(51) International Patent Classification:
G06F 17/30 (2006.01) G06F 19/00 (201.01)

(21) International Application Number:
PCT/US2016/014794

(22) International Filing Date:
26 January 2016 (26.01.2016)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant: HEWLETT PACKARD ENTERPRISE DE¬
VELOPMENT LP [US/US]; 11445 Compaq Center Drive
West, Houston, Texas 77070 (US).

(72) Inventors: KIMURA, Hideaki; 1501 Page Mill Rd., Palo
Alto, California 94304-100 (US). SIMITSIS, Alkis; 1501
WILKINSON, William K.; 1501 Page Mill Rd., Palo
Alto, California 94304-1100 (US).

(74) Agents: CREERON, Kerry T et al; Hewlett Packard En-
terprise, Intellectual Property Administration, 3404 E. Har-
mony Road, Mail Stop 79, Fort Collins, Colorado 80528
(US).

(81) Designated States (unless otherwise indicated, for every
kind of national protection available): AE, AG, AL, AM,
AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY,
BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM,
DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT,
HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR,
KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG,
MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM,
PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC,
SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN,
TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States (unless otherwise indicated, for every
kind of regional protection available): ARIPO (BF, BJ, CF,
CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN,
TD, TG).

Declarations under Rule 4.17:
— as to the identity of the inventor (Rule 4.1.7(a))
— as to applicant’s entitlement to apply for and be granted a
patent (Rule 4.1.7(b))

Published:
— with international search report (Art. 21(3))

(53) Title: PERFORMING OPERATIONS ON A GRAPH

FIG. 1

(57) Abstract: A computing system includes a graph engine comprising an embedded database. The graph engine may: store ver-
tices of a graph in the embedded database as key-value pairs associated with each of the vertices. Each of the key-value pairs may
contain information associated the vertices. The graph engine may further: receive a request indicating operations to perform on a
plurality of the vertices, partition the graph into partitions comprising a subset of the vertices of the graph, partition the vertices
into blocks comprising a subset of the vertices of the partitions, and perform the indicated operations on the plurality of the vertices
of the request.
Performing Operations on a Graph

BACKGROUND

[0001] A graph is a data structure comprising a set of vertices. The vertices may be connected to each other with edges.

BRIEF DESCRIPTION OF THE DRAWINGS

[0002] Certain examples are described in the following detailed description and in reference to the drawings, in which:

[0003] FIG. 1 is a conceptual diagram of an example computing system that may perform operations on a graph;

[0004] FIG. 2 is another conceptual diagram of an example computing system that may perform operations on a graph;

[0005] FIG. 3 is another conceptual diagram of an example computing system that may perform operations on a graph;

[0006] FIG. 4 is a flowchart of an example method for performing operations on a graph;

[0007] FIG. 5 is a flowchart of an example method performing operations on a graph; and

[0008] FIG. 6 is a block diagram of an example for performing operations on a graph.

DETAILED DESCRIPTION

[0009] A graph is a data structure comprising vertices. The vertices may be connected to each other with edges. The edges may be bi-directional or unidirectional. Properties may be associated with a vertex or an edge. A computing device may perform operations, e.g. algorithms, on the graph to determine information about the structure of the graph.

[0010] As an example, a graph may be used to represent flight paths of an airline. The vertices of the graph may represent cities. The cities are connected to each other via flight paths, which may be represented using edges that connect the vertices. Each of the vertices may have an associated cost.
The associated costs may represent a cost, such as distances associated with the flight paths.

[0011] A computing device may be configured to determine properties about the graph, such as the shortest path, or lowest cost that may be obtained to travel between two cities, based on the available flight paths. Various algorithms, such as Dijkstra's algorithm, the Bellman-Ford algorithm, or the like, may be used to determine lowest cost or shortest path by analyzing and/or navigating the vertices of the graph.

[0012] Graphs may comprise millions or billions of vertices in some cases. Being able to efficiently perform graph workloads in a scalable matter is useful as the number of vertices scales upward. This disclosure is related to a graph engine capable of performing operations on graphs with such large numbers of vertices.

[0013] More particularly, the techniques of this disclosure are related to a graph engine. A graph engine is software specifically designed to perform operations on a graph. The graph engine of this disclosure may scale efficiently to hundreds or thousands of cores, and terabytes of memory. The graph engine of this disclosure may also perform well on non-uniform memory architecture (NUMA) systems, which have non-uniform for transferring data between cores of the computing system.

[0014] The graph engine as described herein comprises, or is coupled with an embedded database. The embedded database is so-called because the graph engine has knowledge about the underlying structure and performance of the database. The graph engine uses the embedded database to store the vertices of the graph in a particular fashion, and to perform operations on the stored vertices in order to maximize performance. As an example, the graph engine may control partitioning of the database among various NUMA cores, and may control threading of the embedded database.

[0015] The embedded database may store each vertex as a row in the embedded database. Each row may further comprise a set of key-value pairs. Additionally, the embedded database may store additional information associated with each vertex, such as information that indicates the edges that
connect with each vertex along with each key-value pair, cost associated with each vertex, and/or metadata.

The graph engine also uses the embedded database to enforce transactional consistency (e.g., atomicity, consistency, isolation and durability, "ACID") when accessing a key-value pair associated with a vertex or metadata. When performing an operation on a key-value pair and associated information, the embedded database also performs "lightweight" transactions. These lightweight transactions do not lock significant portions of the database, but rather lock the portion of the database associated with a particular key-value pair. These database transactions also provide high transactional throughput due to the embedded database not locking larger objects, such as partitions of the database or entire tables of a database.

FIG. 1 is a conceptual diagram of an example computing system that may perform operations on a graph. Computing system 100 is illustrated in FIG. 1. Computing system 100 comprises a graph engine 116, embedded database 112, and graph 102. Graph engine 116 may receive requests, e.g. from a client device. Based on the request, graph engine 116 may generate one or more operations 114 to perform on graph 102.

Embedded database 112 is coupled with graph engine 116, and comprises key value pairs 118. Key-value pairs 118 are stored in blocks 106A-106N, and 108A-108N, although illustrated otherwise for the purpose of simplicity. Embedded database 112 may comprise a NoSQL database, in some examples. Graph engine 116 may control the structure of key-value pairs 118, as well as the execution of embedded database 112.

Graph 102 may comprise a set of vertices that are stored as key-value pairs 118 within embedded database 112. Graph engine 116 may cause embedded database 112 to store the vertices and edges of graph 102 as entries in a table, denoted as \( V \), also referred to as "VertexIndexTable." \( V \) is illustrated as table 210 (described below with respect to FIG. 2). The information in the table comprises rows each of which is associated with a vertex. Each row comprises a vertex and information about outgoing edges associated with that vertex.
Each row of $V_G$ may comprise a schema corresponding to: [vid, <meta>, edg_cnt, <eidi, metai>, ... , <eidN, metaN>], where "vid" denotes a vertex id that is associated with a particular vertex, "<meta>" denotes metadata associated with the vertex, "edg_cnt" is the out-degree (number of edges that exit) of the vertex, and <eidi, metai>... <eidN, metaN> are tuples corresponding to identifiers of each of the outgoing edges of the vertex and any metadata associated with the edges. Depending on the size and complexity of graph 102, graph engine 116 and embedded database 112 may store the metadata in a table row of $V_G$, or the table row may comprise a reference to the metadata, e.g. a pointer.

Graph engine 116 may divide the table comprising the set of vertices of graph 102 into partitions, such as partitions 104A through 104N, where "N" is any number of partitions. Graph engine 116 may further divide the vertices of each partition into blocks comprising a subset of the vertices of the partition. For example, partition 104A is divided into blocks 106A-106N, and block 104N is divided into blocks 108A-108N, where "N" may be number of blocks. In various examples, a partition may comprise 256x256 blocks, and each block may comprise a grid having 32 vertices. Graph engine 116 may partition the vertices of graph 102 such that each block is assigned to a different core of a NUMA computing device in some examples.

Graph engine 116 may access a row having a particular vertex based on the vertex id, which is a key value that maps to the row of $V_G$ having the particular vertex id. Thus, the keys of key-value pairs 118 may comprise vertex identifiers and the pair values may comprise the corresponding row of the table, $V_G$. When graph engine 116 performs an operation, such as one of operations 114 on a vertex, embedded database 112 may perform a transaction on a row associated with a vertex id involved in the operation, and may exclusively lock the row associated with that vertex id during the transaction.

When performing an operation, graph engine 116 may cause embedded database 112 to execute a single transaction or multiple transactions depending on whether graph engine 116 is performing a navigational operation or an analytical operation on graph 102. A navigational operation may comprise
a navigational query of graph 102 that touches or involves a relatively small number of vertices. An example navigational operation may comprise a shortest-path algorithm, which may for example be performed using Dijkstra's algorithm. The navigational query may attempt to find a shortest-path between vertices located within tens of hops of each other. Graph engine 116 may cause embedded database 112 to perform the navigational operation using a single transaction that executes on a single thread, and/or on a single core of a computing device, as is described in greater detail elsewhere.

[0024] Although described as performing a single transaction, a single transaction may involve or interact with multiple vertices. For example, a navigational operation may interact with vertices 108A-108N. Corresponding to the interaction with vertices 108A-108N, embedded database 112 may lock corresponding rows of embedded database 112 corresponding to multiple involved vertices in some examples.

[0025] Operations 114 may also comprise an analytics operation in various examples. Graph engine 116 may also execute the analytics operation using vertices stored in embedded database 112. An analytics operation is a query that may access a large fraction of the vertices of graph 102. In some examples, the number of vertices involved may comprise hundreds of millions vertices. As an example referring to FIG. 1, an analytics operation of operation 114 may access most or all of the vertices of each of partitions 104A-104N.

[0026] Graph engine 116 may cause embedded database 112 to execute multiple transactions when performing an analytics operation. Using multiple transactions may improve analytical operation execution performance. Each transaction may correspond to accessing a particular vertex or to updating a shared table that indicates the progress of the analytics operation. Graph engine 116 may use an algorithm, such as a Bellman-Ford algorithm for analytics workloads in some examples. Additional details regarding performing an analytics operations are illustrated in greater detail below.

[0027] FIG. 2 is another conceptual diagram of an example computing device that may perform operations on a graph. FIG. 2 illustrates a computing
system 200. Graph engine 116 and embedded database 112 perform navigational operation 202 in the example of FIG. 2.

[0028] As described above, a navigational operation may involve a smaller number of vertices relative to an analytical operation. In the example of FIG. 2, graph engine 116 may receive a request 206, which may comprise a single operation or multiple operations that a client is requesting that graph engine 116 perform. Based on request 206, graph engine 116 determines that graph engine 116 is to perform a navigational operation, and generates navigational operation 202.

[0029] Graph engine 116 determines that embedded database 112 should perform a single transaction 204 as part of executing navigational operation 202. Transaction 204 may access vertices 108A and 108N in this example. Embedded database 112 may access vertices 108A and 108N as part of executing transaction 204. Each navigational transaction, e.g., transaction 204 may execute as a single thread. Additionally, embedded database 112 locks rows 208A and 208N of table 210 because rows 208A and 208N are associated with vertices 108A and 108N as part of performing transaction 204. Upon completing transaction 204, embedded database 112 unlocks rows 208A and 208N. As described herein, a lock may comprise a conceptual lock. A conceptual lock may comprise any transaction serialization mechanism that embedded database 112 may use to supports high concurrency.

[0030] FIG. 3 is another conceptual diagram of an example computing device that may perform operations on a graph. FIG. 3 illustrates a computing system 300. Graph engine 116 and embedded database 112 perform an analytics operation 302 in the example of FIG. 3.

[0031] As described above, an analytical operation, such as analytical operation 302, may involve a much greater number of vertices relative to a navigational operation. In the example of FIG. 3, graph engine 116 may receive a request 206, which may comprise a single operation or multiple operations that a client is requesting that graph engine 116 perform. Based on request
206, graph engine 116 determines that graph engine 116 is to perform an analytical operation, and generates analytical operation 302.

[0032] Graph engine 116 determines that embedded database 112 should perform multiple transactions as part of executing navigational operation 202. More particularly, graph engine 116 may determine that analytical operation 302 accesses vertices 108A, 108B, and 108N. Based on the determination that vertices 108A, 108B, and 108N are accessed, embedded database 112 generates corresponding transactions, i.e. transactions 304A, 304B, and 304N.

[0033] In some examples, for each access of a vertex, embedded database 112 may perform a separate transaction. For example, embedded database 112 performs transaction 304A when accessing vertex 108A, and locks corresponding row 208A of table 208A. Similarly, embedded database 112 performs transaction 304B when accessing vertex 108B, and locks row 208B, and a similar process when accessing vertex 108N. The transaction may then read, update, or add data to the locked row. Embedded database 112 unlocks a row when the corresponding transaction completes. In various examples, transactions 304 may depend upon each other and may be performed in a sequence based on their dependencies. For example, the execution of transaction 304B may depend upon the completion of transaction 304A.

[0034] As an example, analytical operation 302 may comprise a Bellman-Ford algorithm. A Bellman-Ford algorithm determines a single shortest path (SSSP) starting from a given source vertex, and finds the shortest path to every other vertex in the graph that is connected with the source. The Bellman-Ford algorithm may be distributed across partitions and blocks to compute the shortest path from the source vertex in parallel using multiple cores of a computing device.

[0035] To execute the Bellman-ford algorithm or another analytical operation, threads may calculate and propagate so-called "relax" distances from distances form the source vertex. Upon propagating an updated distance value, the completing thread communicates with another thread to request further
calculation and propagation. The request for further calculation is referred to as an "activate" operation.

[0036] When performing an analytical operation 302, such as a Bellman-Ford algorithm, graph engine 116 may cause embedded database 112 to generate and execute a transaction for both distance calculation, and for distance propagation. In the example of FIG. 3, transactions 304A-304N may each calculate a distance value. After a distance value has been calculated, embedded database 112 executes a separate transaction to update the distance value in shared table 308. Table 308 may comprise rows, and each row may comprise a vertex identifier, a distance value, and a parent vertex identifier.

[0037] As a more particular example, if vertex 108A is a vertex, graph engine 116 uses row table 202 to determine all neighboring vertices and their associated distances. When graph engine determines the distance between vertex 108A and a neighboring vertex, such as vertex 108B, embedded database 112 executes a single transaction 104A. Additionally, upon determining the distance between vertex 108A and 108B, embedded database 112 executes a separate transaction 306 to update distance in shared table 308.

[0038] To generalize, graph engine 116 may partition operations to be performed on graph 102 into multiple computations. Graph engine 116 may spawn one or more threads. Each of the threads may be associated with, and may execute one or more of the computations, and each of the threads may attach to a particular core. Examples of such computations may comprise activation and relaxation operations, e.g. performed as part of executing a Bellman-Ford algorithm.

[0039] By using fine-grained locking of rows associated with vertices, as well as shared table 308 to store state for various computations, e.g. for activation and relaxation operations, respectively, the graph engine techniques of this disclosure achieve speed-ups in throughput for analytical workloads. Additionally, the techniques of this disclosure may assign each block, e.g. block 106A, to a different core of a computing device. In NUMA systems, threads may
attach to a particular core to be NUMA-aware. That is, graph engine 116 may assign cores, e.g. a group of cores comprising a NUMA node, to work on blocks that are near to each other. For a navigational workload, graph engine 116 may attach the navigational workload to a particular core.

[0040] The techniques of this disclosure also provide the ability to run navigational and analytical operations (referred to as "mixed workloads") without compromising performance, to execute mixed workloads the same storage engine using transactions, and to execute mixed workloads without having to segregate navigational vs. analytical operations. Experimental observations indicate multiple orders of magnitude improvement when performing analytics workloads on large datasets as compared to other approaches for performing analytical graph workloads in some cases.

[0041] FIG. 4 is a flowchart of an example method for performing operations on a graph. Method 400 may be described below as being executed or performed by a system, for example, computing system 100 (FIG. 1), computing system 200 (FIG. 2), or computing system 300 (FIG. 3). In various examples, method 400 may be performed by hardware, software, firmware, or any combination thereof. Other suitable systems and/or computing devices may be used as well. Method 400 may be implemented in the form of executable instructions stored on at least one machine-readable storage medium of the system and executed by at least one processor of the system. Alternatively or in addition, method 400 may be implemented in the form of electronic circuitry (e.g., hardware). In alternate examples of the present disclosure, one or more blocks of method 400 may be executed substantially concurrently or in a different order than shown in FIG. 4. In alternate examples of the present disclosure, method 400 may include more or fewer blocks than are shown in FIG. 4. In some examples, one or more of the blocks of method 400 may, at certain times, be ongoing and/or may repeat.

[0042] Method 400 may start at block 402 at which point the computing system, e.g. computing system 100, e.g. graph engine 116 may store vertices of graph 102 in an embedded database 112 comprising key-value pairs 118
associated with each of the vertices, e.g. vertices 108A-108N. Each of the key-value pairs may comprises information associated with vertices 108.

[0043] Method 400 may proceed to block 402 at which point graph engine 116 may cause embedded database 112 to partition the graph into partitions comprised of a subset of the vertices, and to block 402 where graph engine 116 and embedded database 112 may partition the vertices into blocks, e.g. blocks 106 comprising a subset of the vertices, e.g. vertices 108A of the partition, e.g. partition 104A. Method 400 may proceed to block 408 at which point graph engine 116 and/or embedded database 112 may perform operations on a plurality of the vertices of embedded database 112, wherein performing the operations comprises performing transactions using the embedded database 112.

[0044] FIG. 5 is a flowchart of an example method for performing operations on a graph. FIG. 5 illustrates method 500. Method 500 may be described below as being executed or performed by a system, for example, computing system 100 (FIG. 1) or computing system 200 (FIG. 2). Other suitable systems and/or computing devices may be used as well. Method 500 may be implemented in the form of executable instructions stored on at least one machine-readable storage medium of the system and executed by at least one processor of the system. Method 500 may be performed by hardware, software, firmware, or any combination thereof.

[0045] Alternatively or in addition, method 500 may be implemented in the form of electronic circuitry (e.g., hardware). In alternate examples of the present disclosure, one or more blocks of method 500 may be executed substantially concurrently or in a different order than shown in FIG. 5. In alternate examples of the present disclosure, method 500 may include more or fewer blocks than are shown in FIG. 5. In some examples, one or more of the blocks of method 500 may, at certain times, be ongoing and/or may repeat.

[0046] In various examples, method 500 may start at block 502, at which point graph engine 116 may store vertices of graph 102 in an embedded database 112 comprising key-value pairs 118 associated with each of the vertices, wherein each of the key-value pairs 118 comprises information
associated with a vertex of the vertices. In various examples, the associated information may at least one of: an identifier of the vertex, a distance to another vertex of the vertices, or edge information of the vertex.

[0047] Method 500 may proceed to block 504 at which point graph engine 116 may cause embedded database 112 to partition the graph into partitions comprised of a subset of the vertices, and to block 506 where graph engine 116 and embedded database 112 may partition the vertices into blocks, e.g. blocks 106 comprising a subset of the vertices, e.g. vertices 108A of the partition, e.g. partition 104A.

[0048] In various examples, method 500 may proceed to block 508, at which point graph engine 116 may receive a request, such as request 206, that indicates operations to perform. At block 510, database engine 116 and embedded database 112 may perform at least one of the operations. The operations may comprise at least one of a navigation operation, e.g. navigational operation 202, or an analytical operation, e.g. analytical operation 302.

[0049] In various examples, to perform the operations, graph engine 116 and embedded database 112 may partition the operations into multiple computations. Each of the computations may be handled by a thread associated with one of the operations. The threads may execute each of the operations associated with the threads. The threads may also lock rows of embedded database 112, e.g. using conceptual locking. The rows of embedded database 112 comprise key-value pairs 118, which are associated with each of the operations, e.g. one or more analytical operations 302.

[0050] Graph engine 116 and embedded database 112 may perform a single transaction for each of the operations comprising navigational operations. Database engine 116 and embedded database 112 may perform a transaction for each relaxation operation of a vertex of a subset of vertices of a block, and a transaction for each activation operation associated with the vertex.

[0051] In various examples, method 500 may proceed to block 512. To perform block 512, e.g. to perform the operations on a vertex of the subset of vertices of the block, embedded database 112 may perform a transaction on the
vertex. To perform the transaction, embedded database 112 may lock a row of the embedded database corresponding to a vertex of the vertices on which the operation is being performed.

[0052] In some examples, graph 102 may be partitioned into a first block and a second block. In this example, graph engine 116 may perform block 514, and may process the first block, e.g. block 106A, on a thread executing on a first core of a computing device and process the second block, e.g. block 106N, on a thread executing on a second core of the computing device.

[0053] FIG. 6 is a block diagram of an example for performing operations on a graph. In the example of FIG. 6, system 600 includes a processor 610 and a machine-readable storage medium 620. Although the following descriptions refer to a single processor and a single machine-readable storage medium, the descriptions may also apply to a system with multiple processors and multiple machine-readable storage mediums. In such examples, the instructions may be distributed (e.g., stored) across multiple machine-readable storage mediums and the instructions may be distributed (e.g., executed by) across multiple processors.

[0054] Processor 610 may be one or more central processing units (CPUs), microprocessors, and/or other hardware devices suitable for retrieval and execution of instructions stored in machine-readable storage medium 620. In the particular example shown in FIG. 6, processor 610 may fetch, decode, and execute instructions 622, 624, 626, 628 to perform operations on a graph.

[0055] As an alternative or in addition to retrieving and executing instructions, processor 610 may include one or more electronic circuits comprising a number of electronic components for performing the functionality of one or more of the instructions in machine-readable storage medium 620. With respect to the executable instruction representations (e.g., boxes) described and shown herein, it should be understood that part or all of the executable instructions and/or electronic circuits included within one box may, in alternate examples, be included in a different box shown in the figures or in a different box not shown.
Machine-readable storage medium 620 may be any electronic, magnetic, optical, or other physical storage device that stores executable instructions. Thus, machine-readable storage medium 620 may be, for example, Random Access Memory (RAM), an Electrically-Erasable Programmable Read-Only Memory (EEPROM), a storage drive, an optical disc, and the like. Machine-readable storage medium 620 may be disposed within system 600, as shown in FIG. 6. In this situation, the executable instructions may be "installed" on the system 600. Alternatively, machine-readable storage medium 620 may be a portable, external or remote storage medium, for example, that allows system 600 to download the instructions from the portable/external/remote storage medium.

Referring to FIG. 6, vertex storage instructions 622, when executed by a processor (e.g., 610), may cause processor 610 store vertices, e.g. vertices 108 of a graph 102 in an embedded database 112 comprising key-value pairs 118 associated with each of vertices 108. Each of the key-value pairs 118 comprises information associated with a vertex of the vertices. In various examples, the information associated with the one of the vertices may comprise at least one of: an identifier of the vertex, a distance to another vertex of the vertices, or edge information of the vertex.

Graph partitioning instructions 624, if executed, may cause processor 610 to determine partition the graph into partitions, e.g. partitions 104, comprising a subset of the vertices.

Processor 610 may execute block partitioning instructions 626 in various examples. Block partitioning instructions 626 may cause processor 610 partition the vertices into blocks comprising a subset of the vertices of the partitions.

In some examples, processor 610 may execute operation performance instructions 628. Operation performance instructions 628, if executed, may cause processor 610 to perform operations, e.g. operations 114, comprising transactions on a plurality of the vertices of embedded database 112.
CLAIMS

1. A method for performing operations on a graph, the method comprising:
   storing vertices of the graph in an embedded database comprising key-value
   pairs associated with each of the vertices, wherein each of the key-value
   pairs comprises information associated with a vertex of the vertices;
   partitioning the graph into partitions comprised of a subset of the vertices;
   partitioning the vertices into blocks comprising a subset of the vertices of
   the partitions; and
   performing operations on a plurality of the vertices of the embedded
   database, wherein performing the operations comprises performing transactions
   using the embedded database.

2. The method of claim 1,
   the method further comprising:
   receiving a request that indicates the operations to perform; and
   performing the operations on the vertices, wherein the operations may
   comprise at least one of a navigational operation or an analytical operation.

3. The method of claim 1, wherein the information associated with the one
   of the vertices comprises at least one of: an identifier of the vertex, a distance to
   another vertex of the vertices, or edge information of the vertex.

4. The method of claim 1, wherein the blocks comprise a first block and a
   second block, the method further comprising:
   processing the first block on a thread executing on a first core of a
   computing device and processing the second block on a thread executing on a
   second core of the computing device.
5. The method of claim 1, further comprising:
   performing an operation on a vertex of the subset of vertices of the block;
   and
   performing a transaction on the vertex using the embedded database,
   wherein performing the transaction comprises locking a row of the embedded
   database corresponding to a vertex of the vertices on which the operation is
   being performed.

6. The method of claim 1, wherein performing the operations comprises
   performing a single transaction for each of the operations comprising
   navigational operations.

7. The method of claim 1, wherein performing the operations comprises
   partitioning the operations into multiple computations;
   spawning threads associated with the computations;
   executing, by the associated threads, the computations associated with
   the threads; and
   locking, by each of the threads, rows of the embedded database
   comprising key-value pairs associated with executing the computations.
8. A computing system comprising:
   a graph engine comprising; and
   an embedded database;
   the graph engine to:
   store vertices of a graph in the embedded database as key-value pairs
   associated with each of the vertices, wherein each of the key-value pairs
   comprises information associated the vertices,
   the graph engine further to:
   receive a request indicating operations to perform on a plurality of the
   vertices;
   partition the graph into partitions comprising of a subset of the vertices of
   the graph;
   partition the vertices into blocks comprising a subset of the vertices of the
   partitions; and
   perform the indicated operations on the plurality of the vertices of the
   request.

9. The computing system of claim 8, further comprising:
   a first core to:
   perform a first operation of the operations on a first one of the
   blocks; and
   a second core to:
   perform a second, different operation of the operations a second
   one of the blocks, wherein the second block is different than the first
   block.

10. The computing system of claim 8, wherein the key-value pairs further
     comprise information associated with the vertices, the information comprising:
     at least one of: an identifier of the vertex, a distance to another vertex of
     the vertices, or edge information of the vertex.
11. The computing system of claim 8, wherein the request comprises a plurality of requests, wherein each of the requests comprise a plurality of operations on a set of the vertices of the graph, the graph engine further to:
   perform the operations of the requests on the set of vertices concurrently.

12. The computing system of claim 8, wherein to perform the operations, the graph engine is further to:
   perform a single transaction using the embedded database for each of the operations comprising navigational operations.

13. The computing system of claim 8, wherein to perform the operations, the graph engine further to:
   partition the operations into multiple computations;
   spawn threads associated with the computations;
   execute, by the associated threads, each of the computations associated with the threads; and
   lock, by each of the threads, rows of the embedded database comprising key-value pairs associated with executing the computations.

14. A non-transitory machine-readable storage medium encoded with instructions, the instructions that, when executed, cause a processor to:
   store vertices of a graph in an embedded database comprising key-value pairs associated with each of the vertices, wherein each of the key-value pairs comprises information associated with a vertex of the vertices;
   partition the graph into partitions comprised of a subset of the vertices;
   partition the vertices of the partitions into blocks comprising a subset of the vertices of the partitions; and
   perform operations comprising transactions on a plurality of the vertices of the embedded database.

15. The non-transitory computer-readable storage medium of claim 14, wherein the information associated with the one of the vertices comprises
at least one of: an identifier of the vertex, a distance to another vertex of the vertices, or edge information of the vertex.
400

STORE VERTICES OF THE GRAPH IN AN EMBEDDED DATABASE COMPRISING KEY-VALUE PAIRS ASSOCIATED WITH EACH OF THE VERTICES

402

PARTITION THE GRAPH INTO PARTITIONS COMPRISED OF A SUBSET OF THE VERTICES

404

PARTITION THE VERTICES INTO BLOCKS COMPRISING A SUBSET OF THE VERTICES

406

PERFORM OPERATIONS ON A PLURALITY OF THE VERTICES

408

FIG. 4
STORE VERTICES OF THE GRAPH IN AN EMBEDDED DATABASE COMPRISING KEY-VALUE PAIRS ASSOCIATED WITH EACH OF THE VERTICES

PARTITION THE GRAPH INTO PARTITIONS COMPRISED OF A SUBSET OF THE VERTICES

PARTITION THE VERTICES INTO BLOCKS COMPRISING A SUBSET OF THE VERTICES

RECEIVE A REQUEST THAT INDICATES OPERATIONS TO PERFORM

PERFORM AT LEAST ONE OF A NAVIGATIONAL OR ANALYTICAL OPERATION ON A PLURALITY OF THE VERTICES

LOCK ROWS OF THE EMBEDDED DATABASE ASSOCIATED WITH THE VERTICES

PROCESS A FIRST BLOCK OF THE BLOCKS ON A FIRST CORE AND A SECOND BLOCK ON A SECOND CORE
FIG. 6
A. CLASSIFICATION OF SUBJECT MATTER
G06F 17/30(2006.01)i, G06F 19/00(2011.01)i
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED
Minimum documentation searched (classification system followed by classification symbols)
G06F 17/30; G06Q 40/00; G06Q 10/00; G06F 19/00

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
Korean utility models and applications for utility models
Japanese utility models and applications for utility models

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
eKOMPASS(KIPO internal) & Keywords: graph, vertices, embedded database, key-value pairs, partitioning, subset of vertices, blocks, operations, transactions

C. DOCUMENTS CONSIDERED TO BE RELEVANT

<table>
<thead>
<tr>
<th>Category</th>
<th>Citation of document, with indication, where appropriate, of the relevant passages</th>
<th>Relevant to claim No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>US 2012-0310916 AI (DANIEL ABADI et al.) 06 December 2012 See paragraphs [0011], [0022] - [0023], [0028], [0063H0064], [0097], [0104], [0112H0115], and [0124]; claim 15; and figures 1, 13, and 16.</td>
<td>1-15</td>
</tr>
<tr>
<td>A</td>
<td>US 2010-0169137 AI (GRAHAME ANDREW IAASREBSKI et al.) 01 July 2010 See paragraphs [0036]-[0037], [0046], [0084], and [0111]; and figures 1-2.</td>
<td>1-15</td>
</tr>
<tr>
<td>A</td>
<td>US 2014-0280360 AI (JAMES WEBBER et al.) 18 September 2014 See paragraphs [0005], [0018], [0039], [0044], and [0046]; and figure 3.</td>
<td>1-15</td>
</tr>
<tr>
<td>A</td>
<td>US 2013-0097133 AI (EBAY INC.) 18 April 2013 See paragraphs [0030]-[0031], [0050], and [0083]-[0084]; and figures 8 and 19.</td>
<td>1-15</td>
</tr>
<tr>
<td>A</td>
<td>US 2015-0169687 AI (AB INITIO TECHNOLOGY LLC) 18 June 2015 See paragraphs [0023]-[0024], [0034], [0046], [0048], and [0052]; and figures 6-7.</td>
<td>1-15</td>
</tr>
</tbody>
</table>

Further documents are listed in the continuation of Box C.

* Special categories of cited documents:
  "A" document defining the general state of the art which is not considered to be of particular relevance
  "E" earlier application or patent but published on or after the international filing date
  "L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
  "O" document referring to an oral disclosure, use, exhibition or other means
  "P" document published prior to the international filing date but later than the priority date claimed

See patent family annex.

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search 24 October 2016 (24.10.2016)

Date of mailing of the international search report 24 October 2016 (24.10.2016)

Name and mailing address of the ISA/KR International Application Division Korean Intellectual Property Office 189 Cheongsa-ro, Seo-gu, Daejeon, 35208, Republic of Korea Facsimile No. +82-42-481-8578

Authorized officer NHO, Ji Myong

Telephone No. +82-42-481-8528

Form PCT/ISA/210 (second sheet) (January 2015)
<table>
<thead>
<tr>
<th>Patent document cited in search report</th>
<th>Publication date</th>
<th>Patent family member(s)</th>
<th>Publication date</th>
</tr>
</thead>
<tbody>
<tr>
<td>US 2012-0310916 Al</td>
<td>06/12/2012</td>
<td>EP 2729883 A2</td>
<td>14/05/2014</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WO 2013-009503 A2</td>
<td>17/01/2013</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WO 2013-009503 A3</td>
<td>30/05/2014</td>
</tr>
<tr>
<td>US 2010-0169137 Al</td>
<td>01/07/2010</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>WO 2014-170762 A2</td>
<td>23/10/2014</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WO 2014-170762 A3</td>
<td>26/03/2015</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US 2016-147905 Al</td>
<td>26/05/2016</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US 8326823 B2</td>
<td>04/12/2012</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US 9251166 B2</td>
<td>02/02/2016</td>
</tr>
<tr>
<td>US 2015-0169687 Al</td>
<td>18/06/2015</td>
<td>AU 2012-250970 Al</td>
<td>02/05/2013</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AU 2012-250970 B2</td>
<td>13/08/2015</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CA 2828914 A1</td>
<td>08/11/2012</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CN 103562910 A1</td>
<td>05/02/2014</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EP 2705446 A2</td>
<td>12/03/2014</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KR 10-2014-0014123 A1</td>
<td>05/02/2014</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US 2012-0284255 Al</td>
<td>08/11/2012</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US 9116955 B2</td>
<td>25/08/2015</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WO 2012-151149 A2</td>
<td>08/11/2012</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WO 2012-151149 A3</td>
<td>20/06/2013</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WO 2012-151149 A4</td>
<td>08/08/2013</td>
</tr>
</tbody>
</table>