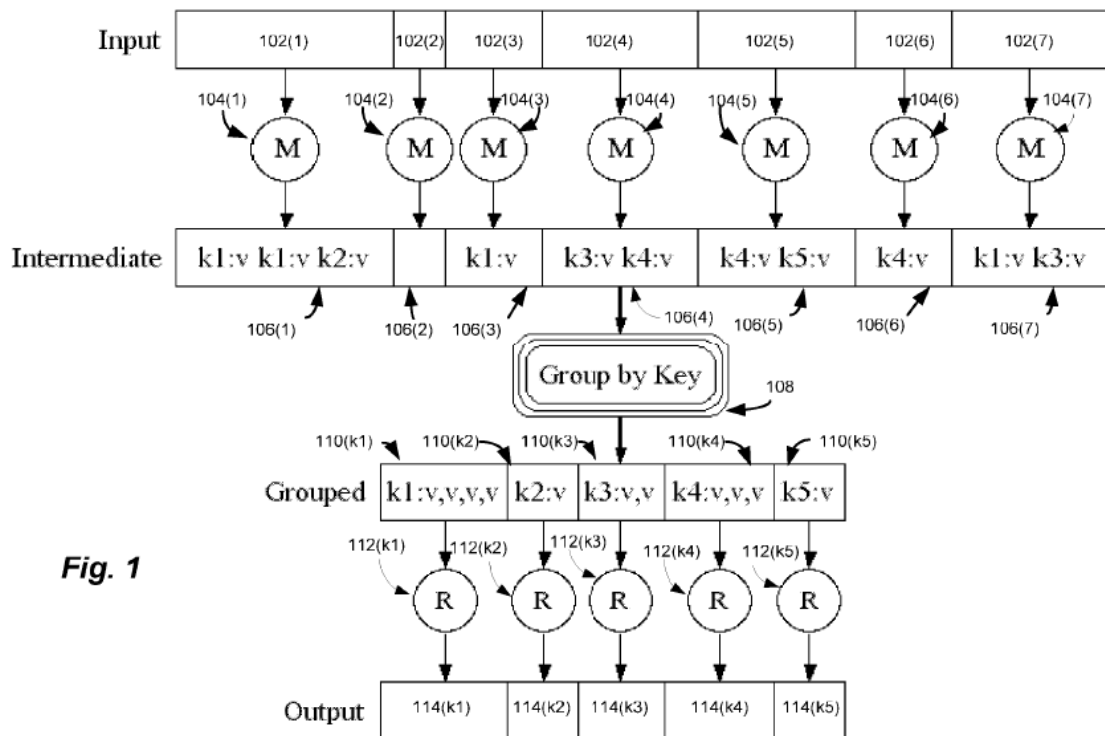


Fig. 1

Id., Fig. 1.

38. In Figure 1, the input data set 102 has several keys, including keys k1-k5. *Id.* 2:9-51. The input data 102 is divided into partitions, represented by numerals 102(1)-102(7). *Id.* Each partition, according to the '610 patent, was conventionally “provided to [a] corresponding invocation of [a] map function,” 104(1)-104(7), to generate intermediate data 106(1)-106(7). *Id.* The partitions

102(1)-102(7) could represent different files or be groups of data constituted from fewer input files. *Id.*

39. The intermediate data of the map processing could be grouped by key 108 and then provided to corresponding reduce functions 112(k1)-112(k5). *Id.* The reduce functions generate corresponding output data 114(k1)-114(k5). *Id.* As the '610 patent states, the map and reduce functions are application-specific, dependent on use case, and "user provided." *Id.* 1:17-27, 2:9-51. In other words, a human programmer develops each map and reduce function based on the data being analyzed and the given use case. In my opinion, this confirms my understanding that it was known in the art that a particular map and reduce function was designed for a given use case, and that the functions may vary depending on the type of data being consumed. Nothing is unique about using application-specific, data-specific, or use-case-specific functions.

40. Figure 2 (below) depicts a "parallel implementation of the FIG. 1 architecture." *Id.*, 2:52-67. This implementation is one possible realization of the conceptual MapReduce operations specified in Figure 1, separating the map processing into three tasks, 202a-202c, and the reduce processing into two tasks, 208a, 208b. *Id.*, 2:52-67. These tasks can be assigned to different computers over a distributed system.

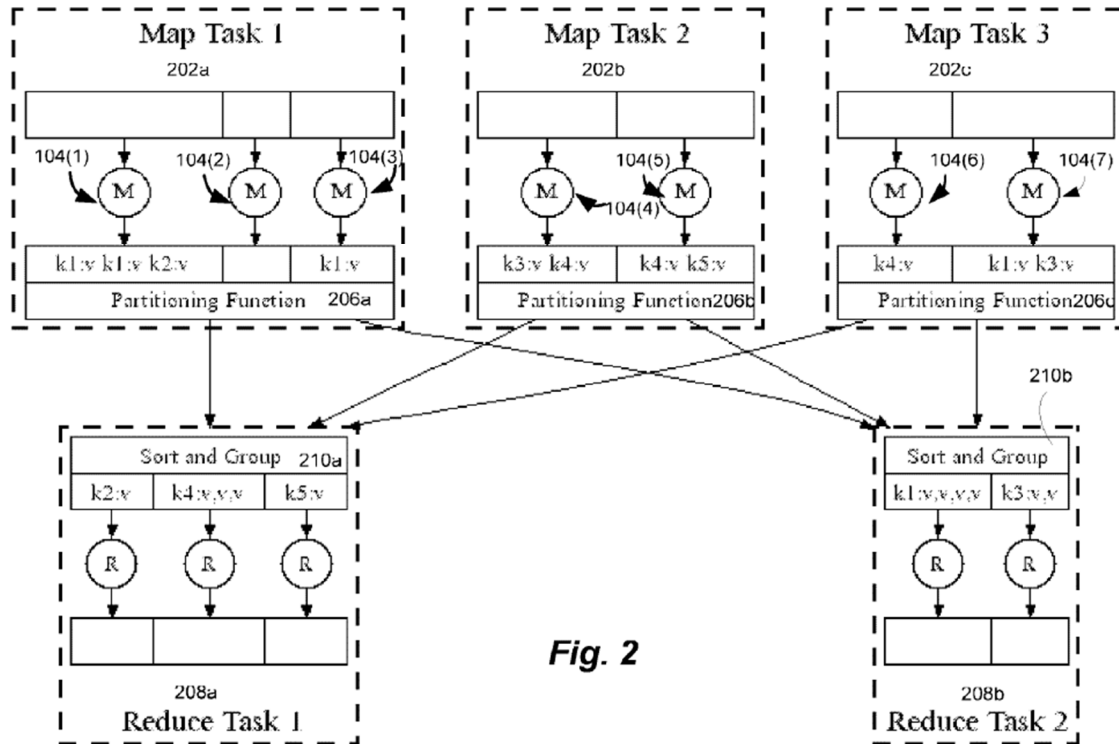


Fig. 2

Id., Fig. 2.

41. The Dean and Ghemawat paper on MapReduce, cited by the '610 patent, provides real-world uses of MapReduce. *Id.*, 1:6-15 (citing EX-1009, “Dean”). One example Dean provides is using MapReduce to “count[] the number of occurrences of each word in a large collection of documents.” EX-1009, 2 (Section 2.1). In this example, the map function “emits each word plus an associated count of occurrences” across the documents, while the reduce function “sums together all counts emitted for a particular word.” *Id.* Dean also provides pseudo-code that intakes documents whose “values” are mapped without limitation to a specific schema:

```
map (String key, String value):  
    // key: document name  
    // value: document contents  
    for each word w in value:  
        EmitIntermediate(w, "1");  
  
reduce(String key, Iterator values):  
    // key: a word  
    // values: a list of counts  
    int result = 0;  
    for each v in values:  
        result += ParseInt(v);  
    Emit(AsString(result));
```

Id.

42. Other examples in the Dean and Ghemawat paper include counting URLs by processing logs of web page requests and adding all values for the same URL (i.e., “Count of URL Access Frequency”), 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 (Section 2.3).

43. According to Dean and Ghemawat, one of Google’s “most significant uses of MapReduce” by 2004 was using MapReduce to index webpages for Google’s web search service. *Id.*, 10-11 (Section 6.1). The indexing process ran “as a sequence of five to ten MapReduce operations,” and considered “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.*

44. The above examples demonstrate what a person of ordinary skill in the art (“POSITA”) already knew before the ’610 patent: that MapReduce could process diverse data sets and analyze large volumes of input data in an efficient manner. As a POSITA would have understood, Dean and Ghemawat place no limitation on the type of input data, such as its format or “schema.” Instead, Dean and Ghemawat contemplate processing “large collection[s] of documents,” including websites and logs from the Internet. *See id.*, 2, 10-11. A POSITA would have known that different websites and logs have different schema. Dean and Ghemawat also disclose a MapReduce library that provides “support for reading input data in several different formats. For example, ‘text’ mode input treats each line as a key/value pair . . . [e]ach input type implementation knows how to split itself into meaningful ranges for processing as separate map tasks (e.g., text mode’s range splitting ensures that range splits occur only at line boundaries).” *Id.* at 6-7 (Section 4.4).

45. In my experience, and as indicated by Dean and Ghemawat, there was no limitation on the type of data MapReduce could consume in 2006. The map and reduce functions were developed by software developers to scan data of diverse format/type and from diverse sources. The map and reduce processing could be

and was designed to handle numerous data types at once, depending on the use-case.

B. State of the Art of Relational Processing

46. The concept of relational processing, processing tables of different schema and computing the results of a query that merges (or joins) data from multiple tables, including doing so over a distributed network, was not new in 2006. Instead, Relational Database Management Systems (RDBMS) employing Structured Query Language (SQL) have performed this type of processing for years. *See* EX-1020, 1-1 through 1-3.

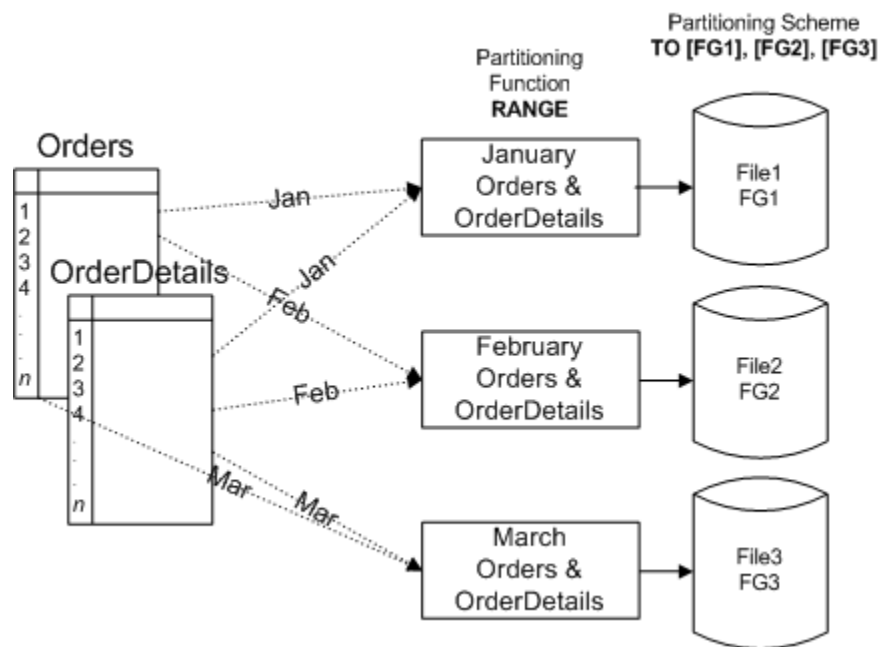
47. For example, literature describing Microsoft's SQL Server 2005 demonstrates that SQL commands were in use to partition data, process it in parallel, and then combine the results, well before the '610 patent:

Large-scale operations across extremely large data sets – typically many million rows - can benefit by performing multiple operations against individual subsets in parallel. A simple example of performance gains over partitions can be seen in previous releases with aggregations. For example, instead of aggregating a single large table, SQL Server can work on partitions independently and then aggregate the aggregates. In SQL Server 2005, joins can benefit directly from partitioning; SQL Server 2000 supported parallel join operations on subsets yet needed to create the subsets on the fly. In SQL Server 2005, related tables (i.e. Order and OrderDetails) that are partitioned to the same partitioning key and the same partitioning function are said to be aligned. When the optimizer detects that two partitioned and aligned tables are joined, SQL Server 2005 can choose to join the data which resides on

the same partitions first and then combine the results. This allows SQL Server 2005 to more effectively use multiple-CPU machines.

EX-1016 (“Tripp”), 3 (“Why do you need Partitioning?”).

48. SQL Server 2005 could process tables of different schema just like those disclosed in the '610 patent. *Id.* Indeed, just like the '610 patent, Tripp discloses using SQL Server 2005 to process “Orders” and “OrderDetails” tables:



Id. at Fig. 3.

49. Relational processing to combine tables over distributed systems was also thoroughly disclosed in patent literature before the '610 patent. U.S. Patent No. 5,797,000 to Bhattacharya, for example, described a system in 1998 for “performing a parallel join operation on a pair of relations R1 and R2 in a system containing P processors organized into Q clusters.” EX-1017, Abstract.

Bhattacharya disclosed “a join operation performed on two database tables (or relations, in relational database terminology) such as tables 200 (EMPLOYEE) and 300 (DEPT).” *Id.*, 5:9-7:29, Fig. 3 (below). Similar to the map and reduce processing described by the ’610 patent, Bhattacharya disclosed that its tables were merged in “four successive phases: a sort phase 510, a partition phase 520, a transfer phase 530, and a join phase 540” that are “performed in parallel.” *Id.*, 8:31-62, Fig. 5.

200

EMPLOYEE

LAST NAME	DEPT	SALARY
ALLEN	SALES	30000
BROWN	PERSONNEL	40000
•	•	•
JONES	SALES	50000
•	•	•
SMITH	ACCOUNTING	45000
•	•	•
YOUNG	ACCOUNTING	25000

FIG.2

300

DEPT

DEPT	MGR
ACCOUNTING PERSONNEL	SMITH BROWN
SALES	JONES

FIG.3

Id., Figs. 2, 3.

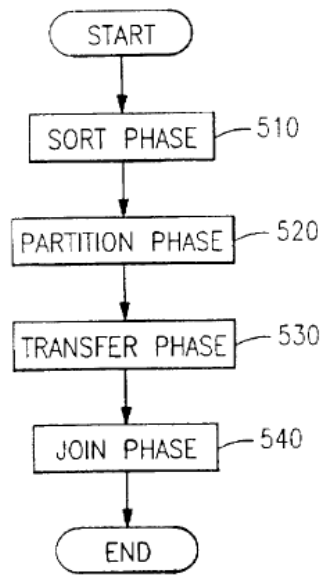
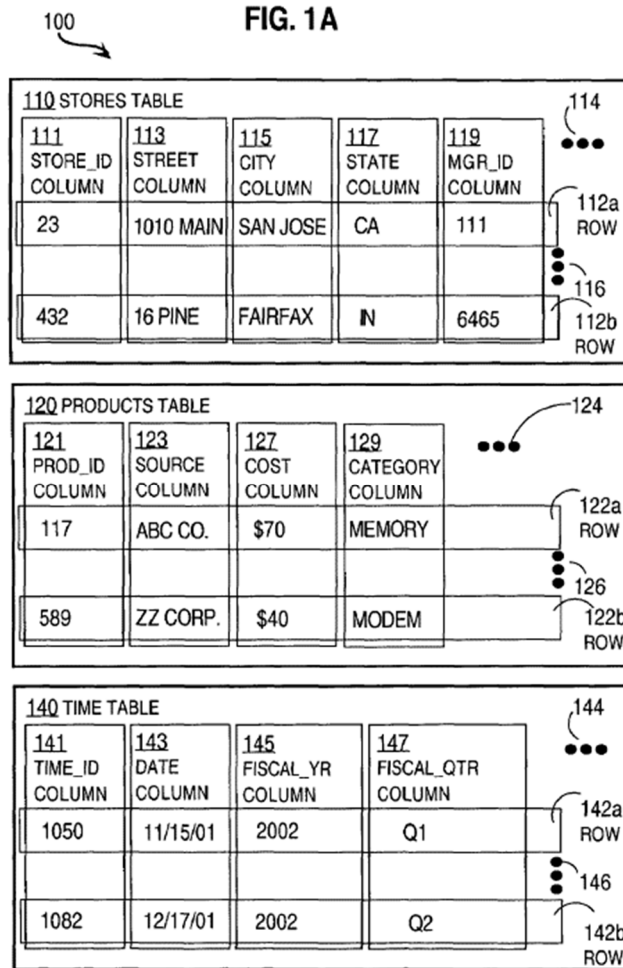


FIG.5

Id., Fig. 5.

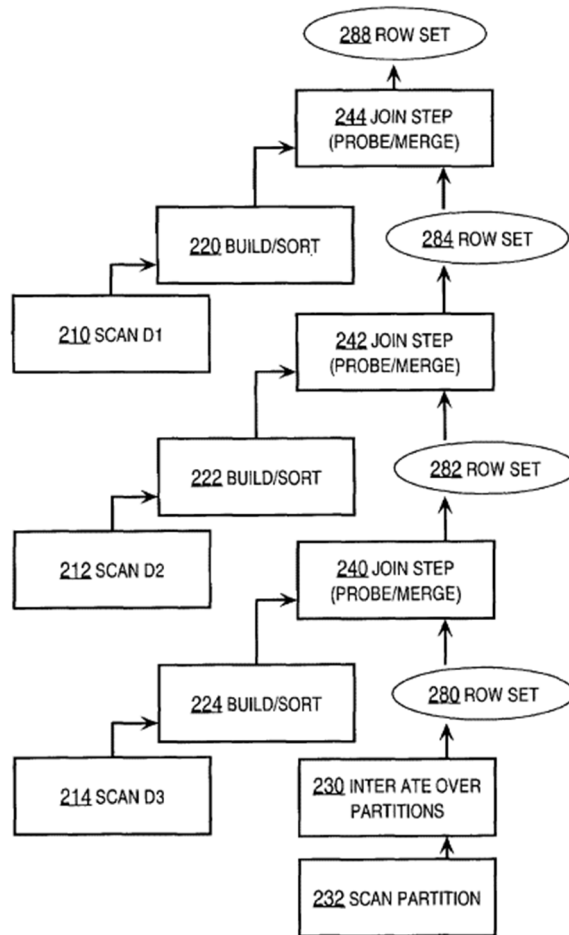
50. In my opinion, a POSITA would have recognized the similarities and correspondences between the phases defined by Bhattacharya and MapReduce, including the MapReduce disclosures in Pike. *See infra* Section VII.A (“Overview of Pike”).

51. Another patent document, U.S. Patent No. 7,020,661 to Cruanes, similarly disclosed joining a plurality of data objects from different tables, including a stores table 110, products table 120, and time table 140. EX-1018, Abstract, 5:29-6:32, Fig. 1A. Similar to the processing described in the ’610 patent, Cruanes scanned each table for a “sort-merge join” operation, as shown in Figure 2. *Id.*, 9:4-10:5, Fig. 2.



Id., Fig. 1A.

FIG. 2



Id., Fig. 2.

52. Just like Bhattacharya, in my opinion a POSITA would have recognized the similarities and correspondences between the “sort-merge join” operation defined in Cruanes and MapReduce, including the MapReduce disclosures in Pike. *See infra* Section VII.A (“Overview of Pike”).

53. A POSITA would have understood that the ’610 patent’s mapping and joining tables of different schema was merely an application of known relational processing in prior art, as disclosed in Bhattacharya and Cruanes. Indeed, the exact

example of processing a table of Employees and Departments, as illustrated in Figures 3 and 5 of the '610 patent, is taken from textbook examples of "JOIN" and "GROUP BY" functions defined in known SQL databases. *See, e.g.*, EX-1020, 14-73 (providing an example SQL query to "GROUP BY department_id"), 19-48 through 19-49 (illustrating a "Join Query" that "returns the name and job of each employee and the number and name of the department in which the employee works" where the input tables are separate "employees" and "departments" tables).

54. In fact, the example in the '610 patent is nearly identical to a 2004 Wikipedia page that described an SQL "Join" operation. EX-1021. As shown below, an SQL join "combines records from two or more tables in a relational database," just like the '610 patent. *Id.*

610 Patent	Join (SQL) Wikipedia Page (2004)
All Emp (DeptID, EmpName) 34, Smith 33, Jones 34, Robinson 34, Jasper 33, Stone 31, Rosen	name dept_id ---- - Smith 34 Jones 33 Robinson 34 Jasper 34 Steinberg 33 Rafferty 31
All Dept (DeptID, DeptName) 31, Sales 33, Engineering 34, Clerical 35, Marketing	job ID ---- -- Sales 31 Engineering 33 Clerical 34

306

All Emp and Dept
(DeptID, EmpName, DeptName)

34, Smith, Clerical
33, Jones, Engineering
34, Robinson, Clerical
34, Jasper, Clerical
33, Stone, Engineering
31, Rosen, Sales

Your result set will be:

name	job
---	---
Smith	Clerical
Jones	Engineering
Robinson	Clerical
Jasper	Clerical
Steinberg	Engineering
Rafferty	Sales

```
// group: "emp" or "dept"  
// key: DeptID  
// val: LastName for "emp", DeptName for "dept"
```

Use this SQL statement to join the tables:

```
SELECT name, job  
FROM employee, department  
WHERE employee.dept_id = department.ID
```

Compare *id.*, with '610 patent, 6:28-34, Fig. 3.

55. After 2006, the inventors of the '610 patent conceded that their alleged contribution to MapReduce was simply the introduction of a known relational processing operation. *See* EX-1022. They explained that “[p]rocessing data relationships [wa]s ubiquitous, especially in enterprise information systems.” *Id.* at 1030 (Section 1). They further explained that “[o]ne major focus of the extremely popular relational algebra and RDBMS [wa]s to model and manage data relationships efficiently.” *Id.* In applying MapReduce to implement these known relational processing techniques, the inventors acknowledged that the “[s]traightforward filtering conditions . . . in a SQL query’s ‘where’ and ‘having’

clauses can be implemented using [MapReduce's] mappers and reducers, respectively.” *Id.*, 1036 (Section 4.1). They further acknowledged that MapReduce’s “reducers can easily implement the ‘group by’ clause and ‘aggregate’ operators in SQL.” *Id.* The inventors then described implementing “an SQL-like interface” for MapReduce that “can readily reuse well-studied RDBMS techniques.” *Id.*, 1040 (Section 8). These statements confirm what a POSITA would have understood: that the inventors of the ’610 patent disclosed a number of straightforward and known relational processing techniques—including joins.

C. Overview of the ’610 Patent

56. The ’610 patent claims to have invented an “enhanced” form of MapReduce. ’610 patent, Abstract, 1:31-44. In my opinion, the disclosed “enhancement” is anything but new and amounts to a basic characterization of MapReduce’s input data. The patent concedes this: “[t]he inventors have realized that, 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, Abstract. In other words, the patent specifies a plurality of “data groups” that are processed by MapReduce. 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 therefore specifies a different data format or “schema” in different input data

groups. The '610 patent's example of different "schema" is an employee table 302 and department table 304, where each table includes a different data field from the other. *Id.*, 3:20-47, 3:65-4:3, 8:47-52, Figs. 3, 4.

57. In my opinion, specifying MapReduce's input data as data groups having different schema is not new. It is simply organizing the data in each partition 102 in MapReduce, as shown in Figure 1 of the patent. In my opinion, a person of ordinary skill would have known how to do this as the Dean and Ghemawat paper already discloses MapReduce's ability to process diverse datasets having different schemas. *See supra* Section V.A ("Overview of MapReduce"). The '610 patent does not discuss whether MapReduce provided this same capability—to map and reduce input data of different schema. For example, the patent states the "enhanced" MapReduce can map and reduce "two or more related" data sets that are only "***possibly*** heterogeneous" to one another. *Id.*, Abstract (emphasis added), 2:47-52. The patent also alleges that MapReduce could not "***efficiently*** process data from heterogeneous sources," *id.*, 3:9-20 (emphasis added), and that it was "impractical" for MapReduce to join two relational tables having different schemas, *id.* The patent does not assert that MapReduce could not process data of different schema from heterogeneous data sources. It is my opinion that existing functionality in MapReduce before the '610 patent could perform this

function, a person of ordinary skill knew how to do this, and that the patent discloses a mere use case of MapReduce that maps and reduces specific data.

58. Indeed, the patent acknowledges that the “related . . . datasets” of its input data are just like those of conventional MapReduce: “characterized by a common primary key.” *Compare id.*, Abstract, 1:34-36, 2:2-4, 8:4752, *with id.* Fig. 1 (input data set 102(1), 102(1), and 102(7) sharing a common key in k1:v), 2:12-18, 2:49-51. In my opinion, the ’610 patent does not describe any enhancement to MapReduce, it only describes a use-case. Input data of MapReduce could be of diverse format before the ’610 patent, and could be grouped with each group having different schema.

59. Figures 4 and 5 provide depictions of the alleged enhanced MapReduce applied to an Employee table 302 and a Department table 304. *Id.*, 3:65-4:22, 8:15-46, Figs. 4, 5. But these figures mirror conventional MapReduce in Figures 1 and 2, as I illustrate below in the annotations to Figure 4, showing the same **partitioned input data**, **mapping tasks**, **intermediate data sorted by key-value pair**, and **reducing tasks with output data**. The only difference is that groupings of particular data are mapped and reduced.

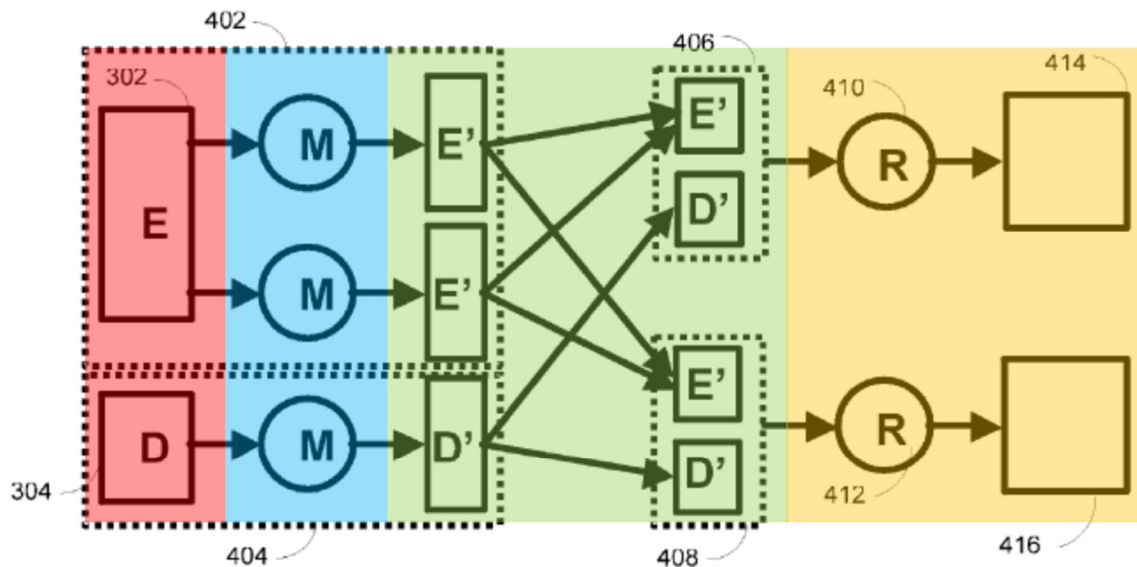


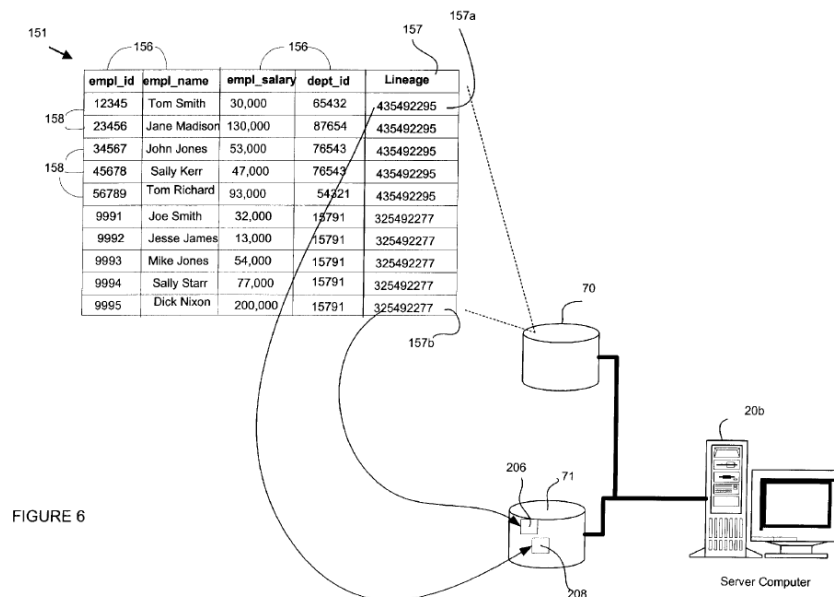
Fig. 4

Id., Fig. 4 (annotated).

60. As explained above, the inventors of the '610 patent acknowledged that MapReduce could process data “from multiple sources.” EX-1022, 1029-30 (Section 1). The inventors only concede that developers would have to encode “an additional data-source attribute in the data and extra conditions in the code.” *Id.* Despite acknowledging this capability, the inventors asserted an improvement in the use of identifiers for different inputs: i.e., “introduc[ing] a naming and configuring scheme.” *Id.*, 1030 (Section 1). Using these identifiers, “[d]ata entries from different datasets with the same key value will be grouped in the same reduce partition,” and “[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 (Section 2.2). In my opinion, the inventors conceded their purported improvement to MapReduce—encoding data tags or identifiers for the source of data—was a known MapReduce capability, and that developers could easily encode input data this way before the '610 patent.

61. Moreover, encoding data with data tags or identifiers was not new in 2006, including doing so to track data in a data processing system. For example, U.S. Patent No. 6,343,295 to MacLeod, which issued in 2002, disclosed tracking data in a database by “adding a lineage identifier” to data in an RDMBS database. EX-1024, Abstract, 5:30-42, 5:53-60, Fig. 6. The lineage identifier could be used to “trace the source of the data,” “provide details about transformations undergone by the data,” and “track[] program modules . . . that transformed the particular data under consideration.” *Id.*, Abstract.



Id., Fig. 6.

62. U.S. Patent Publication No. 2005/0165865 to Farmer, which published in 2005, similarly disclosed a system that “records changes to an information system” using metadata. EX-1025, Abstract, ¶¶5, 8, 26. Farmer explained that the system generates metadata, including a unique key, whenever an object is modified, created or deleted, and the metadata is stored in a journal that “may be searched and queried to provide metadata analysis, such as lineage or impact analysis for an object in the system.” *Id.*, Abstract, ¶¶26-27.

63. MacLeod and Farmer are exemplary disclosures known in the art that demonstrate that encoding data with identifiers or metadata to track data through a system was known in the art before the ’610 patent. This reinforces my opinion that the ’610 patent introduced nothing new in the art, and at most employed MapReduce with known techniques of identifying data.

D. Summary of the Prosecution History of the ’610 Patent

64. I have reviewed the prosecution history of the application that led to the ’610 patent. EX-1004. I understand that the Patent Office rejected the application that would issue as the ’610 patent in four separate office actions. I understand that these office actions included rejections for ineligible subject matter (i.e., claiming abstract ideas), and rejections for being obvious over the prior art. EX-1004, 202-39, 151-63, 109-23, 74-86. I understand that the Patent Office

Examiner did not consider the art cited below and asserted by Petitioner (Cloudera) in this proceeding. '610 patent, Cover.

E. Claim Construction of Terms in the '610 Patent

65. As discussed above, I have been informed that, for the purposes of this proceeding, the standard for claim construction of terms within the claims of the patent is the same as that applied in federal district court litigation. I understand that a district court has previously construed terms of this patent, and I have been informed of those constructions. Specifically, I understand the district court issued the following constructions:

Claim Term	District Court's Construction
a plurality of mapping functions that are each user-configurable	two or more mapping functions that are each configurable by a user
the data of the first data group is mapped differently than the data of the second data group	plain meaning
data set	plain meaning
data group	a group of data and a mechanism for identifying data from that group
data partition	plain meaning
including processing the intermediate data for each data group in a manner that is defined to correspond to that data group, so as to result in a merging of the corresponding different	plain meaning

intermediate data based on the key in common	
the at least one output data group is a plurality of output data groups	plain meaning
identifiable to [that/the] data group	plain meaning
independently output a plurality of lists of values	plain meaning

EX-1007, 13-31; EX-1008, 10-19. I understand that the district court also found the preambles of claims 1 and 17 limiting. EX-1007, 19-23. For mapping the data groups “differently,” I understand that the court declined to require the mapping *functions* to be different. *Id.*, 16-18.

66. I have applied these constructions in my analysis. For terms that the district court did not previously construe, I have applied the ordinary and customary meaning as understood by one of ordinary skill in the art in light of the specification and the prosecution history.

VI. PERSON OF ORDINARY SKILL IN THE ART

67. I am informed that patentability must be analyzed from the perspective of “one of ordinary skill in the art” in the same field as the patent-in-suit at the time of the invention. I am also informed that several factors are considered in assessing the level of ordinary skill in the art, including (1) the types of problems encountered in the art; (2) the prior art solutions to those

problems; (3) the rapidity with which innovations are made; (4) the sophistication of the technology; and (5) the education level of active workers in the field.

68. I will refer to a person of ordinary skill in the art here as a “POSITA.” In my opinion, pertinent to the ’610 patent as of its earliest priority date, a POSITA would have had a minimum of a bachelor’s degree in computer science or similar field, and approximately two years of industry or academic experience in a field related to performing data analytics and/or related data processing tasks. Work experience can substitute for formal education and additional formal education can substitute for work experience. I at least meet this definition of a POSITA, now and as of the priority date of the ’610 patent.

69. I am also familiar with the knowledge of the person of ordinary skill in the art as of the priority date of the ’610 patent. I am able to opine on how the person of ordinary skill in the art would have understood the disclosure and claims of the ’610 patent, the disclosures of the prior art, the motivation to combine the prior art, and what combinations would have been obvious to one of ordinary skill in the art.

VII. THE PRIOR ART

A. Overview of Pike

70. Pike is a patent assigned on its face to Google, and it names Jeffrey Dean and Sanjay Ghemawat as coinventors. Pike (EX-1002), Cover. Pike discloses

a “large-scale data processing model 100” of MapReduce that includes mapping operations in a map phase 102, and reduction operations in a reduce phase 104. *Id.*, 2:12-39, 8:47-11:55.

71. In Figure 1, reproduced below, the mapping operations 102 are applied “to a set of input data α_i , (e.g., text files, records, logs, sorted maps, etc.) to provide a set of intermediate data values β_i .” *Id.* The reduction operations 104 reduce “the set of intermediate data values β_i , to provide a set of output data ϕ_i (e.g., tables, sorted maps, record I/O, etc.).” *Id.* Pike discloses that the mapping 102 and reducing 104 operations are “one or more application-specific” functions, *id.*, and that the data processing can be performed over distributed data processing system 500, *id.*, 2:41-3:35.

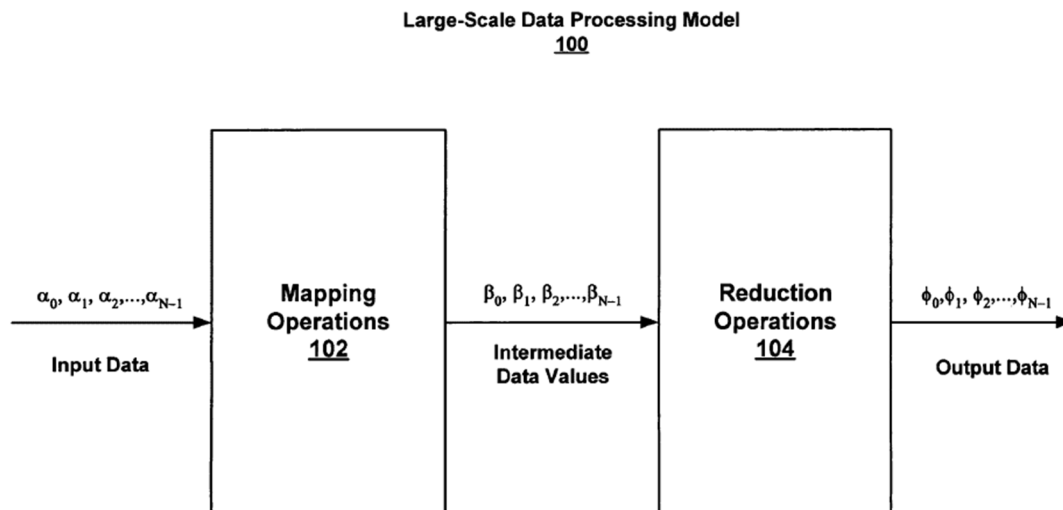


Figure 1

Id., Fig. 1.

72. In an embodiment shown in Figure 3, Pike discloses worker processes 304 and 308 that invoke map and reduce processes, respectively. *Id.*, 8:65-10:27, Fig. 3. These operations may be carried out by different machines available in system 300. *Id.* The input data for the system includes “input files 302,” which “can include a variety of data types typically used in data processing systems,” including “text files” and “tables.” *Id.*, 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 to the respective input data block 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 one or more output files 310” using “application-specific reduce() operator[s].” *Id.*, 11:6-55.

73. As I show in annotated Figure 3 below, Pike’s process corresponds to the preferred embodiment of the ’610 patent by disclosing **partitioned input data**, **mapping tasks**, **intermediate data sorted by key-value pair**, and **reducing tasks with output data**.

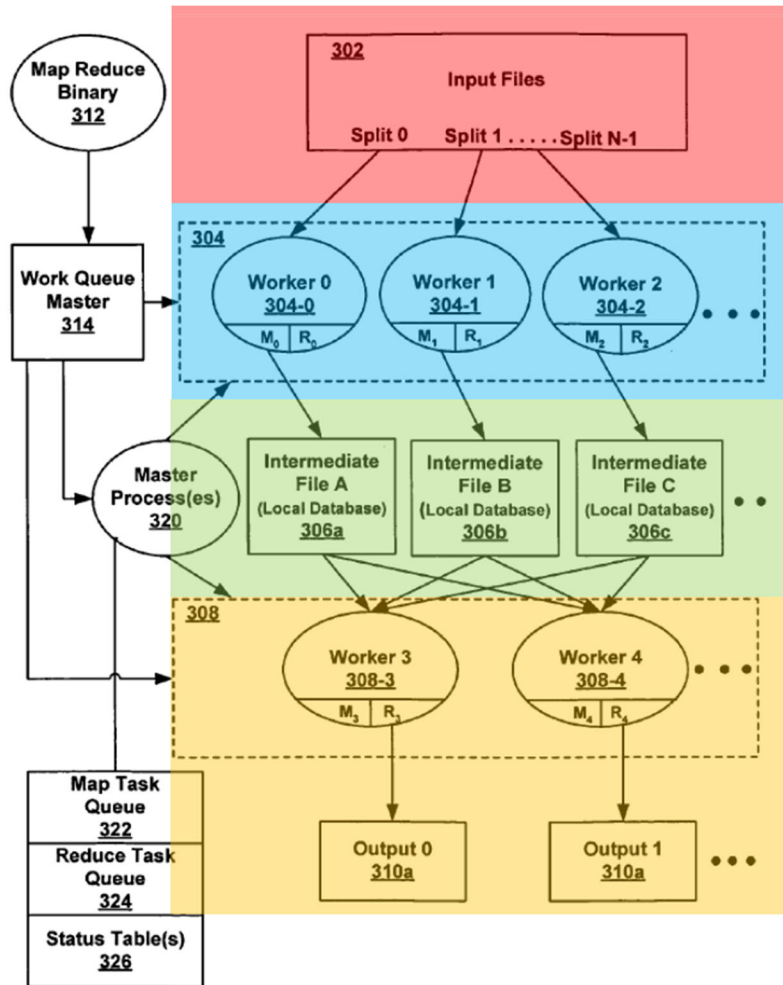


Figure 3

Id., Fig. 3 (annotated).

74. Pike additionally discloses, as shown in Figure 7A below, 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).

75. Pike discloses no file format or schema limitation to its mapping and reducing processes. *Id.*, 10:4-7, 2:17-41, 10:31-48. Indeed, a POSITA would have known from examining Pike’s disclosure that its mapping and reducing processes accept various data sets having different formats or schemas, from different sources. Pike further discloses that its mapping and reducing operations are application-specific, and thus depend on use case. *Id.*, 10:4-7, 10:29-52, 11:36-41. This reflects the common knowledge before the ’610 patent that, in the MapReduce framework, mapping and reducing functions are defined based on the given use case or data set.

B. Overview of Chowdhuri

76. Chowdhuri discloses methods for improving data query performance in the context of relational processing by dynamically partitioning data. Chowdhuri

(EX-1003), Abstract. Chowdhuri employs iterators that are “self-contained software objects that accept a stream of rows from one, two, or multiple data sources.” *Id.*, ¶¶129-31. Chowdhuri discloses that the iterators can operate without knowing “the source of a data stream.” *Id.*

77. In one embodiment, Chowdhuri discloses the use of iterators in a “serial operator tree 300.” *Id.* ¶131, Fig. 3A. In this embodiment, “order and customer tables (data 308, 309) are scanned by two separate scan iterators, 301 and 302, respectively.” *Id.*

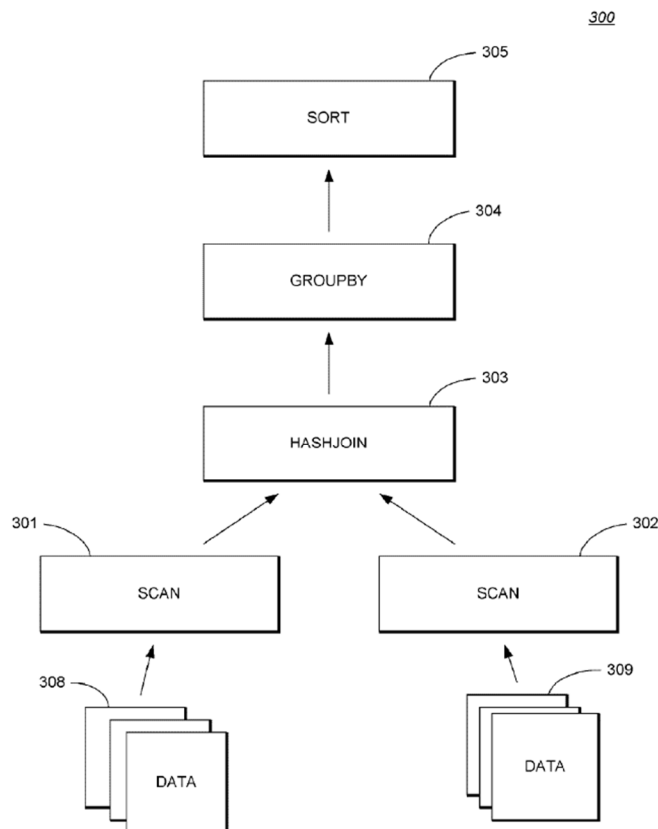
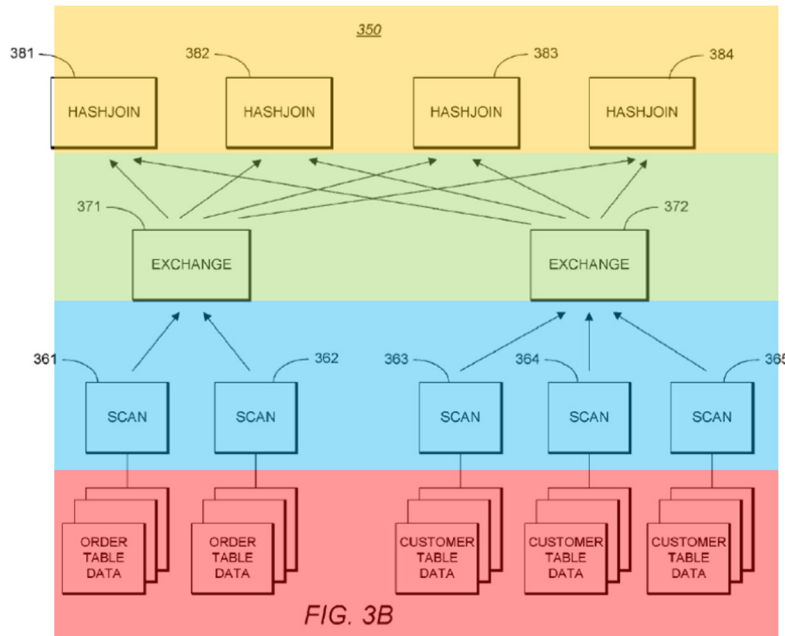


FIG. 3A

Id., Fig. 3A.

78. The results of Chowdhuri's scanning are sent to a hash join iterator 303 for the results to be matched, after which the "joined rows are then sent to the grouping iterator (GroupBy) 304 that performs a grouping operation." *Id.* ¶131. The results of the GroupBy iterator 304 are then sorted by the sort iterator 305. *Id.* In this embodiment, Chowdhuri discloses scanning two data tables of different schema, joining and sorting data from the same, and then merging the data into an output. *Id.* ¶¶130-31.

79. Chowdhuri also discloses in another embodiment using "exchange operators (iterators)" to "parallelize the serial query execution plan of FIG. 3A." *Id.*, ¶135. This process similarly performs map and reduce operations, as I illustrate in annotations to Figure 3B below, including mapping and reducing data tables of different schema. For example, Chowdhuri discloses **partitioned input data**, **mapping tasks**, **intermediate data sorted by key-value pair**, and **reducing tasks** with output data.



Id., Fig. 3B (annotated).

80. In my opinion, Chowdhuri teaches mapping and reducing using its iterators. Chowdhuri’s embodiments also disclose mapping and reducing input data of different schema in its order and customer tables.

C. Overview of MacLeod

81. MacLeod discloses a system “for tracking the lineage of data in a database.” EX-1024, Abstract, 1:12-59. Like Chowdhuri, MacLeod discloses improvements to relational databases, where “collection[s] of related data [] is organized in related two-dimensional tables of columns and rows wherein information can be derived by performing set operations on the tables, such as join, sort, merge, and so on.” *Id.*, 1:19-59. To track data through various operations, MacLeod discloses a “lineage identifier” that traces the source of the data that 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.*, 1:60-2:22. Figure 5 depicts a diagram showing “the transformation of data as it moves between databases” and exemplary lineage identifiers 157a. *Id.*, 2:42-43, 7:35-61.

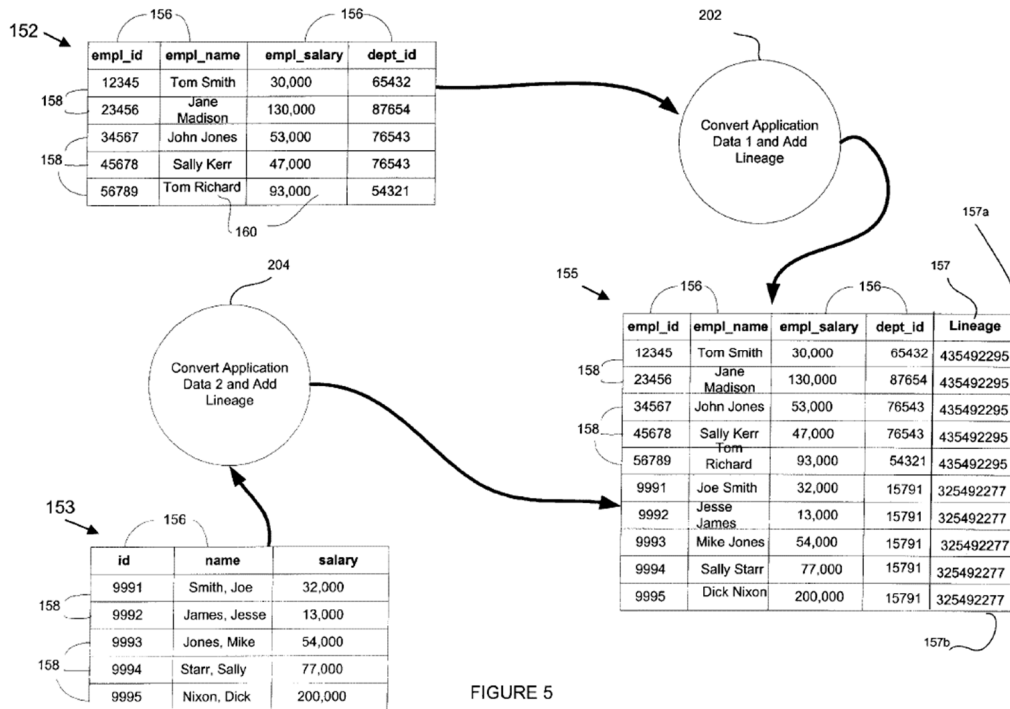


FIGURE 5

Id., Fig. 5.

VIII. OPINIONS ON UNPATENTABILITY

82. In the analysis that follows, I identify the following prior art and combinations of prior art that, in my opinion, render obvious the claims of the '610 patent:

Grounds	Combination of Prior Art References
1	Pike renders obvious claims 1-46 under 35 U.S.C. § 103.
2	Pike in view of Chowdhuri renders obvious claims 1-46 under 35 U.S.C. § 103.
3	Pike and Chowdhuri in view of MacLeod renders obvious claims 3, 4, 9-11, 19, 26, 27, 37-39, 44-46 under 35 U.S.C. § 103.

A. Pike (Ground 1) and Pike in View of Chowdhuri (Ground 2) Render Claims 1-46 Obvious; Pike and Chowdhuri in view of MacLeod (Ground 3) Renders Claims 3, 4, 9-11, 19, 26, 27, 37-39 Obvious

1. Claim 1

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

83. The '610 patent admits that this feature was “conventional” in MapReduce. '610 patent, 1:6-27, 3:9-15. Pike also discloses this element by describing a method of “large-scale data processing” implemented “in a parallel and *distributed* processing environment.” Pike, 1:10-26, 2:14-49 (emphasis added). For example, Pike discloses distributed data processing system 500 and computer system 400.

84. Pike 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. For example, 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.

85. In my opinion, a POSITA would have understood that Pike’s large-scale data processing system 300, operated over either computer system 400, discloses a processing data of a data set over a distributed system, as required by claim 1 because Pike discloses processing data over a system with distributed machines and remote storage.

86. Chowdhuri also discloses this claim feature by disclosing that its methods are “embodied in a multi-user computer system, such as a client/server system” (**Ground 2**). Chowdhuri ¶¶105-08. 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.

87. In my opinion, Chowdhuri's iterators executing in a client/server system and across multiple CPUs discloses "processing data of a data set over a distributed system," as claimed. A POSITA would have found it obvious to implement the combined Pike-Chowdhuri system (as I describe in greater detail below) using either of Pike's large-scale data processing system 300 or Chowdhuri's distributed system. In my opinion, 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.

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

88. I understand that the term "data group" has been construed to mean "a group of data and a mechanism for identifying data from that group." *See supra* Section V.E ("Claim Construction of Terms in the '610 Patent"). This feature was known because MapReduce operated on a data set comprising a plurality of groupings 102(1) through 102(7). '610 patent, 2:9-35, Fig. 1. Indeed, the patent shows and describes an "input data set 102" and that "the input data set 102 is partitioned into an arbitrary grouping of seven partitions 102(1) through 102(7)." *Id.*

89. A POSITA would have understood that each input data partition 102(1) through 102(7) is a grouping of data, or data group, that is provided to a respective map function. The '610 patent explains that “[e]ach partition is provided to the corresponding invocation of the map function, which has seven invocations 104(1) through 104(7).” *Id.* Each partition from Figure 1 of the patent—constituting one data group with its own invocation of a map function—corresponds with data groups of the “enhanced” MapReduce, which are provided to respective invocation of a map function as a single partition. *Id.*, 3:65-4:22, Fig. 4 (showing “table 304” one of the “separate data groups” that is provided to a map task 404). To the extent it is argued this data group feature is not disclosed by the '610 patent as part of the prior art, I disagree.

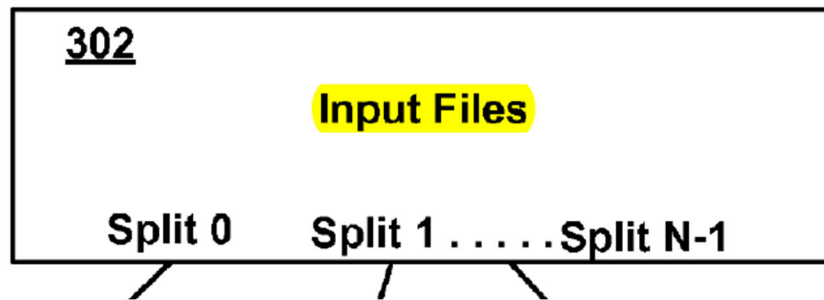
90. Pike also discloses or renders this feature obvious (**Grounds 1 and 2**). Pike discloses that its input data for large-scale data processing system 300 includes a plurality of data groups in its “input files 302.” *E.g.*, Pike, 9:58-64, Fig. 3. The input files 302 “include a variety of data types typically used in data processing systems, including without limitation text files, . . . , tables and the like.” *Id.*, 10:1-4, 2:22-26.

91. A POSITA would have understood that Pike’s input files 302 teach a group of data and a mechanism for identifying data from that group because Pike’s input files 302 are groups of data identified by, for example, a file name. 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. This is a “mechanism for identifying data from [a data] group.”

92. 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). A POSITA would understand that other input files in Pike would have similar identifiers indicating the file content and structure. 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* ’610 patent , 4:1-5:18, 7:18-23, 8:15-46, Figs. 4, 5. Pike therefore teaches a data set comprising data groups.

93. A POSITA at least 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. This would have allowed for locating that data group in Pike’s system and understanding the data in that file. 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.



Id., Fig. 3 (annotated).

94. 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.

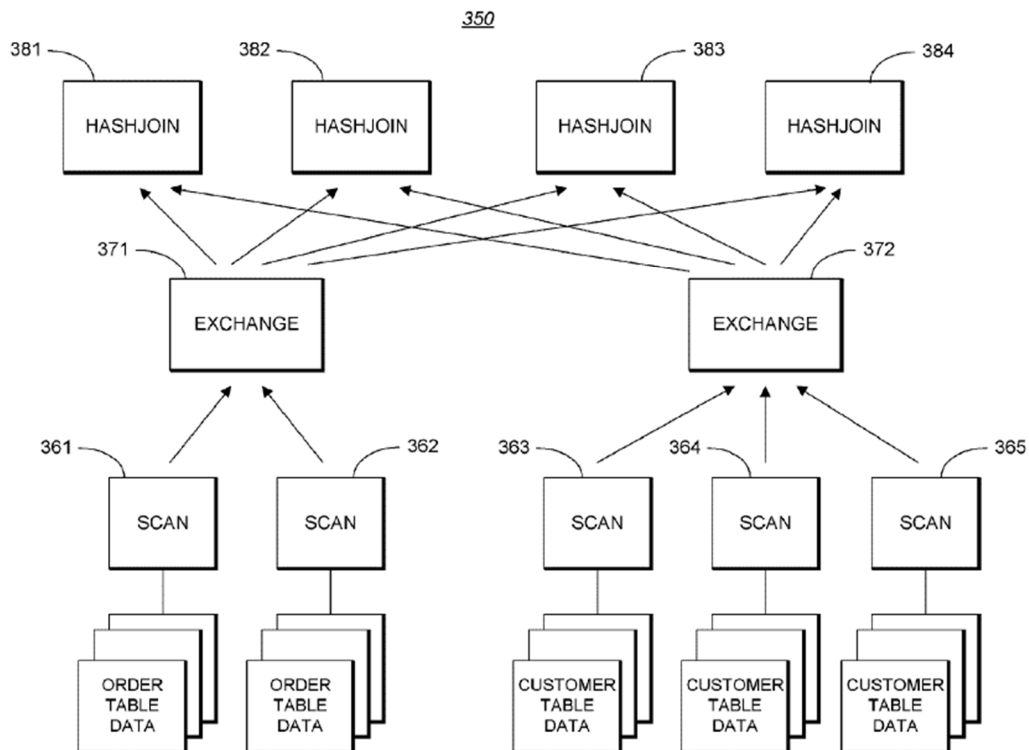


FIG. 3B

Id., Fig. 3B.

95. It would have been obvious to combine Pike with Chowdhuri to process input data that includes Chowdhuri's data groups. Doing so in my opinion would have involved nothing more than the simple substitution of Chowdhuri's input tables for Pike's input files, with predictable results. 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.

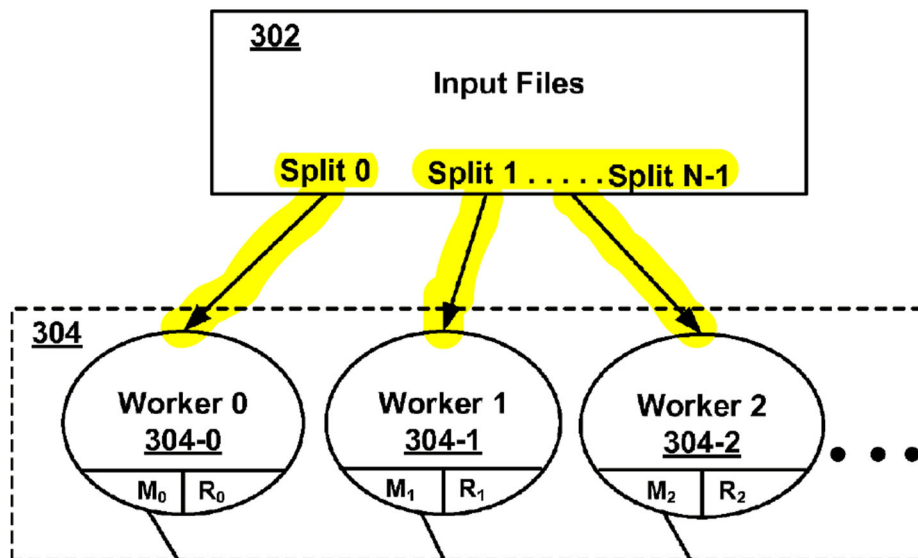
96. A POSITA also would have been motivated to map and reduce Chowdhuri's tables in Pike's system 300 because Chowdhuri teaches and suggests performing similar MapReduce operations on its data tables—scanning and merging its customer and order tables to identify orders from a specific "state." Chowdhuri ¶¶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. Chowdhuri's scan, hashjoin, GroupBy, and sort iterators also correspond to the '610 patent's description of map and reduce functions. '610 patent, 1:20-27 ("[t]he map function iterates" and "[t]he reduce operation ... combines elements"), 5:19-39 (sorting), 2:65-66 ("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").

97. A POSITA would have looked to Pike to map and reduce Chowdhuri's tables using Pike's distributed system to, for example, leverage Pike's distributed system and its fault tolerant task scheduling. Pike, 4:28-53, 6:37-54, 6:55-7:25, 11:57-13:3. Indeed, Pike explains that its fault tolerance is a major advance over existing processing methods: no existing "languages and systems automatically handle system faults (e.g., processor failures) and I/O scheduling." *Id.*, 1:27-35, 6:37-54 (describing when processes fail and "causing the failed process to be restarted or replaced by a new process"), 11:57-13:3 (handling "straggler" tasks or machines). Combining Chowdhuri's tables with Pike's distributed system would have been nothing more than combining known elements in the prior art to achieve predictable results. 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. The POSITA would have been motivated to do so to leverage Pike's fault tolerant processing and efficient task management, which Pike discloses "significantly reduces the time to complete large map-reduce operations." *Id.*, 11:57-13:3.

- c. [1c] “the method comprising: partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and”

98. In my opinion, the '610 patent admits that this feature was conventional because it states that the input data set is partitioned before mapping. '610 patent, 2:19-52, Fig. 1.

99. Pike also discloses this feature (**Grounds 1 and 2**). Pike discloses that its input files 302 (i.e., “data groups”) 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. Pike’s data blocks are then “treated as key/value pairs” to be read by the worker processes 304 to apply an “application-specific map() operator.” *Id.*, 9:58-10:19.



Id. at Fig. 3 (annotated).

100. 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. A POSITA would have understood from this 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.

101. Implementing Pike to partition each input file would have been obvious and well within the abilities of any POSITA or MapReduce programmer. Indeed, Pike’s “division of input files 302 into data blocks 0, ... , N-1” suggests doing so. *Id.*, 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. This reduces computation time and takes advantage of Pike’s parallel processing capabilities.

102. 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”). 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.” *Id.* ¶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.

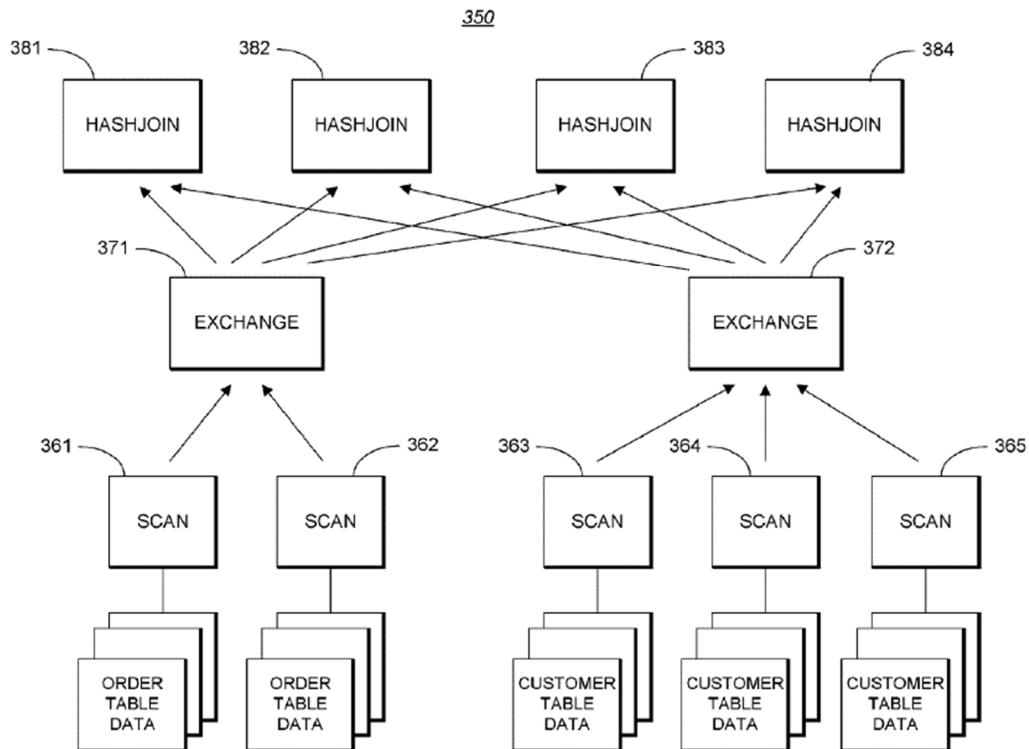


FIG. 3B

Id., Fig. 3B.

103. In my opinion, 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. Pike, 4:28-53, 6:55-7:25, 11:57-13:3.

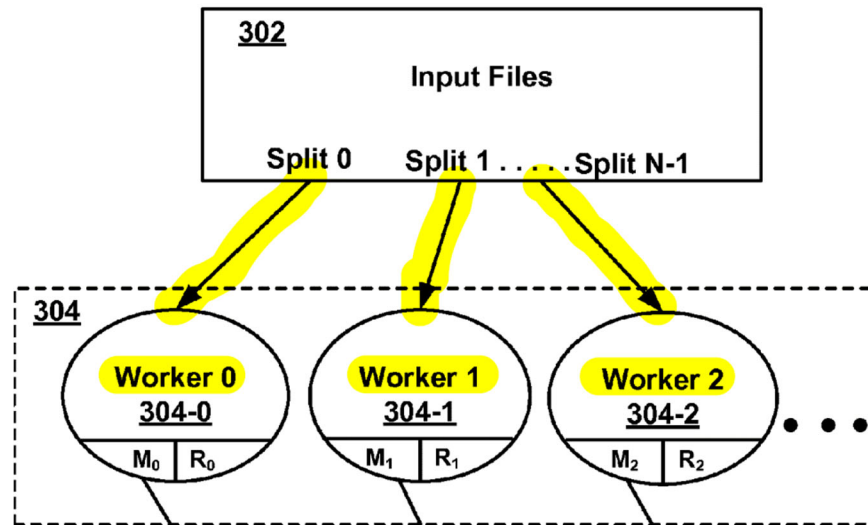
The combination would have been nothing more than combining known elements

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. 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 efficient, fault-tolerant architecture to do so.

d. [1d] “providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable”

104. The '610 patent admits that this feature was conventional by disclosing “conventional” MapReduce provided partitioned data to a “corresponding invocation of the map function.” '610 patent, 2:33-35, Fig. 1.

105. Pike also discloses this feature (**Grounds 1 and 2**) by disclosing that 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, Fig. 3; *see also id.*, 14:57-63.



Id. at Fig. 3 (annotated).

106. 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 (emphasis added). The output is “collected and written to one or more intermediate files 306.” *Id.*

107. Pike discloses an exemplary map operator that includes “counting the number of occurrences of a term[] to provide intermediate data values.” *Id.*, 14:58-63, 10:36-39. Pike’s application of a map task to each “respective input data block” is consistent with the ’610 patent, which discloses portions of its tables are provided to specific invocations of mapping functions, and each mapping function provides intermediate data. *Id.*, 10:4-7; ’610 patent, 8:15-24, Fig. 5.

108. Pike also discloses that its map operators are “user-configurable.” Specifically, Pike discloses that “an application programmer can create a script or

program using the application software 422, which includes one or more [map, reduce, and partition] operators 424, 426 and 428.” Pike, 14:35-39, 14:12-27; *see id.*, 4:28-53 (same), 5:26-39, 8:34-45. Pike also discloses that its map operators are “application-specific” and thus developed by programmers based on the use case. *Id.*, 2:22-26, 2:31-36, 3:59-64, 4:28-53. This is consistent with my understanding of map operators. *See* Section V.A (“Overview of MapReduce”), V.B (“State of the Art of Relational Processing”), V.C (“Overview of the ’610 Patent”). Pike’s disclosure of map operators, developed by application programmers, teaches “two or more mapping functions that are each configurable by a user.” EX-1007, 13-15.

109. 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.”

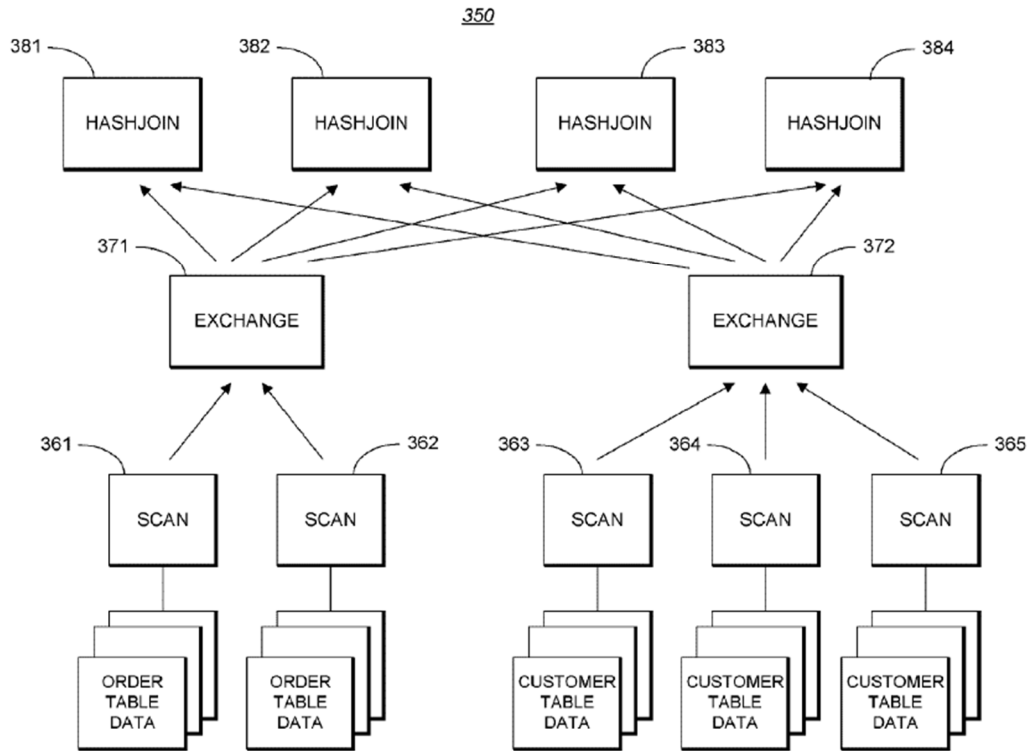


FIG. 3B

Id., Fig. 3B.

110. As I 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. *See supra* ¶¶ 80, 95. 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 (“[I]n some embodiments, the 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.); ’610 patent, 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.”); *see also* Sections V.A. (“Overview of MapReduce”), V.B. (“State of the Art of Relational Processing”), V.C. (“Overview of the ’610 Patent”), VII.A (“Overview of Pike”).

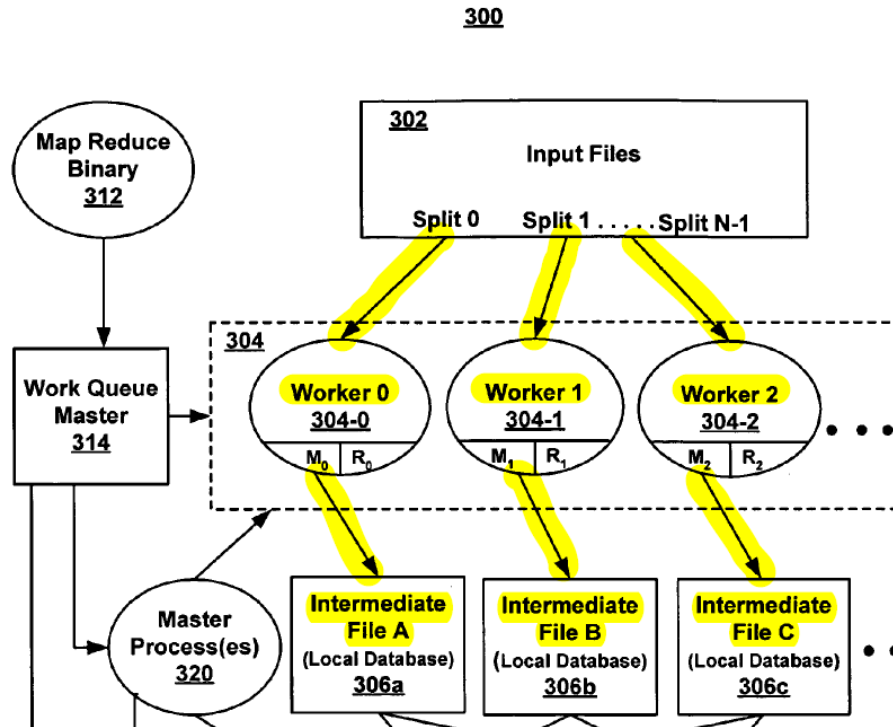
111. In my opinion, Chowdhuri therefore teaches or renders obvious providing each data partition to a selected one of a plurality of mapping functions. Chowdhuri’s scan iterators are user-configurable because Chowdhuri discloses user-defined pseudo code for scanning customer and order tables for customer 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.

112. In my opinion, it would have been obvious for a POSITA to implement Chowdhuri’s scan iterators as Pike’s map operators in Pike’s system 300. A POSITA would have done so, for example, to identify customer and order

states based on Chowdhuri's input tables. 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. A POSITA would have been motivated to do so, as discussed above, to leverage Pike's distributed system and fault tolerant task management. *See* Sections VIII.A.1.b-c (claim 1, elements [1b]-[1c]).

- e. **[1e] “to independently output a plurality of lists of values for each of a set of keys found in such map function’s corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,”**

113. In my opinion, Pike discloses “independently output[ing]” intermediate data (**Grounds 1 and 2**) by each worker process 304 “invok[ing] 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 (emphasis added), Fig. 3. Figure 3 depicts each worker process 304 independently outputting intermediate data to intermediate files 306 in Figure 3. *Id.*, Fig. 3 (below). In my opinion, a POSITA would have understood that this teaches map functions “independently outputting” intermediate data because each worker process 304 invokes a map task to individually output intermediate data.

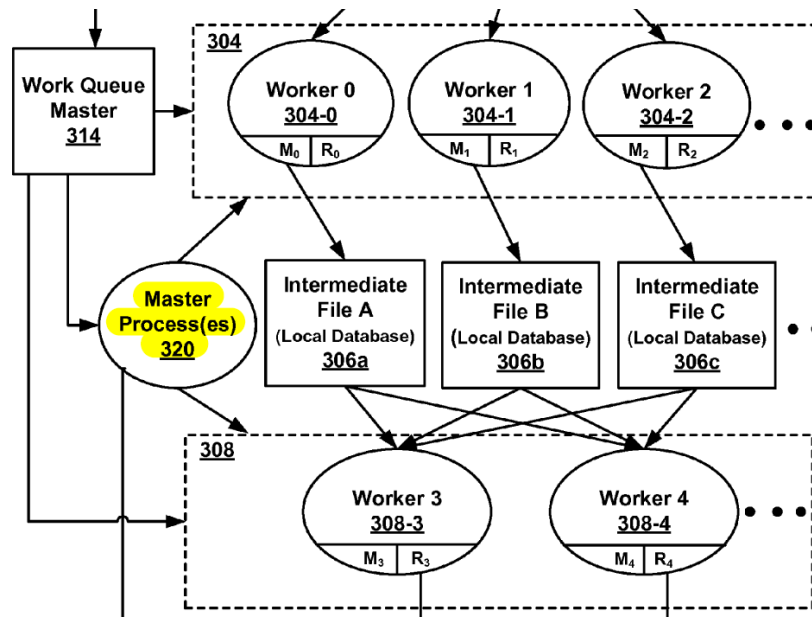


Id., Fig. 3 (annotated).

114. 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 discloses that “[t]he input data blocks 0, . . . , N-1, which may . . . 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.* (emphasis added). Each data block (i.e., “split”), as shown above in Figure 3, is provided to a specific worker process 304 for application of a map operator. *Id.*, Fig. 3. 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 <w, 1>.” *Id.*, 10:31-11:4 (“Each map task in this example will produce hundreds or thousands of records of the form <word, 1>.”); *see also id.*, 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. In my opinion, this teaches outputting lists of values for each of a set of keys found in a map function’s corresponding data partition, because it is outputting values based on a key for each “respective input data block.”

115. Pike additionally discloses that its map operators form “corresponding intermediate data for [a] data group and identifiable to that data group,” as claimed, 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). 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).

116. 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, as depicted below, map and reduce tasks are tracked according to “Input files” (i.e., Pike’s “data groups”). *Id.*, Fig. 7A. Pike discloses that “the input files field 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. In the status table of Figure 7A (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.

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. (annotated).

117. 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.” The intermediate data for input file “12340” is stored in output file 12341. *Id.* 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. Pike’s use of file name identifiers to associate intermediate data “for that data group and identifiable to that data group” is not meaningfully different from the ’610 patent’s

use of identifiers to track its Employee or Department tables. '610 patent, 4:1-22, Fig. 4.

118. 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. Pike's status tables 326 suggest doing so, as explained above. Data identifiers were not novel at the time of the '610 patent. *See* Section V.B ("State of the Art of Relational Processing"); EX-1024, Abstract, 5:30-42, 5:53-60, Fig. 6. 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. The addition would have involved basic encoding of an identifier for each input file and tracking those identifiers.

119. 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. This could later be used in a post-processing operation to rerun reduce tasks on a subset of intermediate data. For example, particular input files could be omitted from the intermediate data before being sent to the reduce tasks.

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. Particular input files would instead be excised from the intermediate data before reducing.

120. 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.

121. 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.” 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 merged by customer state and customer order based on different input data to yield orders by state. Chowdhuri ¶¶126-27, 131, 135. The customer state and customer order are therefore identifiable to each input table.

122. 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. 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. 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. 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. 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. A POSITA would have combined Chowdhuri and Pike for the reasons explained above to scan and merge these input tables in Pike, as Chowdhuri suggests.

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

123. In my opinion, this feature is obvious over Pike (**Grounds 1 and 2**), and obvious in view of Pike and Chowdhuri (**Ground 2**).

124. This element recites the data of a first data group having a different schema than that of a second data group. The '610 patent describes a data set's “schema” as a “set of attributes (such as DeptID, LastName, DeptName) and their properties (such as their data types: integer DeptID, string LastName, string DeptName).” '610 patent, 3:35-47. The one example of data groups having different schema in the '610 patent is a pair of tables with a dissimilar field (i.e., “EmpName” versus “DeptName”). *Id.*; *see also* Fig. 3.

125. The '610 patent acknowledges that “conventional MapReduce” could handle data with “different schemas,” but only alleges that it could not do so “efficiently” or “[p]ractical[ly].” *Id.*, 3:9-20; *see* Sections V.A (“Overview of MapReduce”), V.B (“State of the Art of Relational Processing”), V.C (Overview of the '610 Patent”). This is confirmed by the inventors' own statements, who acknowledged that MapReduce could process data “from multiple sources” and track the same with a “data-source attribute.” EX-1022, 1029-30; *see* Sections V.A (“Overview of MapReduce”), V.B (“State of the Art of Relational Processing”), V.C (Overview of the '610 Patent”). This confirms my opinion that the claims of

the '610 patent are nothing more than a use-case of MapReduce that specifies particular data in different input data groups. Pike teaches identifying particular data as a “data group” and tracking that data. Pike, 6:55-7:12, 13:7-17, Fig. 7A. Indeed, this was ubiquitous in the art. *See* Section V.B (“State of the Art of Relational Processing”); EX-1024, Abstract, 5:30-42, 5:53-60, Fig. 6; EX-1025, Abstract, ¶¶5, 8, 26. In addition to being admittedly conventional, Pike discloses or at least suggests mapping and reducing data of different schema.

126. Pike (**Grounds 1 and 2**) discloses that its input files “can include a variety of data types typically used in data processing systems, including *without limitation* text files, . . . , tables and the like.” Pike, 10:1-7 (emphasis added). Pike further discloses an exemplary MapReduce use-case of analyzing “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. In my opinion, a POSITA would have understood that Pike and MapReduce could handle diverse, heterogeneous data sets. *See id.*, 10:1-7, 2:17-41, 10:31-48; *supra* Sections V.A (“Overview of MapReduce”), V.B (“State of the Art of Relational Processing”).

127. To the extent it is argued that Pike does not expressly disclose processing data groups having different schema, a POSITA would have found it obvious to implement Pike with input files of this character (**Grounds 1, 2**). A

POSITA would have understood that Pike discloses or at least suggests applying its techniques to input files having different attributes, including “text files” and “tables.” Pike, 10:1-7. Implementing Pike’s large-scale data processing system 300 with input files of different schema would be nothing more than implementing Pike in a manner that it suggests: to analyze “a variety of data types.” *Id.* This would have yielded predictable result in a system that analyzes input files of different schema, partitions those input files, and tracks the files through the map and reduce phases.

128. A POSITA also would have recognized that Pike’s disclosure of application-specific map and reduce operators, developed by programmers to be application dependent, suggests map and reduce processing based on use case. Pike, 2:22-26 (disclosing “application-specific mapping functions”), 2:31-36 (same), 3:59-64 (same), 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.”). This is consistent with a POSITA’s understanding of the map and reduce operators of MapReduce. *See* Section V.A (“Overview of MapReduce”). Pike discloses that, “to perform a specific data processing operation on a set of input files, the only information that must be provided by an application programmer is: information identifying the input file(s), information identifying or

specifying the output files to receive output data, and two application-specific data processing operators, hereinafter referred to as map() and reduce().” Pike, 4:33-39, 6:1-12 (same), 10:4-7 (same), 10:39-50 (same), 11:36-38 (same), 14:19-27 (same). A programmer therefore designs the map and reduce functions to operate in a particular way for different applications and different input files based on a given use case or purpose of the analysis.

129. Analyzing input files of a diverse schema, including type or format, would have been one known use-case and obvious implementation of Pike’s system 300. 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. Pike, 10:31-11:4, 4:28-53. A POSITA also would have found it obvious to, for example, map a variety of different document types to search for particular words, as Pike discloses. Pike, 10:31-11:4. This is the very nature of MapReduce and Pike’s system 300: analyzing large, diverse collections of documents using application-specific functions.

130. To the extent asserted otherwise, in my opinion that POSITAs knew about, were implementing, and were using MapReduce, as disclosed in Pike, to analyze large data sets well before the ’610 patent. Indeed, the inventors themselves conceded that MapReduce could analyze data of different schema by simply tracking input files. *See supra* Sections V.A (“Overview of MapReduce”),

V.C (“Overview of the ’610 Patent”). 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. Mapping input files of different schema in Pike therefore would have been routine for a POSITA. 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.

131. 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. As discussed above in Element [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). *See* Section VIII.A.1.e (claim 1, element [1e]). This would have improved Pike’s efficiency and reduced computation time by avoiding rerunning the map process on the subset of input files.

132. The ’610 patent alleges an “efficien[cy]” or “[p]ractical[ly]” limitation to mapping and reducing these input files in MapReduce. *See* ’610 patent, 3:15-20. But this does not mean that MapReduce could not map and reduce data from

heterogeneous sources. Indeed, nothing in the background section of the '610 patent (*id.*, 1:6-27), the admitted prior art (*id.*, 1:66-3:15), or the Dean and Ghemawat paper (EX-1009) suggests MapReduce or Pike was so limited. Instead, each discloses using application-specific operators to map and reduce large collections of documents, including web pages scraped from the Internet. *See* Section V.A (“Overview of MapReduce”), V.B (“State of the Art of Relational Processing”). Indeed, a POSITA would have understood documents scraped from the Internet include different data files of schema.

133. In addition to Pike, this claim feature is also obvious in view of Pike and Chowdhuri (**Ground 2**). 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.

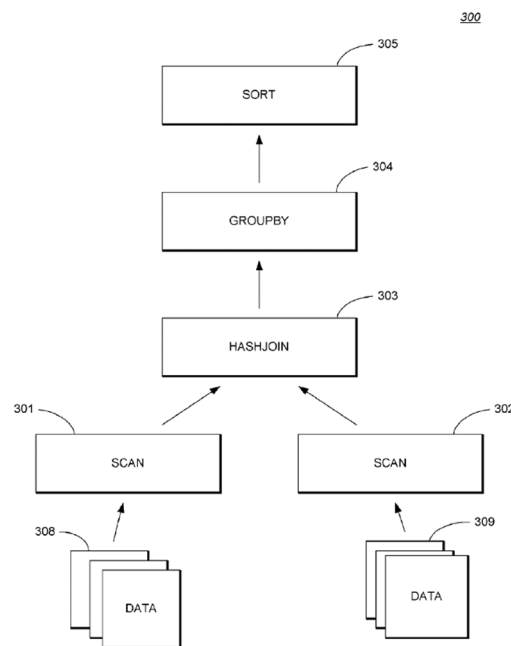


FIG. 3A

Id., Fig. 3A.

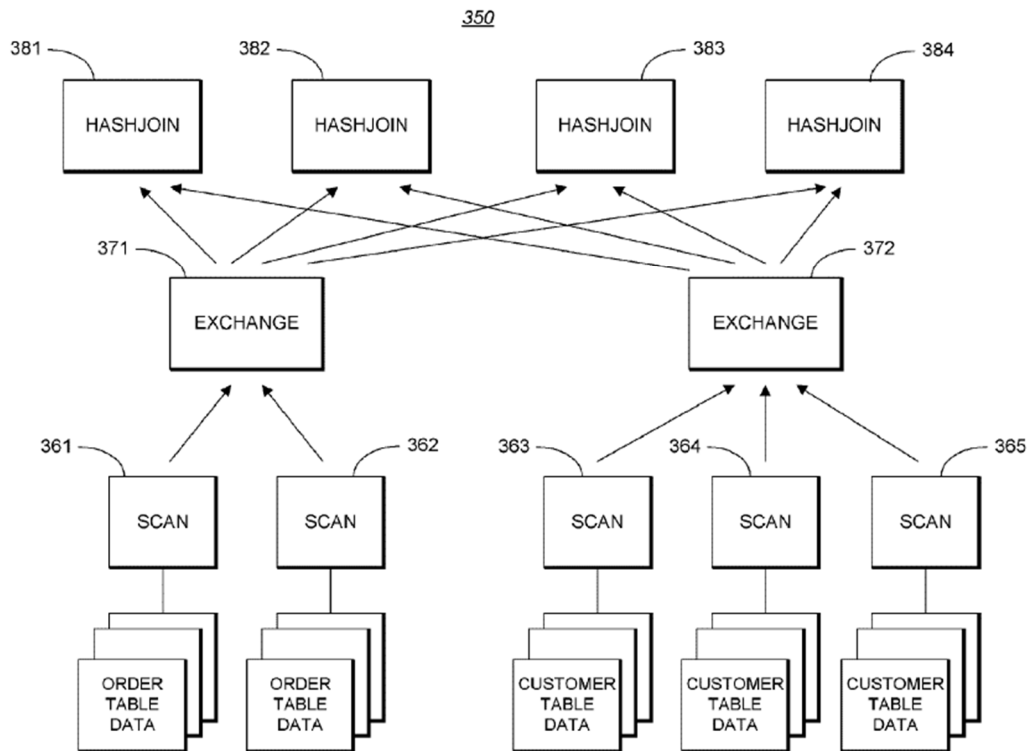


FIG. 3B

Id., Fig. 3B.

134. 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.* ¶¶130-135, Figs. 3A-3B; *See* Sections VIII.A.1.b-c (claim 1, elements [1b]-[1c]). Chowdhuri's order and customer tables are data groups as discussed in element [1b]. *See* Section VIII.A.1.b (claim 1, element [1b]). These data groups have different schema in order data and customer data. Chowdhuri, ¶¶127, 131, 135, Figs. 3A-3B. This is consistent with the '610 patent's employee

and department tables, which the patent contends are exemplary tables having different schema. '610 patent, 3:23-34, 3:65-4:13, Fig. 3.

135. A POSITA would have found it obvious to combine the teachings of Chowdhuri with Pike's system 300, as discussed above. 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. Pike and Chowdhuri disclose similar mapping and reducing operations, for similar purposes. 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.

136. 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. The result would have incorporated well-known table

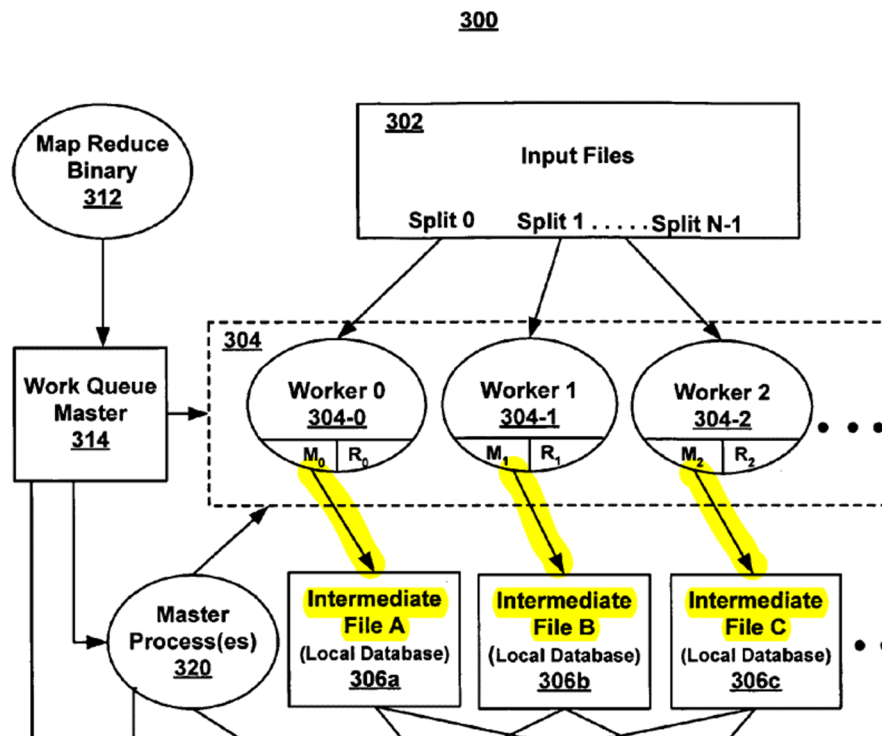
processing techniques in RDBMS and SQL queries, as taught by Chowdhuri, into MapReduce, based on Chowdhuri's suggestion of employing a similar set of serial map (e.g., scan) and reduce (e.g., merge) processes. As discussed above, implementing Pike and Chowdhuri to track input table identifiers, as taught 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. This would have allowed, for example, the omission of particular tables from the final merge result in the Pike-Chowdhuri system. 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.

- g. [1g] "and the data of the first data group is mapped differently than the data of the second data group so that different lists of values are output for the corresponding different intermediate data,"**

137. I understand that the district court held that "mapped differently" in claim 1 does *not* require different map *functions*. EX-1007, 16-18. Instead, different instantiations of the same map function, like that provided in the '610 patent's description of "conventional" MapReduce (*see* '610 patent, Figs. 1, items 104(1)-104(7)), satisfies this limitation.

138. Pike (**Grounds 1 and 2**) discloses or renders obvious this element. As shown in Figure 3 (below), Pike discloses or at least renders obvious mapping

differently by disclosing that each of its data blocks (i.e., “split[s]”) are provided to a different worker process 304-0, 304-1, and 304-2, each of which invoke a map operator, M_0 , M_1 , and M_2 , 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.”). Figure 3 illustrates this:



Id., Fig. 3 (annotated).

139. 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. In certain embodiments, “each of the intermediate files 306 receives output from **only one** worker process 304, as shown in FIG. 3.” *Id.*, 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. ’610 patent, Figs. 4, 5.

140. To the extent “mapped differently” is construed to require invoking different mapping **functions** during the map phase, Pike still renders this feature obvious. As described in Element [1f], Pike discloses that the map functions are application-specific and developed for specific use-cases. *See* Section VIII.A.1.f (claim 1, element [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 further discloses computer system 400 for carrying out the data processing of system 300, and states that system 400 includes application software 422. Pike, 13:34-14:27, Fig. 4. The application software 422 includes “application-specific [map] **functions** for producing output.” *Id.* (emphasis added). This would have informed a POSITA that Pike contemplates multiple map functions, and that these map functions are invoked for processing particular data depending on the use-case, including using **different** map functions for a given map phase (e.g., for different input files). 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.” *Id.*, 3:59-61 (emphases added).

141. 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 and states. Chowdhuri ¶¶ 126-28, 131-35, Fig. 3A, 3B. As shown in Chowdhuri’s 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 VIII.A.1.b, 1.d (claim 1, elements [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).

142. 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. As explained in Elements [1b]-[1f], a POSITA would have been motivated and found it obvious to combine Chowdhuri and Pike, with expected results. *See* Sections VIII.A.1.b-f (claim 1, elements [1b]-[1f]).

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

143. The ’610 patent concedes that “conventional” MapReduce’s input data and intermediate data shared a common key. ’610 patent, 1:17-27 (“A ‘reduce’ function represents all mapped (intermediate) key-value pairs sharing the same key to a single key-value pair or a list of values.”), 2:12-51 (referring to Figure 1 and identifying common keys “k1 to k5”), Figs. 1, 2.

144. As explained in Element [1f], Pike (**Ground 1**) or Pike in view of Chowdhuri (**Ground 2**) teach input files having “different schema.” *See* Section VIII.A.1.f (claim 1, element [1f]). Pike discloses its input data and intermediate data have a key in common (**Grounds 1 and 2**) by disclosing that each 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 (emphasis added), 14:65-15:3. To reduce on “the same key,” the input data and intermediate data must have “a key in common.”

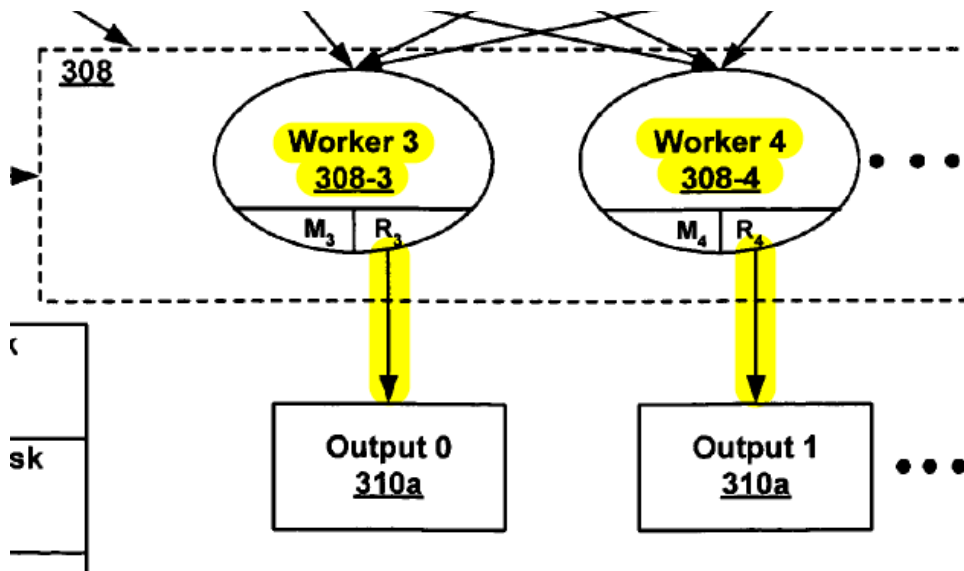
145. Chowdhuri alternatively discloses this (**Ground 2**) by disclosing that its customer table 308 and order table 309 have a common key in the “customer_id,” which enables identifying customer “state” (i.e., within the United States) with customer orders. Chowdhuri ¶¶126-27 (“where o.customer_id=c.customer_id,” “group by state,” and sorting “orders for each of the states”), 131, 135. 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 VIII.A.1.e-g (claim 1, elements [1e]-[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. 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 I explained in elements [1b]-[1f], a POSITA would have been motivated and found it obvious to combine Chowdhuri and Pike, with expected results. *See* Sections VIII.A.1.b-f (claim 1, elements [1b]-[1f]).

- i. **[1i] “reducing the intermediate data for the data groups to at least one output data group, including processing the intermediate data for each data group in a manner that is defined to correspond to that data group, so as to result in a merging of the corresponding different intermediate data based on the key in common,”**

146. Pike discloses or renders these features obvious (**Grounds 1 and 2**).

For example, Pike discloses its worker processes 308 invoke reduce operators to reduce its 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.* (emphases added). 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.* (emphasis added). 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.”

147. In Figure 3 of Pike, just like the '610 patent in its Figures 4 and 5, Pike illustrates its worker processes 308 receiving intermediate data from the intermediate files 306, and merging them into outputs:



Id., Fig. 3 (annotated).

148. As explained in elements [1b] and [1f], Pike teaches or renders obvious “data groups.” *See* Sections VIII.A.1.b, 1.f (claim 1, elements [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. 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 VIII.A.1.e (claim 1, element [1e]).

149. Pike in view of Chowdhuri (**Ground 2**) also renders this feature obvious by disclosing joining, grouping, and sorting by customer and order “state,” corresponding to the customer and order table. *See* Section VIII.A.1.h (claim 1, element [1h]). Specifically, 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.

150. As I 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. *See* Section VIII.A.1.d (claim 1,

element [1d]). Chowdhuri's iterators also correspond to the description of the reducing operations disclosed in the '610 patent. For example, the '610 patent discloses that "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." '610 patent, 4:56-58, 5:36-59; *see also id.*, 1:20-27 ("[t]he reduce operation ... combines elements"), 5:19-39 (sorting), 2:65-66 ("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").

151. 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. A POSITA would have done so to merge data from Chowdhuri's customer and order states, as Chowdhuri discloses and suggests. 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. A POSITA would have been motivated to do so, as discussed above, to leverage Pike's distributed system and fault tolerant task management.

152. To the extent claim 1 requires different reducing *functions*, or reducing each intermediate file differently, Chowdhuri discloses or at least renders this feature obvious. Chowdhuri's joining, grouping, and sorting iterators process intermediate data associated with customer and order tables, and join, group, and/or sort this intermediate data based on customer, order, and state. This teaches reducing "differently" or employing different reduce functions—specific to each input file—just as the '610 patent describes reducing intermediate data from an employee and department table. *Compare* Chowdhuri ¶¶126-27, 131, 135, with '610 patent at Figs. 4, 5.

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

153. As described in Element [1a], Pike's computer system 400 for use with processing system 300 discloses Pike's mapping and reducing operations being performed by a distributed system. *See* Section VIII.A.1.a (claim 1, element [1a]); Pike, 13:53-14:27, 9:20-28. System 300 "distribut[es] file reads over multiple local databases," where "smaller tasks are spread across many machines." Pike, 13:37-50; *see also id.*, 1:20-35 ("distribution of data and computations among multiple disks and processors"), *id.* ("automatically parallelizing these operations across multiple processors in a *distributed and parallel processing environment*.")) (emphasis added). Pike also expressly discloses that "distributed data processing system 500" on which "the embodiments described here can be

performed.” *Id.*, 2:43-51. Chowdhuri also teaches this feature. (**Ground 2**). *See*

Section VIII.A.1.a (claim 1, element [1a]); Chowdhuri ¶¶105-08, 128, 131-35.

2. Claim 17

- a. [17a] “A computer system including a plurality of computing devices, the computer system configured to process data of a data set, wherein the data set comprises a plurality of data groups, the computer system comprises at least one processor and memory that are operable to perform the following operations:”**

154. In my opinion, Pike’s computer system 400 for carrying out the processes of system 300 discloses a computer system having a plurality of computing devices with at least one processor and memory. *See* Section VIII.A.1.a (claim 1, element [1a]), VIII.A.1.j (claim 1, element [1j]); *see also* Pike, 1:20-35; 2:43-3:35, 7:13-25, 11:57-12:33. Pike’s computer system processes a data set that includes plurality of data groups in Pike’s input files, as discussed in Section VIII.A.1.b (claim 1, element [1b]).

- b. [17b] “partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and”**

155. Pike and Pike-Chowdhuri disclose and render this feature obvious for the reasons explained in Section VIII.A.1.c (claim 1, element [1c]).

- c. **[17c] “providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable”**

156. Pike and Pike-Chowdhuri disclose and render this feature obvious for the reasons explained in Section VIII.A.1.d (claim 1, element [1d]).

- d. **[17d] “to independently output a plurality of lists of values for each of a set of keys found in such map function’s corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,”**

157. Pike and Pike-Chowdhuri disclose and render this feature obvious for the reasons explained in Section VIII.A.1.e (claim 1, element [1e]).

- e. **[17e] “wherein the data of a first data group has a different schema than the data of a second data group”**

158. Pike and Pike-Chowdhuri disclose and render this feature obvious for the reasons explained in Section VIII.A.1.f (claim 1, element [1f]).

- f. **[17f] “and the data of the first data group is mapped differently than the data of the second data group so that different lists of values are output for the corresponding different intermediate data,”**

159. Pike and Pike-Chowdhuri disclose and render this feature obvious for the reasons explained in Section VIII.A.1.g (claim 1, element [1g]).

- g. [17g] “wherein the different schema and corresponding different intermediate data have a key in common; and,”

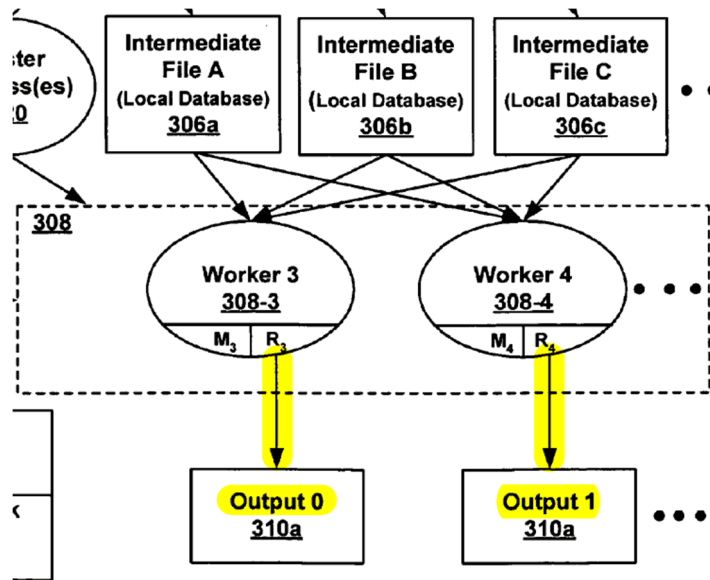
160. Pike and Pike-Chowdhuri disclose and render this feature obvious for the reasons explained in Section VIII.A.1.h (claim 1, element [1h]).

- h. [17h] “reduce the intermediate data for the data groups to at least one output data group, including processing the intermediate data for each data group in a manner that is defined to correspond to that data group so as to result in a merging of the corresponding different intermediate data based on the key in common.”

161. Pike and Pike-Chowdhuri disclose and render this feature obvious for the reasons explained in Section VIII.A.1.i. (claim 1, element [1i]).

- 3. Claims 2 and 18 — “wherein: the at least one output data group is a plurality of output data groups.”

162. In my opinion, claims 2 and 18 do not require the data groups in the map phase to be the same as the “output data groups.” The claims only require reducing intermediate data to a plurality of “output data groups.” The specification confirms this: “the output group identifiers may be *different* from the input/intermediate group identifiers.” ’610 patent, 7:18-23 (emphasis added). 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.



Id., Fig. 3 (annotated).

163. Chowdhuri alternatively discloses this (**Ground 2**) by disclosing more than one output from its scan iterators. Chowdhuri ¶¶132-35, Fig. 3B. 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 VIII.A.1.b (claim 1, element [1b]).

4. **Claims 3 and 19 — “wherein: corresponding intermediate data for a data group being identifiable to the data group includes the corresponding intermediate data being stored such that metadata associated with a manner of storage is identifiable to the data group.”**

164. The ’610 patent references “metadata” in only two paragraphs, ’610 patent, 5:19-32, 7:31-35, stating that, “the operations can collect information about the data that they are processing” and “save the information as metadata.” One

example in the '610 patent is saving *keys* associated with a partition in the intermediate data. *Id.*, 5:22-32.

165. In my opinion, Pike (**Grounds 1, 2**) discloses saving data associated with each partition's key in its intermediate files 306. 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.

166. A POSITA would have understood Pike's processing each intermediate data files 306 in accordance with the key of the intermediate data teaches or at least suggests storing keys "in each partition processed." '610 patent, 5:22-24. 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." *Id.*

167. Pike's master process 320 tracking associated 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 (e.g., "data groups") with output files, including intermediate files. *See* Section VIII.A.1.e (claim 1, element [1e]). This teaches metadata "identifiable to the data group."

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).

168. To the extent it is argued that the above disclosure does not expressly disclose metadata, it would have been obvious to implement these features in Pike’s intermediate data files 306 to include metadata about its keys. 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. Indeed, it was well known in the art to use identifiers or metadata to track data in a system like Pike. *See* Section V.B (“State of the Art of Relational Processing”), V.C (“Overview of the ’610 Patent”); EX-1024, Abstract, 5:30-42, 5:53-60, Fig. 6; EX-1025, Abstract, ¶¶5, 8, 26, 27 (disclosing “record[ing] changes to an information system” using metadata). The

same result would follow in Pike's system 300: the worker process 308 would access the keys from metadata associated with the intermediate data to reduce the intermediate data into output data, or master process 320 would access status information metadata to track and assign tasks.

169. 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.

170. 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., an input file or "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 the lineage of the intermediate data through the mapping and reducing. A POSITA would have been motivated to do so, as discussed above, to allow input files to be individually tracked and optionally removed from the reduce phase. *See* Section VIII.A.1.e (claim 1, element [1e]) ("This would improve efficiency of the

system.”). The result would have been predictable in attaching an identifier, or metadata, to the intermediate data for selective reducing.

5. Claims 4 and 20 — “store[] [the corresponding intermediate data] in association with data that is identifiable to the data group.”

171. Pike (**Grounds 1 and 2**) discloses or renders this feature obvious for the reasons explained in Section VIII.A.1.e (claim 1, element [1e]) Pike additionally teaches this feature (i.e., “data”) for the reasons explained in claims 3 and 19, which recite “metadata.” *See* Section VIII.A.4 (claims 3 and 19).

6. Claims 5 and 21 — “process[ing] the intermediate data” “includes, for each data group,” “employ[ing] an iterator that corresponds to that data group.”

172. In my opinion, a POSITA would understand that an “iterator” includes a software object that accesses or traverses a data structure and applies a function to elements in the data structure. The patent describes these as “useable within reduce functions *to access* intermediate data.” ’610 patent, 3:58-62 (emphasis added). This corresponds with Chowdhuri’s description of “iterators,” which are “self-contained software objects that accept a stream of rows from null or n-ary data sets.” Chowdhuri ¶61. Regardless of the function applied, iterators “all open data streams, iteratively read the streams, process it and close the streams.” *Id.*

173. Pike's worker process 308 (**Ground 1**) 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"). A POSITA would have known that Pike's reducer operators are iterators because the inventors of Pike described them this way in the Dean and Ghemawat paper, which is cited by the '610 patent:

The reduce worker *iterates* over the sorted intermediate data and for each unique intermediate key encountered, it passes the key and the corresponding set of intermediate values to the user's *Reduce* function.

EX-1009, 4; *see also* Section VIII.A.1.f (claim 1, element [1f]). Indeed, a POSITA would have understood that a reducer is an iterator because it iterates over data records. Pike's worker processes 308 "corresponds to that data group" because it corresponds to the input file, as explained in Section VIII.A.1.i (claim 1, element [1i]).

174. Chowdhuri also discloses this feature (**Ground 2**) by employing a hash join iterator, GroupBy iterator, and sort iterator for reducing the results of a scanning operation. Chowdhuri ¶¶129-35, Figs. 3A, 3B; *see also* Section VIII.A.1.f (claim 1, element [1f]). A POSITA would have combined Chowdhuri and Pike for the reasons explained in Sections VIII.A.1.b-f (claim 1, elements [1b]-[1f]).

175. A POSITA would additionally have been motivated to implement Pike's worker process 308 and reduce operators as an iterator because Pike suggests it, Chowdhuri discloses operations that to a POSITA would have understood were comparable to map and reduce processes using iterators, and for the reasons explained in Chowdhuri: "iterators are self-contained entities that maintain their own states, and they can be easily duplicated for horizontal parallelism across CPU and I/O threads." Chowdhuri ¶131. Implementing Pike's worker process 308 as an iterator would therefore promote parallelism in system 300, and thus reduce run times.

7. Claims 6 and 23

- a. [a] "wherein: the intermediate data includes a plurality of grouped sets of key/value pairs" and reducing via "a plurality of reducers;"**

176. Pike's intermediate data files 306 including key/value pairs discloses this feature (**Grounds 1 and 2**). Pike, 11:27-31; *see also id.*, 7:34-52 ("[M]ap() operator . . . generate[s] its output data in the form of key/value pairs."), 10:31-58 (same). A POSITA would have understood that an intermediate data file 306 that includes key/value pairs is a "grouped set[]" of key value pairs" because it is a group of data that includes key/value pairs. Chowdhuri also discloses this (**Ground 2**) in the results of its scanning iterators, which include customer keys and values (customer state and order data). Chowdhuri ¶¶126-35, Figs. 3A-3B.

177. Pike's plurality of worker processes 308 that invoke reduce operators also discloses a reducing step carried out by a plurality of reducers (**Grounds 1 and 2**). Pike, 8:65-28 ("one or more worker processes 304 and 308"), 11:6-55, 13:5-17, 14:65-15:6, Fig. 3 (items 308-3, 308-4). 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. **[b] "partition[ing] the intermediate data into a plurality of partitions, [to provide] each key/value pair of the intermediate data . . . to a separate one of the partitions; and provid[ing] the intermediate data of each partition to a separate one of the reducers."**

178. In my opinion, Pike (**Grounds 1 and 2**) discloses this feature by disclosing a partition operator that "may specify 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. In my opinion, a POSITA would have understood from this that Pike's partition operator partitions intermediate data into a plurality of partitions, each having separate key/value pairs.

179. 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. In my opinion, a POSITA would have understood that each of Pike's worker processes 308 "read[ing] a corresponding one of the intermediate files 306 produced by all or

a subset of the worker processes 304” teaches providing the intermediate data of each partition to a separate one of the reducers. *Id.*

180. To the extent it is argued Pike does not expressly disclose partitioning intermediate data by key and providing each partition to a reduce function (Pike does disclose this), Pike renders this feature obvious in my opinion. Pike discloses that the partition operator, like the map and reduce operators, are application-specific and determined by the user and use-case. Pike, 14:12-27, 4:49-53.

Implementing Pike such that each intermediate file includes only one group of key-value pairs, and providing each partition to a different reduce function is one implementation a POSITA would have found obvious from Pike’s own disclosure. The ’610 patent also admits this feature was known and conventional: “[e]ach key/value pair go[ing] to only one of the partitions . . . is similar to what is done by the conventional MapReduce.” ’610 patent, 4:63-67; *see also id.*, 2:61-64 (partition functions 206a, 206b, 206c of “conventional” MapReduce), Fig. 2. A POSITA would therefore have understood that this feature was known, based on experience with MapReduce and by the ’610 patent’s admissions, and that it was taught and suggested by Pike for implementation in system 300.

181. Chowdhuri also 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.

8. Claims 7 and 24

- a. [a] “the intermediate data includes a plurality of grouped sets of key/value pairs; the reducing step is carried out by a plurality of reducers;”

182. Pike discloses this feature for the reasons explained in Section VIII.A.7.a (claims 6 and 23, element [a]).

- b. [b] “partition[ing] the intermediate data into a plurality of partitions, [to provide] at least some of the key/value pairs of the intermediate data . . . to more than one of the partitions; and provid[ing] the intermediate data of each partition to a separate one of the reducers.”

183. Pike’s partition operator discloses partitioning the intermediate data into a plurality of partitions and providing each partition to a reducer. *See* Section VIII.A.7.b (claims 6 and 23, element [b]).

184. In my opinion, 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).

185. Pike (**Grounds 1 and 2**) discloses providing some of the key/value pairs to more than one partition by disclosing the partition operator may be “a modulo function” where the application programmer “specifies the modulus to be used by the modulo function.” Pike, 7:52-55. In my opinion, a POSITA would have understood that this teaches dividing the intermediate data between the intermediate data files based on a modulus, rather than dividing them by key. This teaches providing key/value pairs of the intermediate data to multiple partitions.

186. To the extent claims 7 and 24 are understood to require providing the same key/value pair to more than one partition, this would have been an obvious implementation of MapReduce in Pike (**Grounds 1 and 2**). A POSITA would have understood that, to make a particular key-value pair go to more than one partition, the map operation would simply emit the same key/value pair multiple times. Repeating the same process twice is not new and would yield expected results: emitting the same key/value pair more than once, each of which would then be provided to a separate partition for reduction. This is a straightforward application of a map function in “conventional MapReduce” that a POSITA would have found obvious to implement.

187. A POSITA also would have found this implementation obvious based on the suggestion of Chowdhuri (**Ground 2**), which discloses the use of exchange operators 371, 372. Chowdhuri ¶¶128-35, Fig. 3B. The exchange operators “take[] the data from the producer tasks and is capable of redistributing data for a different degree of parallelism in its consuming tasks.” *Id.* ¶134. Chowdhuri discloses that the redistribution “is extended to redistribution of data to *multiple servers* on a cluster for processing the data.” *Id.* (emphasis added).

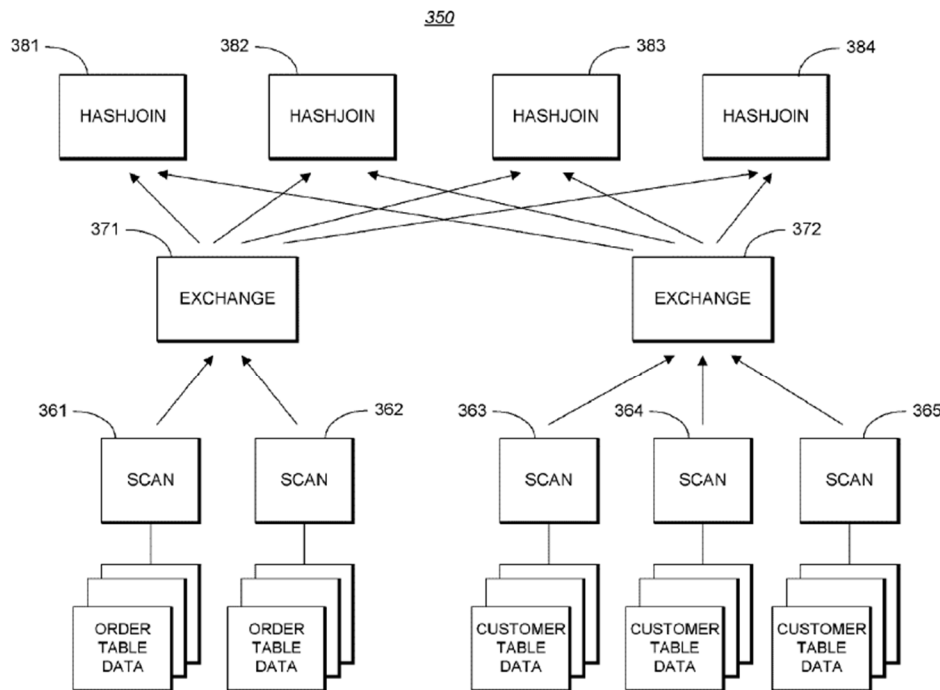


FIG. 3B

Id., Fig. 3B.

188. In certain embodiments where a “replicated join” is requested, Chowdhuri expressly discloses that “*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.” *Id.* ¶227 (emphases added). A POSITA therefore would have understood that it was known to send values, including key/value pairs, to more than one partition for processing, and would have been motivated to do so based on Chowdhuri. 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.

9. Claims 8 and 25 — “The method of claim 7, wherein: in the partitioning step all of the key/value pairs of the intermediate data are provided to all of the partitions.”

189. Claims 8 and 25 do not require all of the key-value pairs to be provided to *each* of the partitions. Accordingly, Pike (**Grounds 1 and 2**) teaches this feature by disclosing that all of its key/value pairs are provided to its partitions. *See* Section VIII.A.7.b (claims 6 and 23, element [b]).

190. To the extent claims 8 and 25 require each key/value pair to be provided to each partition, Pike renders this feature obvious (**Grounds 1 and 2**). Providing each key to each partition is known in the art as a “broadcast.” In my opinion, a POSITA would have understood that broadcasting each key/value pair to each partition would be one of several options for partitioning a set of keys. As explained for claims 6 and 23, the partition operators of Pike are application-specific and will vary with use-case. *See* Section VIII.A.7.b (claims 6

and 23, element [b]) (citing Pike, 14:12-27, 4:49-53). A POSITA would have been motivated to provide each key/value pair to each partition to increase fault tolerance. As Pike explains, machines and processes regularly fail. Pike, 4:30-39, 6:37-7:12, 9:20-28, 11:54-55. By providing each key/value pair to each partition, if one task fails, another can be used to reduce the same data and no one set of key/value pairs will be left unprocessed. Each of the key/value pairs would also be processed in parallel, by separate worker processes 308, applying reducing tasks to different partitions.

191. 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. 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. 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.

10. Claims 9 and 26

- a. [a] “wherein: at least some of the reducers include a sort, group-by-key and combine task;”**

192. In my opinion, 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 and 2**).

193. Pike discloses that its worker process 308 “sorts the intermediate data values” by key, Pike, 11:28-38; *see also id.*, 12:7-17, 14:65-15:6, 9:45-49, and “merges or otherwise combines the sorted intermediate data values having the same key,” *id.*, 11:28-38. While Pike does not expressly disclose that its worker process 308 includes a “group-by-key,” the '610 patent admits this feature was “conventional” in MapReduce. '610 patent, 2:9-67 (“conventional architecture”), 2:64-67 (“Each reduce task 206a and 206b includes a sort and group-by-key task 210a and 210b.”), 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 implemented it in Pike. Doing so would have been obvious and the implementation of a known MapReduce feature in an existing MapReduce system in Pike’s system 300. A POSITA would have been motivated to do so to make the processing of the output data “more efficient and/or convenient,” as the '610 patent concedes. '610 patent, 2:64-67.

194. Pike in view of Chowdhuri also renders this feature obvious

(**Ground 2**). As explained in claim 1, Chowdhuri discloses reducing data using a hash join iterator 303, GroupBy iterator 304, and sort iterator 305. *See* Section VIII.A.1.f (claim 1, element [1f]); Chowdhuri, ¶¶127, 130. A POSITA would have combined Pike and Chowdhuri to implement Pike's worker processes 308 as these iterators for the reasons explained in Sections VIII.A.1.a-f (claim 1, elements [1a]-[1f]).

b. [b] “generating and providing metadata for at least some of the mapping, partitioning, combining, grouping and sorting.”

195. Pike's storing “keys” in its intermediate data files 306, data regarding the associated input file, and the status of the mapping process, teaches providing metadata during at least the mapping phase (**Grounds 1 and 2**). *See* Section VIII.A.4 (claims 3 and 19) (citing Pike, 11:28-31, 6:55-7:12, Fig. 7A). So does MacLeod (**Ground 3**).

196. 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 read by that worker process in accordance with the key of the key/value pairs in the intermediate data,” Pike, 11:28-31, and “merges or otherwise combines the sorted intermediate data values” by *key* also discloses this processing, *id.*, 11:33-38 (emphasis added).

197. 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 VIII.A.4 (claims 3 and 19). *See also* Pike, 6:55-7:25, Fig. 7A. Master process 320 associates the input files (e.g., “data groups”) with output files, including intermediate files. *See* Section VIII.A.1.e (claim 1, element [1e]).

198. As explained with respect to claims 3 and 19, Pike renders obvious storing this information as metadata. *See* Section VIII.A.4 (claims 3 and 19). A POSITA would have understood that to sort and merge by key, Pike’s keys or task scheduling data would be “provid[ed]” to the combining and sorting operations.

199. A POSITA also would have understood that the metadata may also include MacLeod’s identifier (**Ground 3**), which would identify the “data group” associated with the intermediate data for subsequent grouping or sorting. *See* Section VIII.A.4 (claim 3).

11. Claims 10 and 27 — “wherein: the reducing [step] includes processing the metadata.”

200. Pike discloses this feature in several ways (**Grounds 1, 2, and 3**). Pike discloses this feature by the reducers sorting the values in the intermediate data files by key. As explained in claims 3, 9, 19, and 26, a POSITA would have understood that Pike’s “keys” in its intermediate data files 306 (Pike, 11:28-31) teaches generating and providing metadata during the mapping and reducing. *See* Section VIII.A.4 (claims 3 and 19), VIII.A.10.b (claims 9 and 26, element [b]).

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. Pike’s merging the sorted intermediate data values by key also discloses this metadata processing. *Id.*, 11:33-38. Finally, Pike’s master process 320 processing status data when reducing is complete also teaches processing the metadata. *Id.*, 11:49-55; *see also* Sections VIII.A.4 (claims 3 and 19) (citing Pike, 6:55-7:12, Fig. 7A), VIII.A.10.b (claims 9 and 26, element [b]).

12. **Claims 11 and 28 — “wherein: . . . process[ing] the intermediate data for each data group in a manner that is defined to correspond to that data group includes, for each data group, . . . employ[ing] an iterator that corresponds to that data group, wherein the iterator . . . provid[es] the associated metadata to the [processing of the] reducing [step].”**

201. For the reasons explained in claims 5 and 21, Pike’s worker process 308 (**Grounds 1, 2, and 3**), and alternatively 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 VIII.A.6 (claims 5 and 21).

202. As explained in claims 9, 10, 26, and 27, Pike (**Grounds 1, 2, and 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* Section VIII.A.10.b (claims 9 and 26, element [b]), VIII.A.11 (claims 10 and 27). Pike (**Grounds 1 and 2**), and alternatively Pike in view of Chowdhuri (**Ground 2**), therefore discloses the features of claims 11 and 28.

13. Claims 12 and 22

- a. [b] “wherein: the intermediate data includes a plurality of grouped sets of key/value pairs” and reducing step via “a plurality of reducers;”

203. Pike discloses and renders this feature obvious for the reasons explained in claims 6 and 23. *See* Section VIII.A.7.a (claims 6 and 23, element [a]).

- b. [b] “for at least one of the reducers, the iterator corresponding to a particular data group, for that reducer, operates according to a different key of a different schema than the iterator corresponding to another particular data group, for that reducer.”

204. Pike’s worker process 308 (**Grounds 1 and 2**), and alternatively 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 VIII.A.6 (claims 5 and 21). Pike renders obvious and Chowdhuri discloses

that the data groups (i.e., input files) may have different schema. *See* Section VIII.A.1.f (claim 1, element [1f]). From this, 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. This satisfies claims 12 and 22.

205. 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. In my opinion, this suggests that Pike’s sorting is performed on multiple keys across the intermediate data. Sorting multiple keys in MapReduce was also conventional, as the ’610 patent concedes in its “sort and group-by-key” tasks 210a, 201b. *See* ’610 patent, 2:9-67 (describing “conventional” MapReduce), 2:64-67 (“Each reduce task 206a and 206b includes a sort and group-by-key task 210a and 210b, respectively.”), Fig. 2.

206. In my opinion, a POSITA would have found it obvious to implement Pike’s sort process, or Chowdhuri’s sort iterator 305, to sort different keys corresponding to different data groups. Pike suggests sorting different keys and operating on different input data, like Chowdhuri. And the ’610 patent admits sorting different keys among different partitions was known. Implementing Pike’s

or Chowdhuri's sorting to sort different keys associated with different "data groups," at most would have been a straightforward modification based on the suggestions of Pike and the known features of MapReduce, which would have been well-within the capabilities of a POSITA.

207. 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. 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.

208. 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. 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. 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. The results would have been predictable.

14. Claims 13 and 29 — “wherein: the intermediate data processing step of the reducing step further comprises processing data that is not intermediate data.”

209. 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.” '610 patent, 5:66-6:7. A POSITA would have understood that this data includes data received from sources other than map functions, and includes data provided by other reduce functions.

210. Pike (**Grounds 1 and 2**) discloses this feature in several ways. 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,” teaches processing data that is not intermediate data. Pike, 11:8-27 (emphasis added). The data is received from the master process 320 and not from a map function, and therefore is not intermediate data. It is also processed by worker process 308 to locate intermediate data to reduce.

211. Chowdhuri's GroupBy and sort iterators 304, 305 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 in claims 1 and 17, a POSITA would have understood that Chowdhuri's GroupBy and sort iterators 304, 305 are reducing functions. *See* Section VIII.A.1.f

(claim 1, element [1f]), VIII.A.2.e (claim 17, element [17e]). A POSITA moreover would have understood these iterators receive data from another hash join reducing function, and that the received data is not intermediate data.

15. Claims 14 and 30 — wherein reducing is carried out via “a plurality of reducers,” and “the data that is not intermediate data, for a particular reducer, includes data that is associated with another reducer.”

212. Pike in view of Chowdhuri (**Ground 2**) discloses this feature.

Chowdhuri discloses a plurality of reducers for the reasons explained in claim 6, 13, 23, and 29. *See* Sections VIII.A.7 (claims 6 and 23), VIII.A.14 (claims 13 and 29) For the reasons 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.

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 for the reasons explained in elements [1b]-[1i]. *See* Sections VIII.A.1.b-i (claim 1, elements [1b]-[1i]).

16. Claims 15 and 31 — wherein reducing is carried out via “a plurality of reducers,” and “the data that is not intermediate data, for a particular reducer, includes data that is associated with that reducer.”

213. Pike (**Grounds 1 and 2**) discloses a plurality of reducers for the reasons explained in claims 6 and 23. *See* Section VIII.A.7.a (claims 6 and 23, element [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 VIII.A.14 (claims 13 and 29) (citing 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. It is also processed by worker process 308 that invokes a reduce function, and is data associated with that reduce function (e.g., where to locate intermediate data to reduce).

17. Claims 16 and 32 — reducing by “relating the data among the plurality of data groups.”

214. Pike (**Grounds 1 and 2**) discloses this feature by its worker processes 308 invoking reducing operators to “merge[] or otherwise combine[]” the intermediate data files 306. Pike, 4:40-44, 11:33-41. A POSITA would have understood that 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.” This is consistent with the written description of the ’610 patent, which discloses “merg[ing] all intermediate data over all groups,” including data from an employee table and department table. *See, e.g.,* ’610 patent, 4:50-59. Chowdhuri also discloses this feature (**Ground 2**) by disclosing that its hashjoin, GroupBy, and sort iterators combine the data from its customer and order tables. Chowdhuri ¶¶126-35.

18. Claim 33

a. [33a] “A map-reduce method of processing data from a plurality of groups having different schema over a computer system, the method comprising:”

215. Pike’s large-scale data processing system 300 discloses a “map-reduce method” because it maps and reduces data. Pike, 2:12-39, 2:50-3:35, 6:557:25, 8:65-11:55. System 300 processes data from a plurality of groups over a computer system for the reasons explained in Sections VIII.A.1.a (claim 1, element [1a]), VIII.A.1.j (claim 1, element [1j]), and VIII.A.1.b (claim 1, element [1b]). Pike (**Grounds 1 and 2**) and Pike in view of Chowdhuri (**Ground 2**) render obvious data groups of “different schema” for the reasons explained in Section VIII.A.1.f (claim 1, element [1f]). A POSITA would have combined Pike and Chowdhuri as explained in Sections VIII.A.1.a-b (claim 1, elements [1a]-[1b]).

- b. **[33b] “for a first data set having a plurality of first key-value pairs, wherein such first data set belongs to a first data group and the first key-value pairs have a first schema”**

216. For the reasons explained in Sections VIII.A.1.b (claim 1, element [1b]), VIII.A.1.f (claim 1, element [1f]), Pike (**Grounds 1 and 2**) and Pike in view of Chowdhuri (**Ground 2**) render this feature obvious. A POSITA would have understood that both Pike and Chowdhuri disclose a “data set” with key-value pairs “belong[ing] to a . . . data group.” Pike discloses input files, as discussed in Section VIII.A.1.b (claim 1, element [1b]). Chowdhuri discloses input tables, as discussed in Section VIII.A.1.f (claim 1, element [1f]). Pike also suggests and renders obvious input files of different schema, and Chowdhuri discloses this expressly. *See* Sections VIII.A.1.b (claim 1, element [1b]), VIII.A.1.f (claim 1, element [1f]). A POSITA would have combined Pike and Chowdhuri as explained in Sections VIII.A.1.a-b (claim 1, elements [1a]-[1b]).

- c. **[33c] “partitioning the first data set into a plurality of data partitions and”**

217. Pike and Pike-Chowdhuri disclose and render this feature obvious for the reasons explained in Section VIII.A.1.c (claim 1, element [1c]). It was also conventional. *See id.*

- d. **[33d] “providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable”**

218. Pike and Pike-Chowdhuri disclose and render this feature obvious for the reasons explained in Section VIII.A.1.d (claim 1, element [1d]). It was also conventional. *See id.*

- e. **[33e] “to independently output a plurality of lists of values for each of a set of keys found in such map function’s corresponding data partition to form a first intermediate data set having a first set of resulting key-value pairs;”**

219. Pike and Pike-Chowdhuri disclose and render this feature obvious for the reasons explained in Section VIII.A.1.e (claim 1, element [1e]). A POSITA would have understood that the “resulting key-value pairs” of mapping, for example, in Pike would include values taken from Pike’s input files or from Chowdhuri’s order table 308.

- f. **[33f] “for a second data set having a plurality of second key-value pairs, wherein such second data set belongs to a second data group and the second key-value pairs have a second schema,”**

220. Pike and Pike-Chowdhuri disclose and render this feature obvious for the reasons explained in Sections VIII.A.1.b (claim 1, element [1b], VIII.A.1.f (claim 1, element [1f]), VIII.A.18.b (claim 33, element [33b])).

g. [33g] “partitioning the first data set into a plurality of data partitions and”

221. This element recites partitioning the *first* data set into a plurality of data partitions. 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 and 2**) or splitting Chowdhuri’s customer table 308 (**Ground 2**) into data blocks renders this feature obvious, as explained in Sections VIII.A.18.c (claim 33, element [33c]), VIII.A.1.c (claim 1, element [1c]).

222. For example, it would have been obvious to split a portion of Pike’s input files or Chowdhuri’s customer table 308 into a first set of data blocks, to 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 308 to provide data blocks to a second set of mapping operators to form second intermediate data. 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.

223. 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 order table 309 into data blocks in the Pike-Chowdhuri system, renders this feature

obvious as explained in Sections VIII.A.18.c (claim 33, element [33c]), VIII.A.18.f (claim 33, element [33f]), VIII.A.1.c (claim 1, element [1c]).

- h. **[33h] “providing each data partition to a selected one of a plurality of mapping functions to form a second intermediate data set having a second set of resulting key-value pairs differing from the first set of resulting key-value pairs,”**

224. Pike and Pike-Chowdhuri render this feature obvious for the reasons explained in Sections VIII.A.1.d, VIII.A.1.e, VIII.A.1.g (claim 1, elements [1d], [1e], [1g]).

225. Forming “differing” sets of resulting key-value pairs is satisfied by mapping operators that each output, for example, a different *number* of key-value pairs. Finding a particular word once in one partition, for example, and two times in another partition, satisfies this limitation. 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,” mapping “every word w in every document in the collection, where the key/value pair is <w, 1>.” Pike, 9:56-10:15, 10:31-43. A POSITA would have understood that Pike’s mapping operators would output particular numbers of the same key for different data blocks, and that the number output from each map operator would differ from block to block.

226. Even if this element requires mapping using different keys, Pike (**Grounds 1 and 2**) discloses or renders this obvious by its mapping operators

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 different first and second key-value pairs, where the different words are the different keys. Pike's outputting a plurality of intermediate files 304 also teaches forming first and second intermediate data sets. *Id.*, 9:65-10:19, Fig. 3.

227. 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. 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. A POSITA would have combined Pike and Chowdhuri for the reasons explain in Sections VIII.A.1.b-h (claim 1, elements [1b]-[1h]), with predictable results.

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

228. Pike (**Grounds 1 and 2**) and Pike in view of Chowdhuri (**Ground 2**) render this feature obvious for the reasons explained in Sections VIII.A.18.a (claim 33, element [33a]), VIII.A.1.f (claim 1, element [1f]). Chowdhuri's order table and customer table are separate data groups with different key-value pairs and different schema. *See* Sections VIII.A.1.b (claim 1, element [1b]), VIII.A.1.f (claim 1, element [1f]).

- j. **[33j] “wherein the first and second schema and the first and second set of resulting key-value pairs have a key in common; and”**

229. Pike (**Grounds 1 and 2**) and Pike in view of Chowdhuri (**Ground 2**) render this feature obvious for the reasons explained in Section VIII.A.1.h (claim 1, element [1h]). For example, Pike’s mapping operators forming resulting key-value pairs based on a single word teaches this feature. *See* Section VIII.A.18.h (claim 33, element [33h]).

230. Chowdhuri alternatively discloses this by disclosing a customer and order table 308, 309 and its resulting intermediate data having a common key: “customer_id.” Chowdhuri ¶¶126-27, 131; *see* Sections VIII.A.1.f (claim 1, element [1f]), VIII.A.1.h (claim 1, element [1h]). Pike’s mapping operators in the Pike-Chowdhuri system would have mapped data blocks of the customer table 308 (or data blocks in both customer and order tables 308, 309), and the resulting key-value pairs would have a “customer_id” in common.

- k. **[33k] “reducing the first and the second intermediate data set together so as to form an output data set, wherein the reducing is accomplished by iterating on at least one key from the first and second intermediate data set, and”**

231. Pike (**Grounds 1 and 2**) discloses reducing intermediate data sets together to form output data for the reasons explained in Section VIII.A.1.i (claim 1, element [1i]). Pike and Chowdhuri (**Ground 2**) also disclose reducing by

iterating on a key from the intermediate data for the reasons explained in Section VIII.A.6 (claims 5 and 21). *See also* Pike, 11:6-38 (“retrieving intermediate data values” and sorting and merging the values “in accordance with the key . . . in the intermediate data”); Chowdhuri ¶¶129-35 (disclosing hash join, GroupBy, and sort iterators for reducing intermediate data). 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 VIII.A.18.h (claim 33, element [33h]).

I. [33I] “the output data set has a different schema than the first and second schema,”

232. Pike (**Grounds 1 and 2**) and Pike in view of Chowdhuri (**Ground 2**) render this feature obvious. For example, a POSITA would have understood that Pike’s word count in a collection of document (i.e., the output of its reducing) would have a different format, or “schema,” than the input documents. To the extent Pike does not expressly teach this feature, a POSITA would have found this obvious based on a POSITA’s knowledge of MapReduce, which describes counting words in documents that are webpages scraped from the Internet. EX-1009, 1-3. A POSITA would have recognized that a word count output from Pike’s reducing operation would have a different schema than a website.

233. The Pike-Chowdhuri system further renders this feature obvious by outputting a merged list of customers and orders based on a common state. *See*

Section VIII.A.18.h (claim 33, element [33h]). A POSITA would have understood that Chowdhuri's merged list of customers and orders based on state suggests an output with a different schema than the input customer and order tables, which would likely contain other fields than just customer names, orders, and states. This is consistent with the '610 patent, which describes merging a department and employee table into a combined department-employee table. '610 patent, 3:23-34, Figs. 4, 5.

m. [33m] "wherein the mapping and reducing operations are performed by a distributed system."

234. Pike and Pike-Chowdhuri disclose and render this feature obvious for the reasons explained in Section VIII.A.1.j (claim 1, element [1j]).

19. Claim 40

a. [40a] "A computer system including a plurality of computing devices, the computer system configured to process data of a data set, wherein the data set comprises a plurality of data groups, the computer system comprises at least one processor and memory that are operable to perform the following operations:"

235. Pike (**Grounds 1 and 2**) and Pike-Chowdhuri (**Ground 2**) a computer system with a plurality of computing devices with at least one processor and memory for the reasons explained in Sections VIII.A.1.a (claim 1, element [1a]), VIII.A.2.a (claim 17, element [17a]). Pike (**Grounds 1 and 2**) and Pike-Chowdhuri (**Ground 2**) disclose and render obvious processing data of a data set

that comprises a plurality of data groups for the reasons explained in Sections VIII.A.1.a (claim 1, element [1a]), VIII.A.1.b. (claim 1, element [1b]). A POSITA would have combined Pike and Chowdhuri as explained in Sections VIII.A.1.a-b (claim 1, elements [1a]-[1b]).

- b. **[40b] “for a first data set having a plurality of first key-value pairs, wherein such first data set belongs to a first data group and the first key-value pairs have a first schema,”**

236. Pike (**Grounds 1 and 2**) and Pike in view of Chowdhuri (**Ground 2**) disclose and render this feature obvious for the reasons explained in Section VIII.A.18.b (claim 33, element [33b]).

- c. **[40c] “partitioning the first data set into a plurality of data partitions and”**

237. Pike (**Grounds 1 and 2**) and Pike in view of Chowdhuri (**Ground 2**) disclose and render this feature obvious for the reasons explained in Section VIII.A.18.c (claim 33, element [33c]).

- d. **[40d] “providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable”**

238. Pike (**Grounds 1 and 2**) and Pike in view of Chowdhuri (**Ground 2**) disclose and render this feature obvious for the reasons explained in Section VIII.A.18.d (claim 33, element [33d]).

- e. **[40e] “to independently output a plurality of lists of values for each of a set of keys found in such map function’s corresponding data partition to form a first intermediate data set having a first set of resulting key value pairs;”**

239. Pike (**Grounds 1 and 2**) and Pike in view of Chowdhuri (**Ground 2**) disclose and render this feature obvious for the reasons explained in Section VIII.A.18.e (claim 33, element [33e]).

- f. **[40f] “for a second data set having a plurality of second key-value pairs, wherein such second data set belongs to a second data group and the second key-value pairs have a second schema,”**

240. Pike (**Grounds 1 and 2**) and Pike in view of Chowdhuri (**Ground 2**) disclose and render this feature obvious for the reasons explained in Section VIII.A.18.f (claim 33, element [33f]).

- g. **[40g] “partitioning the first data set into a plurality of data partitions and”**

241. Pike (**Grounds 1 and 2**) and Pike in view of Chowdhuri (**Ground 2**) disclose and render this feature obvious for the reasons explained in Section VIII.A.18.g (claim 33, element [33g]).

- h. **[40h] “providing each data partition to a selected one of a plurality of mapping functions to form a second intermediate data set having a second set of resulting key-value pairs differing from the first set of resulting key-value pairs,”**

242. Pike (**Grounds 1 and 2**) and Pike in view of Chowdhuri (**Ground 2**) disclose and render this feature obvious for the reasons explained in Section VIII.A.18.h (claim 33, element [33h]).

- i. **[40i] “wherein the first schema differs from the second schema; and”**

243. Pike (**Grounds 1 and 2**) and Pike in view of Chowdhuri (**Ground 2**) disclose and render this feature obvious for the reasons explained in Section VIII.A.18.i (claim 33, element [33i]).

- j. **[40j] “reducing the first and the second intermediate data set together so as to form an output data set, wherein the reducing is accomplished by iterating on at least one key from the first and second intermediate data set”**

244. Pike (**Grounds 1 and 2**) and Pike in view of Chowdhuri (**Ground 2**) disclose and render this feature obvious for the reasons explained in Section VIII.A.18.k (claim 33, element [33k]).

- k. **[40k] “and the output data set has a different schema than the first and second schema,”**

245. Pike (**Grounds 1 and 2**) and Pike in view of Chowdhuri (**Ground 2**) disclose and render this feature obvious for the reasons explained in Section VIII.A.18.l (claim 33, element [33l]).

1. **[40] “wherein the first and second schema and the first and second set of resulting key-value pairs have a key in common.”**

246. Pike (**Grounds 1 and 2**) and Pike in view of Chowdhuri (**Ground 2**) disclose and render this feature obvious for the reasons explained in Section VIII.A.18.m (claim 33, element [33m]).

20. **Claims 34 and 41 — reducing by “iterating on at least one key from each of the first and second intermediate data set so that the output data set is a merging of a portion of the first and second intermediate data set.”**

247. 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 and 2**) and Pike in view of Chowdhuri (**Ground 2**) render this feature obvious for the reasons explained in Sections VIII.A.18.j (claim 33, element [33j]), VIII.A.18.k (claim 33, element [33k]).

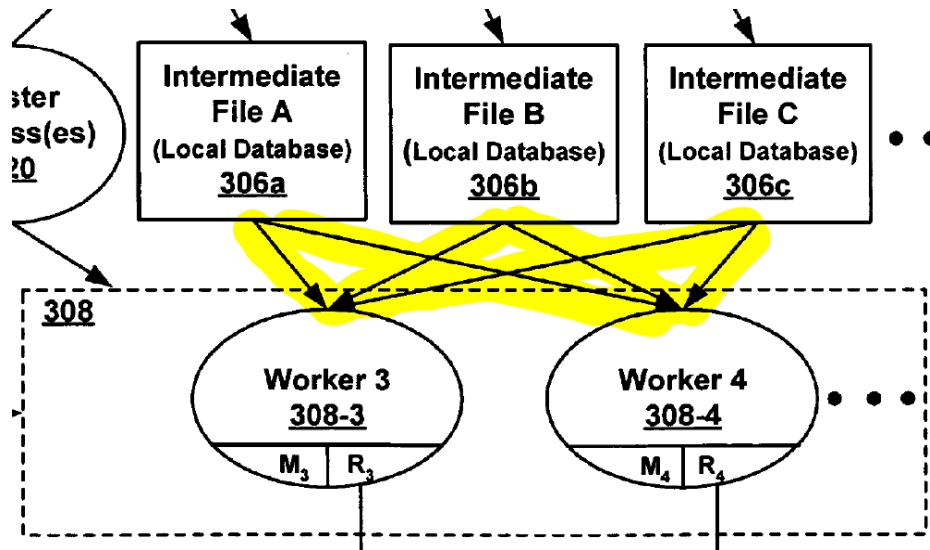
21. **Claims 35 and 42 — “wherein the reducing is accomplished by a plurality of reducers,” “partitioning the first or second intermediate data set into a plurality of partitions, at least some of the key-value pairs of the first or second intermediate data being provided to more than one of the partitions,” and “providing the partitioned intermediate data set of each partition to a separate one of the reducers.”**

248. Pike (**Grounds 1 and 2**), and Pike in view of Chowdhuri (**Ground 2**) disclose and render these features obvious for the reasons explained in claims 7 and 24. *See* Section VIII.A.8.

22. Claims 36 and 43 — “wherein all of the key-value pairs of the first and second intermediate data set are provided to all of the reducers.”

249. 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 of its key-value pairs are provided to its partitions, which in turn are each reduced by worker processes 308 in Pike and hashjoin iterators in Chowdhuri. *See* Sections VIII.A.7.b (claims 6 and 23, element [b]), VIII.A.9 (claims 8 and 25).

250. 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 for the reasons explained in Section VIII.A.9 (claims 8 and 25). Indeed, Pike discloses that data from each of its intermediate data files 306 is provided to all of its worker processes 308, as reflected below in Figure 3. Pike, 11:6-55, Fig. 3. Pike also suggests that its worker processes 308 read more than one intermediate data file, and even all intermediate files: “*each* of the worker processes 308 reads a corresponding one of the intermediate files 306 *produced by all . . .* of the worker processes 304.” *Id.* (emphases added); *see also id.* (“Each worker process 308 sorts the intermediate data values *in the subset of the intermediate files read* by that worker process.”) (emphasis added).



Id., Fig. 3 (annotated).

251. A POSITA would have understood from this that Pike at least suggests each key-value pair being provided to all reducers. Pike depicts the intermediate data from each of its intermediate data files 306 being distributed to each worker process 308 and describes each worker process reading data from “all” worker processes 304. This renders obvious providing each key-value pair in the input data to all reducers. 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. Indeed, Pike suggests this by disclosing its reduce tasks may include statistical operators that provide statistics about the key-value pairs in the input data sets. Pike, 11:38-55. A POSITA would have understood that the most accurate statistics account for all available data in the data sets, including all of the key-value pairs.

- 23. Claims 37 and 44 — “wherein at least some of the reducers include a sort, group-by-key and combine task,” and “generating and providing metadata for at least some of the mapping, partitioning, combining, grouping and sorting.”**

252. Pike (**Grounds 1, 2, and 3**) and Chowdhuri (**Grounds 2 and 3**) teach and render this feature obvious for the reasons explained in Section VIII.A.10 (claims 9 and 26).

- 24. Claims 38 and 45 — “wherein the reducing includes processing the metadata.”**

253. Pike (**Grounds 1, 2, and 3**) and Chowdhuri (**Grounds 2 and 3**) teach and render this feature obvious for the reasons explained in Section VIII.A.11 (claims 10 and 27).

- 25. Claims 39 and 46 — “wherein iterating includes providing the associated metadata to the processing of the reducing step.”**

254. Pike (**Grounds 1, 2, and 3**) and Chowdhuri (**Grounds 2 and 3**) teach and render this feature obvious for the reasons explained in Section VIII.A.12 (claims 11 and 28).

IX. CONCLUSION

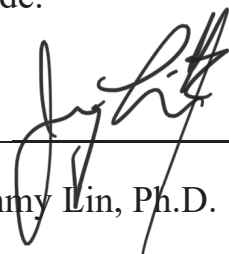
255. For the reasons set forth above, it is my opinion that claims 1-46 of the '610 patent are unpatentable in view of the prior art.

256. In signing this declaration, I understand that the declaration will be filed as evidence in a contested case before the Patent Trial and Appeal Board of the United States Patent and Trademark Office. I acknowledge that I may be subject to cross-examination in this case and that cross-examination will take place within the United States. If cross-examination is required of me, I will appear for cross-examination within the United States during the time allotted for cross-examination.

257. I declare that all statements made herein of my knowledge are true, that all statements made on information and belief are believed to be true, and that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code.

Dated: December 17, 2023

By: _____


Jimmy Lin, Ph.D.