



- (51) **International Patent Classification:**
G06F 16/27 (2019.01) *G06F 16/957* (2019.01)
G06F 16/901 (2019.01)
- (21) **International Application Number:**
PCT/US2024/037407
- (22) **International Filing Date:**
10 July 2024 (10.07.2024)
- (25) **Filing Language:** English
- (26) **Publication Language:** English
- (30) **Priority Data:**
18/352,434 14 July 2023 (14.07.2023) US
- (71) **Applicant: ORACLE INTERNATIONAL CORPORATION** [US/US]: 500 Oracle Parkway, Mailstop 50P7, Redwood Shores, California 94065 (US).
- (72) **Inventor: ÖSTERLUND, Erik**; c/o Oracle International Corporation, 500 Oracle Parkway, M/S 50P7, Redwood Shores, California 94065 (US).
- (74) **Agent: SHAH, Varun A.**; Invoke IP c/o Blackbox IP, 30 Wall Street, Ste 800, 8th Floor, New York, New York 10005 (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, CV, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IQ, IR, IS, IT, JM, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, MG, MK, MN, MU, 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, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, ZA, ZM, ZW.
- (84) **Designated States** (*unless otherwise indicated, for every kind of regional protection available*): ARIPO (BW, CV, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SC, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, ME, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:
— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*

(54) **Title:** SYNCHRONIZING DOCUMENT OBJECT MODEL TREES RESPECTIVELY MAINTAINED BY A SERVER AND A BROWSER

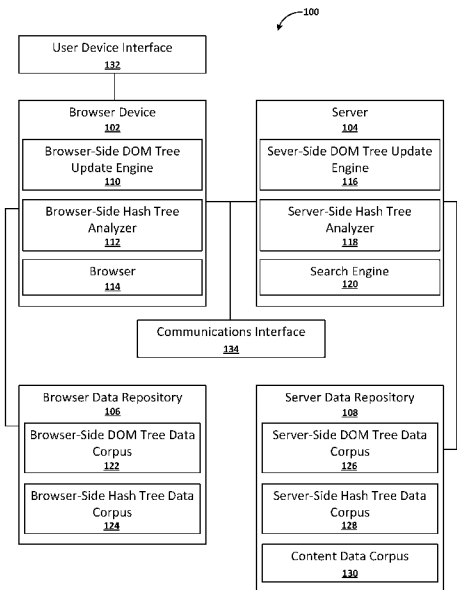


FIG. 1

(57) **Abstract:** A system synchronizes a server-side DOM tree and a browser-side DOM tree with one another. Server may receive from a browser, a hash value of the browser-side DOM tree, and a server-side update instruction for applying a first server-side update to the server-side DOM tree to synchronize with a first browser-side update by the browser to the browser-side DOM tree. The server may identify the server-side DOM tree based on the hash value. The server may execute upon the server-side DOM tree, the first server-side update and a second server-side update that is triggered by the first server-side update. The server may compute a browser-side update instruction for applying a second browser-side update to the browser-side DOM tree to synchronize with the server-side DOM tree. The server may transmit the browser-side update instruction to the browser, and the browser may apply the second browser-side update to the browser-side DOM tree.

Published:

— *with international search report (Art. 21(3))*

SYNCHRONIZING DOCUMENT OBJECT MODEL TREES RESPECTIVELY
MAINTAINED BY A SERVER AND A BROWSER

TECHNICAL FIELD

[0001] The present disclosure relates to synchronization of content between a server and a browser. More particularly, the present disclosure relates to the use of document object model (DOM) trees to synchronize content between a server and a browser.

BACKGROUND

[0002] DOM trees may be utilized to represent data, such as content for a web page, in a tree structure in which each node is an object representing a portion of the data. For example, elements of an HTML or XML document may be represented as nodes in a DOM tree. The content of the document represented by the DOM tree may be changed by programmatically accessing the DOM tree and changing various nodes of the DOM tree to reflect the desired changes. For example, nodes of the DOM tree may be added, removed, or relocated. Additionally, or in the alternative, the content represented by nodes of the DOM tree may be changed.

[0003] The content of this background section should not be construed as prior art merely by virtue of its presence in this section.

BRIEF DESCRIPTION OF THE DRAWINGS

[0004] The embodiments are shown by way of example and not by way of limitation in the figures of the accompanying drawings. It should be noted that references to “an” or “one” embodiment in this disclosure are not necessarily to the same embodiment, and they mean at least one. In the drawings:

[0005] FIG. 1 illustrates an example system for synchronizing DOM trees respectively maintained by a server and a browser, in accordance with one or more embodiments;

[0006] FIG. 2A illustrates example operations of a browser in accordance with one or more embodiments;

[0007] FIG. 2B illustrates example operations of a server in accordance with one or more embodiments;

[0008] FIG. 2C illustrates example operations associated with computing an update instruction for updating a DOM tree in accordance with one or more embodiments;

[0009] FIG. 2D illustrates example operations associated with determining a difference between hash trees in accordance with one or more embodiments; and

[0010] FIG. 3 shows a block diagram that illustrates a computer system in accordance with one or more embodiments.

DETAILED DESCRIPTION

[0011] In the following description, for the purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding. Detailed examples are described below for purposes of clarity. One or more embodiments may be practiced without these specific details. Components and/or operations described below should be understood as one specific example which may not be applicable to certain embodiments. Features described in one embodiment may be combined with features described in a different embodiment. In some examples, well-known structures and devices are described with reference to a block diagram form in order to avoid unnecessarily obscuring the present invention. Components and/or operations described below should not be construed as limiting the scope of any of the claims.

1. GENERAL OVERVIEW
2. EXAMPLE DOM TREE MANAGEMENT SYSTEM
3. EXAMPLE DOM TREE MANAGEMENT OPERATIONS
4. COMPUTER NETWORKS AND CLOUD NETWORKS
5. MICROSERVICE APPLICATIONS
6. HARDWARE OVERVIEW
7. MISCELLANEOUS; EXTENSIONS

[0012] 1. GENERAL OVERVIEW

[0013] One or more embodiments synchronize DOM trees respectively maintained by a server and a browser, for a same web page, using hash values to identify the DOM trees. The browser and server exchange information and updates for corresponding DOM trees using hash values that represent the DOM trees. For example, when a browser transmits a DOM tree update to a server, the browser transmits a hash value of the browser-side DOM tree being updated. The server identifies the corresponding server-side DOM tree based on the hash value received from the browser and performs the same update that was performed by the browser. Transmitting the update may include, for example, transmitting an instruction for specifying a modification for the DOM tree.

[0014] One or more embodiments identify differences between different versions of a DOM tree using hash trees that represent the different versions of the DOM tree. In an example, each version of a DOM tree is represented by a hash tree. The nodes of a DOM tree may be

represented by hash values of a hash tree. The hash values at each level of the hash tree are used to compute the hash values at higher levels of the hash tree until a root hash value is computed for the hash tree. A system (e.g., a server or a browser) identifies differing sections of different versions of a DOM tree based on the corresponding hash trees. For example, the system may traverse a pair of hash trees, corresponding to the different versions of a DOM tree. When the system identifies a corresponding pair of hash values from the hash trees that are identical, the system does not traverse further down the children nodes of the nodes represented by the identical hash values. When the system identifies a pair of hash values from the hash trees that are different, the server traverses down until the identification of differing hash values that correspond to leaf nodes of the hash trees. The portion of the DOM trees represented by the leaf nodes are identified as being different.

[0015] One or more embodiments improve technology associated with the use of DOM trees to track and synchronize updates between a browser and a server. For example, by using root hash values to identify the respective copies of the DOM trees, the correct DOM tree to be updated can be identified with improved accuracy. Additionally, because, the root hash values of the respective hash trees match one another when the corresponding copies of the DOM tree are synchronized, the use of root hash values to identify the DOM tree to be updated provides implicit confirmation that the copies of the DOM tree are synchronized with one another as each update is applied to the respective DOM tree. Further, because a state of the DOM tree is maintained by the browser, the browser can efficiently revert back to a prior version of the DOM tree, for example, in response to a user input selecting a “back” button.

[0016] This General Overview section is intended to provide a general overview without addressing all aspects of the present disclosure. The full scope of the presently disclosed subject matter is understood from the content of the present disclosure in its entirety.

[0017] 2. EXAMPLE DOM TREE MANAGEMENT SYSTEM

[0018] Referring now to FIG. 1, example systems are described. An example system may be utilized to perform operations associated with synchronizing DOM trees respectively maintained by a server and a browser.

[0019] A. Example System Components.

[0020] As shown in FIG. 1, a system 100 in accordance with one or more embodiments may include a browser device 102 and a server 104. The browser device 102 may be a client device. Additionally, or in the alternative, the browser device may be an end-user's computing device that executes a browser to view a web page. The system 100 may further include a browser data repository 106 associated with the browser device 102, and a server data repository 108 associated with the server 104. The browser data repository 106 may define a portion of the browser device 102. Additionally, or in the alternative, the browser data repository 106 may be communicatively coupled to the browser device 102 via a direct connection or via a network. The server data repository 108 may define a portion of the server 104. Additionally, or in the alternative, the server data repository 108 may be communicatively coupled to the server 104 via a direct connection or via a network.

[0021] The system 100 may utilize hash trees to determine an update that has been made to a DOM tree, such as an update to a server-side DOM tree and/or an update to a browser-side DOM tree. The system 100 may compute a first hash tree representing a DOM tree prior to an update, and a second hash tree representing the DOM tree subsequent to the update. As a result of the update, a first hash value representing the first hash tree will differ from a second hash value representing the second hash tree. Further, the system 100 may trace the difference between the first hash value and the second hash value to a particular node or nodes in the hash tree that differ as a result of the update. From these particular nodes of the hash tree, the system 100 may identify the corresponding nodes of the DOM tree that changed as a result of the update. The system 100 may perform this process to identify an update to a browser-side DOM tree and/or to identify an update to a server-side DOM tree. Further, upon having identified an update to a first DOM tree based on a corresponding first hash tree, the system 100 may generate an update instruction that includes one or more operations to be executed on a second DOM tree to synchronize the second DOM tree with the first DOM tree.

[0022] In one example, the browser 114 executing on the browser device 102 may identify a browser-side update to a browser-side DOM tree based on a comparison of browser-side hash trees representing the browser-side DOM tree before and after the update, respectively. The browser 114 may generate a server-side update instruction. The browser 114 may send the server-side update instruction to the server 104, and the server may execute the server-side update instruction to apply a server-side update to a server-side DOM tree corresponding to the

browser-side DOM tree. As a result of the server-side update, the server-side DOM tree may be synchronous with the browser-side DOM tree reflecting the browser-side update.

[0023] Additionally, or in the alternative, the server 104 may identify a server-side update to a server-side DOM tree based on a comparison of server-side hash trees representing the server-side Dom tree before and after the server-side update, respectively. The server 104 may generate a browser-side update instruction. The server 104 may send the browser-side update instruction to the browser 114 executing on the browser device 102, and the browser 114 may execute the browser-side update instruction to apply a browser-side update to a browser-side DOM tree corresponding to the server-side DOM tree. As a result of the browser-side update, the browser-side DOM tree may be synchronous with the server-side DOM tree reflecting the server-side update.

[0024] The browser device 102 may include a browser-side DOM tree update engine 110 and a browser-side hash tree analyzer 112. The browser device 102 may further include a browser 114. The browser 114 may be configured to display content, such as web pages on a user device interface. Additionally, the browser 114 may synchronize browser-side DOM trees with corresponding server-side DOM trees at least by executing operations associated with the browser-side DOM tree update engine 110 and/or the browser-side hash tree analyzer 112. The browser-side DOM tree update engine 110 may be configured to execute operations associated with one or more browser-side DOM trees. The browser-side hash tree analyzer 112 may be configured to execute operations associated with one or more browser-side hash trees.

[0025] The browser-side DOM tree update engine 110 may generate and/or update one or more browser-side DOM trees from time to time. The browser-side DOM tree update engine 110 may generate an initial version of a browser-side DOM tree. The initial version of the browser-side DOM tree may represent content, such as a web page, that may be provided for display by the browser 114 on a user interface device. In one example, the content, such as the web page, may be transmitted to the browser device 102 by the server 104.

[0026] The browser-side DOM tree update engine 110 may update the browser-side DOM tree, for example, in response to a browser event associated with content represented by the browser-side DOM tree. The browser-side DOM tree update engine 110 may send an update instruction to the server 104, for example, in response to a browser event, and the server 104 may update the server-side DOM tree in response to the update instruction from the browser.

Additionally, or in the alternative, the browser-side DOM tree update engine 110 may update the browser-side DOM tree in response to an update instruction from the server 104.

[0027] The browser-side hash tree analyzer 112 may generate and/or analyze one or more browser-side hash trees from time to time. The browser-side hash tree analyzer 112 may generate an initial version of a browser-side hash tree. The initial version of the browser-side hash tree may represent an initial version of a browser-side DOM tree. The browser-side hash tree analyzer 112 may generate an updated browser-side hash tree, for example, to represent an updated browser-side DOM tree. Additionally, or in the alternative, the browser-side hash tree analyzer 112 may determine an update to a browser-side DOM tree, for example, by comparing a first browser-side hash tree representing the browser-side DOM tree prior to the update to a second browser-side hash tree representing the browser-side DOM tree subsequent to the update.

[0028] The server 104 may include a server-side DOM tree update engine 116 and a server-side hash tree analyzer 118. The server 104 may synchronize server-side DOM trees with corresponding browser-side DOM trees at least by executing operations associated with the server-side DOM tree update engine 116 and/or the server-side hash tree analyzer 118. The server 104 may further include a search engine 120. The search engine 120 may be configured to execute search operations, for example in response to search queries received from the browser 114. The server 104 may execute operations associated with the server-side DOM tree update engine 116 and/or the server-side hash tree analyzer 118, for example, in connection with search queries received from the browser 114. The server-side DOM tree update engine 116 may be configured to execute operations associated with one or more server-side DOM trees. The server-side hash tree analyzer 118 may be configured to execute operations associated with one or more server-side hash trees.

[0029] The server-side DOM tree update engine 116 may generate and/or update one or more server-side DOM trees from time to time. The server-side DOM tree update engine 116 may generate an initial version of a server-side DOM tree. The initial version of the server-side DOM tree may match an initial version of a browser-side DOM tree generated by the browser 114. For example, the server 104 may send content, such as a web page, to the browser 114, and in connection with sending the content to the browser 114, the server 104 may generate the initial version of the server-side DOM tree and the browser 114 may generate the initial version of the browser-side DOM tree.

[0030] The server-side DOM tree update engine 116 may update the server-side DOM tree, for example, in response to an update instruction from the browser 114. Additionally, or in the alternative, server-side DOM tree update engine 116 may update the server-side DOM tree in response to a server event associated with content represented by the server-side DOM tree. Additionally, or in the alternative, the server-side DOM tree update engine 116 may send an update instruction to the browser 114, for example, in response to a server event, and the browser-side DOM tree update engine 110 may update the browser-side DOM tree in response to the update instruction from the server 104.

[0031] The server-side hash tree analyzer 118 may generate and/or analyze one or more server-side hash trees from time to time. The server-side hash tree analyzer 118 may generate an initial version of a server-side hash tree. The initial version of the server-side hash tree may represent an initial version of a server-side DOM tree. The server-side hash tree analyzer 118 may generate an updated server-side hash tree, for example, to represent an updated server-side DOM tree. Additionally, or in the alternative, the server-side hash tree analyzer 118 may determine an update to a server-side DOM tree, for example, by comparing a first server-side hash tree representing the server-side DOM tree prior to the update to a second server-side hash tree representing the server-side DOM tree subsequent to the update.

[0032] B. Example Events Triggering DOM Tree Updates.

[0033] The browser 114 browser executing on the browser device 102 may update a browser-side DOM tree in response to a browser event and may send a server-side update instruction to the server 104 so that the server 104 may synchronize a corresponding server-side DOM tree with the browser-side DOM tree reflecting the browser event. Additionally, or in the alternative, the server 104 may update a server-side DOM tree in response to a server event and may send a browser-side update instruction to the browser 114 so that the browser 114 may synchronize a corresponding browser-side DOM tree with the server-side DOM tree reflecting the server event. In one example, an update by the server 104 to a server-side DOM tree in response to a server-side update instruction may trigger a server event, resulting in a subsequent update to the server-side DOM tree and a corresponding browser-side update instruction being sent from the server 104 to the browser 114. Additionally, or in the alternative, an update by the browser 114 to a browser-side DOM tree in response to a browser-side update instruction may

trigger a browser event, resulting in a subsequent update to the browser-side DOM tree and a corresponding server-side update instruction being sent from the browser 114 to the server 104.

[0034] In one example, a browser event may represent a user interaction with content, such as a web page, represented by a browser-side DOM tree and displayed by the browser 114 on a user device interface. For example, the user interaction may include at least one of: activation of a text box, typing in a text box, entering a search query, clicking a button, clicking a hyperlink, creating a bookmark, refreshing a page, interacting with streaming content, scrolling content, expanding content, collapsing content, opening a tab, closing a tab, opening a window, closing a window, resizing content, changing a browser setting, loading a file, closing a file, streaming video content, streaming audio content, changing a browser setting, blocking pop-ups, allowing pop-ups, or viewing a page source, as well as combinations of these. Additionally, or in the alternative, the user interaction may include clicking a back button, clicking a forward button, or clicking a “load more” button. In one example, a button, such as a “load more” button, may be configured to cause the browser to load and/or display additional content, such as curated content. Example curated content may include search results, a photo gallery, a video gallery, an audio gallery, an online shopping gallery, as well as combinations of these. In one example, interacting with streaming content may include clicking a play button, clicking a pause button, clicking a stop button, or skipping forward or backward on a progress bar.

[0035] In one example, the server event may represent a change in content available to the browser. For example, the server event may include at least one of: a search result event, a content update event, a data refresh event, a notification event, a media event, a scroll event, a form submission event, a time-based event, a geolocation event, or an error event, as well as combinations of these.

[0036] A search result event may include the server obtaining search results. The search results may include web pages, URLs, text, images, videos, or files, as well as combinations of these. In one example, a search result event may be triggered by a browser event, such as a query.

[0037] A content update event may include updated content becoming available. For example, updated content may become available when the server 104 dynamically fetches content. In one example, the server 104 may dynamically fetch content such as image files, video files, and/or audio files. Additionally, or in the alternative, the server 104 may dynamically fetch

content such as items for display in an online store and/or an online shopping cart. The server 104 may dynamically fetch the content using at least one of: AJAX (Asynchronous JavaScript and XML), Fetch API, Axios, WebSocket, Server-Sent Events (SSE), GraphQL, or WebRTC.

[0038] A data refresh event may include content being periodically updated or refreshed. In one example, a data refresh update event may include updating at least one of: news feed data, a social media feed data, a message thread data, stock price data, financial data, e-commerce product data, ledger transaction data, sports score data, streaming data, live event data, webinar data, vehicle traffic data, flight schedule data, weather data, auction platform data, project management data, online gaming data, or time-dependent data.

[0039] A notification event may include a notification being provided to a user, for example, via a web page. The notification may pertain to a content update event or a data refresh event. In one example, a notification event may pertain to at least one of: a new messages in a messaging application, product availability in an online store, discounts in an online store, or new media in a multimedia platform.

[0040] A media streaming event may include providing streaming content, such as audio or video content. A media streaming event may be triggered by a browser playing streaming content on a web page. A media event may include at least one of: initiating streaming content, stopping streaming content, pausing streaming content, ending streaming content, updating a progress bar, or updating time data. A scroll event may occur responsive to an indication of scrolling a scrollbar associated with a web page. A scroll event may include dynamically fetching additional content, such as content associated with a feed, a gallery, or search results.

[0041] A form submission event may be triggered responsive to a user having submitted a form on a web page. The form submission event may include at least one of: processing data associated with the form, updating content, performing a search, or providing search results.

[0042] A time-based event may be triggered responsive to a specific time interval or schedule. As examples, a time-based event may include at least one of: updating content based on a time interval or schedule, or updating a timestamp. Additionally, or in the alternative, a time-based event may include fetching time-dependent data corresponding to a time, a time interval, or a schedule.

[0043] A geolocation event may occur in relation to geolocation tracking. In one example, a geolocation event may include updating content responsive to geolocation data, such as to customize and display location-specific content or services.

[0044] An error event may be triggered responsive to an error associated with fetching content, such as a broken image or a failed API request. An error event may include providing an error message or fallback content.

[0045] C. Example Hash Trees.

[0046] A hash tree, such as a browser-side hash tree or a server-side hash tree, may include a tree of nodes in which each node is associated with a hash value that uniquely represents the contents of the node and the position of the node within the tree. A hash tree may include leaf nodes, intermediate nodes, and a root node.

[0047] Each leaf node may represent an individual data block or element in the hash tree. The data associated with a leaf node may be hashed using a hash function resulting in a hash value that uniquely represents the data associated with the leaf node. Each leaf node may be assigned a unique label or identifier number based on its position in the hash tree. The unique label or identifier may include an index number and/or a more complex identifier derived, for example, from the data represented by the leaf node.

[0048] The intermediate nodes of a hash tree represent intermediate levels of the hash tree. A hash tree may include multiple layers of intermediate nodes. Each intermediate node may be associated with an intermediate hash value that is computed based on a concatenation or hashing of the hash values of its child nodes, which may include one or more leaf nodes and/or one or more lower-layer intermediate nodes. Each intermediate node may also be assigned a unique label or identifier number based on its position in the hash tree. The unique label or identifier assigned to an intermediate node may include an index number and/or a more complex identifier derived, for example, from the data represented by the child nodes associated with the intermediate node.

[0049] The root node of the hash tree represents the entire data structure. The root node may be associated with a root hash value computed based on a concatenation or hashing of the hash values of its child nodes, which may include one or more leaf nodes and/or one or more intermediate nodes. The root hash value uniquely represents the entirety of the data structure associated with the hash tree. The root node may also be assigned a unique label or identifier

number based on its position in the hash tree. The unique label or identifier assigned to the root node may include an index number and/or a more complex identifier derived, for example, from the data represented by the child nodes associated with the root node.

[0050] The nodes of a hash tree may be determined through the use of a hash function, such as a cryptographic hash function or a non-cryptographic hash function. A hash function may be selected based on one or more of the following properties: determinism, irreversibility, or avalanche effect. Determinism refers to the same input producing the same hash value. Irreversibility refers to difficulty in deriving an original input from the hash value. Avalanche effect refers to small changes in an input resulting in significant changes in the hash value. An example hash function may include a cryptographic hash function, such as an SHA-256 hash function, a RIPEMED hash function, a Whirlpool hash function, a Tiger hash function, a SipHash function, a BLAKE2 hash function, or an SHA3 (e.g., SHA3-256) hash function. By way of illustration, an SHA-256 or an SHA3-256 hash function may transform an input into a 256-bit output. Additionally, or in the alternative, an example hash function may include a non-cryptographic hash function, such as a MurmurHash hash function, a CityHash hash function, an xxHash hash function, a Fowler-Noll-Vo hash function, or a CRC32 hash function. A non-cryptographic hash function that may be particularly suitable for use in hash tables or hash-based data structures.

[0051] A browser-side hash tree may be computed for a browser-side DOM tree. The browser-side hash tree may include a browser-side root node, a plurality of browser-side intermediate nodes, and a plurality of browser-side leaf nodes. Each node in the browser-side hash tree may include a hash tree-node that represents a corresponding DOM tree-node of the browser-side DOM tree. For example, each browser-side leaf node may include a browser-side leaf hash value that represents a corresponding leaf node of the browser-side DOM tree. The browser-side hash tree may include a browser-side root hash value corresponding to a browser-side root node. The browser-side root hash value may be selected as a browser-side hash value utilized to identify the browser-side hash tree and/or the browser-side DOM tree in the browser data repository 106. For example, the browser hash value may be stored in the browser-side DOM tree data corpus 122 in association with the corresponding browser-side DOM tree. Additionally, or in the alternative, the browser hash value may be stored in the browser-side hash tree data corpus 124 in association with the corresponding browser-side hash tree.

[0052] A server-side hash tree may be computed for a server-side DOM tree. The server-side hash tree may include a server-side root node, a plurality of server-side intermediate nodes, and a plurality of server-side leaf nodes. Each node in the server-side hash tree may include a hash tree-node that represents a corresponding DOM tree-node of the server-side DOM tree. For example, each server-side leaf node may include a server-side leaf hash value that represents a corresponding leaf node of the server-side DOM tree. The server-side hash tree may include a server-side root hash value corresponding to a server-side root node. The server-side root hash value may be selected as a server-side hash value utilized to identify the server-side hash tree and/or the server-side DOM tree in the server data repository 108. For example, the server hash value may be stored in the server-side DOM tree data corpus 126 in association with the corresponding server-side DOM tree. Additionally, or in the alternative, the server hash value may be stored in the server-side hash tree data corpus 128 in association with the corresponding server-side hash tree.

[0053] D. Node Numbering Conventions.

[0054] In one example, the nodes of a hash tree, such as a browser-side hash tree and/or a server-side hash tree, may be numbered and/or renumbered according to a numbering convention. For example, the unique labels or identifiers of the nodes of a hash tree may be replaced with a node identification number determined according to the numbering convention. Additionally, or in the alternative, the node identification number for a node may be associated with, or appended to, the unique label or identifier of the node. The use of the numbering convention to assign node identification numbers to the nodes of a hash tree may assure that identity of the nodes of the hash tree is preserved from one hash tree to the next. In one example, a hash tree may be generated, and the nodes of the hash tree may be numbered and/or renumbered according to the numbering convention, after each update to a corresponding DOM tree. Additionally, or in the alternative, the nodes of a DOM tree may be numbered and/or renumbered according to the numbering convention after each update to the DOM tree.

[0055] The node identification numbers resulting from the numbering and/or renumbering of a hash tree may be utilized, at least in part, to identify an update to a DOM tree corresponding to the hash tree. For example, if a node is deleted from a DOM tree or moved to a different location in the DOM tree, the deletion or moving of the node may be identified in the corresponding hash tree based at least in part on the node identification number. For example, if a leaf node is moved

from position A to position B beneath an intermediate node, the data represented by the leaf node may still produce the same hash value, but the identification number of the node would change as a result of the move from position A to position B.

[0056] Additionally, or in the alternative, the node identification numbers resulting from the numbering and/or renumbering of a hash tree may be utilized, at least in part, to confirm that a browser-side DOM tree and a server-side DOM tree are synchronous or asynchronous with respect to one another. For example, if a browser-side hash tree representing the browser-side DOM tree has a browser-side leaf node at position A that represents a data element, and a server-side hash tree representing the server-side DOM tree has a server-side leaf node at position B that represents the data element, an asynchrony between the server-side DOM tree corresponding to the difference in position of the respective leaf nodes may be identified by differing identification numbers assigned to the respective leaf nodes. Additionally, or in the alternative, when a browser-side DOM tree and a server-side hash tree are synchronized with one another, each browser-side node in a browser-side hash tree representing the browser-side DOM tree will have a node-value and a node identification number that matches a corresponding node-value and node identification number of a server-side node in a server-side hash tree representing the browser-side DOM tree.

[0057] In one example, the numbering convention may include a depth-first search numbering technique. With a depth-first search numbering technique, a unique node identification number is assigned to each node in the hash tree. The depth-first search numbering technique may start with an initial node in the hash tree. The initial node may be the root node. The depth-first search numbering technique may assign a node identification number to the initial node, such as the root node, followed by adjacent nodes until each node in the hash tree has been numbered/renumbered. The unique node identification number assigned to the nodes in the hash tree may represent the order in which the nodes are processed during the depth-first search numbering. The depth-first search numbering technique may number/renumber nodes as deeply as possible before backtracking and numbering/renumbering nodes in other branches of the hash tree. The depth-first search numbering technique assures that all nodes in the hash tree are numbered/renumbered in a systematic manner, thereby revealing relative positions and relationships of the nodes in the hash tree.

[0058] In addition, or in the alternative to a depth-first search numbering technique, the numbering convention may include one or more of: a breadth-first search numbering technique, a preorder numbering technique, a post-order numbering technique, a level order numbering technique. A breadth-first search numbering technique may assign a node identification number to each node based on a breadth-first search traversal. With breadth-first search numbering, nodes at the same level may be numbered sequentially before moving to the next level. A preorder numbering technique may assign a unique node identification number to each node based on a preorder traversal. With a preorder traversal, the root node is numbered first, followed by its left subtree, and then its right subtree. A post-order numbering technique may assign a unique node identification number to each node based on a post-order traversal. With a post-order traversal, the left subtree is visited first, then the right subtree, and finally the root node. A level order numbering technique may assign a numerical label to each node based on a level order traversal. With a level order traversal, nodes at each level are visited from left to right before moving to the next level.

[0059] In one example, the nodes of a DOM tree, such as a browser-side DOM tree and/or a server-side DOM tree, may be numbered and/or renumbered according to the numbering convention, such as the depth-first search numbering technique. In one example, a node of a DOM tree that corresponds to a node of a hash tree may be identified at least in part by matching the node identification numbers between the hash tree and the DOM tree. The node identification numbers may be used to identify content in a DOM tree for inclusion in an update instruction. For example, when generating an update instruction, the system 100 may navigate to a node in the DOM tree based on a node identification number from a corresponding hash tree. The system may then select the content from the DOM tree at the node having the corresponding node identification number and generate an update instruction based on the content. Additionally, or in the alternative, the node identification numbers may be used to identify a location in a DOM tree for applying an update. For example, when applying an update instruction, the system 100 may identify a node identification number in the update instruction and navigate to a node in the DOM tree based on the node identification number. The system may then apply the update in accordance with the update instruction at the node of the DOM tree identified based on the node identification number. Additionally, or in the alternative, the node identification numbers resulting from the numbering and/or renumbering of a hash tree may be utilized, at least in part,

to confirm that a browser-side DOM tree and a server-side DOM tree are synchronous with one another, and/or to identify that a browser-side DOM tree and a server-side DOM tree are asynchronous with respect to one another.

[0060] In one example, the browser 114 may assign a browser-side node identification number to each of the plurality of browser-side hash tree-nodes of the browser-side hash tree at least by numbering the plurality of browser-side hash tree-nodes of the browser-side hash tree according to the numbering convention. For at least some of the browser-side hash tree-nodes, the browser 114 may replace a node identification number previously assigned to one or more of the browser-side hash tree-nodes with a new browser-side hash tree-node identification number. Additionally, or in the alternative, the server 104 may assign a server-side node identification number to each of the plurality of server-side hash tree-nodes of the server-side hash tree at least by numbering the plurality of server-side hash tree-nodes of the server-side hash tree according to the numbering convention. For at least some of the server-side hash tree-nodes, the server 104 may replace a node identification number previously assigned to one or more of the server-side hash tree-nodes with a new server-side hash tree-node identification number. When the browser-side hash tree represents a browser-side DOM tree that is synchronized with a server-side DOM tree represented by a server-side hash tree, the browser-side node-value of each browser-side hash tree-node identifiable by a corresponding browser-side node identification number, matches the server-side node-value of a corresponding server-side hash tree-node identifiable by a corresponding server-side node identification number.

[0061] E. Example Update Instructions.

[0062] The system 100 may generate update instructions for updating a browser-side DOM tree and/or a server-side DOM tree. In one example, the browser 114 executing on the browser device may generate a server-side update instruction and may send the server-side update instruction to the server 104. The server-side update instruction may include one or more node identification numbers that conform to a numbering convention, and a corresponding one or more node operations to be executing on the node identified by a respective node identification number. For example, a server-side update instruction may include a first node identification number corresponding to a first server-side node of the server-side DOM tree, and a first node operation to be executed upon the first server-side node of the server-side DOM tree. Additionally, or in the alternative, the server 104 may generate a browser-side update instruction

and may send the browser-side update instruction to the browser 114 executing on the browser device 102. The browser-side update instruction may include one or more node identification numbers that conform to the numbering convention, and a corresponding one or more node operations to be executing on the node identified by a respective node identification number. For example, a browser-side update instruction may include a first node identification number corresponding to a first browser-side node of the browser-side DOM tree, and a first node operation to be executed upon the first browser-side node of the browser-side DOM tree.

[0063] E. Data Repositories

[0064] Referring further to FIG. 1, the browser data repository 106 may include data associated with the browser device 102 and/or the browser 114. As shown, the browser data repository 106 may include a browser-side DOM tree data corpus 122 and a browser-side hash tree data corpus 124. The browser-side DOM tree data corpus 122 may include one or more browser-side DOM trees and/or data associated with one or more browser-side DOM trees. For example, the browser-side DOM tree data corpus 122 may include various iterations of a browser-side DOM tree resulting from various browser-side updates. Additionally, or in the alternative, the browser-side DOM tree data corpus 122 may include one or more update logs corresponding to respective browser-side DOM trees. The update log may include updates to a browser-side DOM tree corresponding to various browser-side updates and/or browser-side update instructions received from the server 104. Additionally, or in the alternative, the update log may include server-side update instructions computed by the browser 114 and/or sent to the server 104.

[0065] The browser-side hash tree data corpus 124 may include one or more browser-side hash trees and/or data associated with one or more browser-side hash trees. For example, the browser-side hash tree data corpus 124 may include one or more iterations of a browser-side hash tree resulting from various browser-side updates. Additionally, or in the alternative, the browser-side hash tree data corpus 124 may include one or more browser-side hash values corresponding to a browser-side hash tree. The browser-side hash values may include root hash values, intermediate hash values, and/or leaf hash values. Additionally, or in the alternative, the browser-side hash tree data corpus 124 may include one or more update logs corresponding to respective browser-side hash tree. The update log may include updates to a browser-side hash tree corresponding to various browser-side updates and/or browser-side update instructions.

[0066] In one example, the browser data repository 106 may include one or more browser-side hash values, such as one or more root hash values, relationally associated with a corresponding browser-side hash tree and/or a corresponding browser-side DOM tree represented by the browser-side hash tree. A browser-side hash tree and/or a browser-side DOM tree may be identified by a corresponding browser-side hash value, such as a root hash value. The browser-side DOM tree data corpus 122 and the browser-side hash tree data corpus 124 may represent common or separate portions of the browser data repository 106. Data items in the browser-side DOM tree data corpus 122 may be relationally associated with data items in the browser-side hash tree data corpus 124. The browser-side DOM tree data corpus 122 and the browser-side hash tree data corpus 124 may be updated from time to time, for example, responsive to browser-side update instructions.

[0067] In one or more embodiments, the browser data repository 106 may include any type of storage unit and/or device (e.g., a file system, database, collection of tables, or any other storage mechanism) for storing data. Further, the browser data repository 106 may include multiple different storage units and/or devices. The multiple different storage units and/or devices may or may not be of the same type or located at the same physical site. Further, the browser data repository 106 may be implemented or executed on the same computing system as the browser device 102. Additionally, or in the alternative, the browser data repository 106 may be implemented or executed on a computing system separate from the browser device 102.

[0068] The server data repository 108 may include data associated with the server 104. As shown, the server data repository 108 may include a server-side DOM tree data corpus 126 and a server-side hash tree data corpus 128. The server-side DOM tree data corpus 126 may include one or more server-side DOM trees and/or data associated with one or more server-side DOM trees. For example, the server-side DOM tree data corpus 126 may include various iterations of a server-side DOM tree resulting from various server-side updates. Additionally, or in the alternative, the server-side DOM tree data corpus 126 may include one or more update logs corresponding to respective server-side DOM trees. The update log may include updates to a server-side DOM tree corresponding to various server-side updates and/or server-side update instructions received from the browser 114 executing on the browser device 102. Additionally, or in the alternative, the update log may include browser-side update instructions computed by the server 104 and/or sent to the browser device 102.

[0069] The server-side hash tree data corpus 128 may include one or more server-side hash trees and/or data associated with one or more server-side hash trees. For example, the server-side hash tree data corpus 128 may include one or more iterations of a server-side hash tree resulting from various server-side updates. Additionally, or in the alternative, the server-side hash tree data corpus 128 may include one or more server-side hash values corresponding to a server-side hash tree. The server-side hash values may include root hash values, intermediate hash values, and/or leaf hash values. Additionally, or in the alternative, the server-side hash tree data corpus 128 may include one or more update logs corresponding to respective server-side hash tree. The update log may include updates to a server-side hash tree corresponding to various server-side updates and/or server-side update instructions.

[0070] In one example, the server data repository 108 may include one or more server-side hash values, such as one or more root hash values, relationally associated with a corresponding server-side hash tree and/or a corresponding server-side DOM tree represented by the server-side hash tree. A server-side hash tree and/or a server-side DOM tree may be identified by a corresponding server-side hash value, such as a root hash value. The server-side DOM tree data corpus 126 and the server-side hash tree data corpus 128 may represent common or separate portions of the browser data repository 106. Data items in the server-side DOM tree data corpus 126 may be relationally associated with data items in the server-side hash tree data corpus 128. The server-side DOM tree data corpus 126 and the server-side hash tree data corpus 128 may be updated from time to time, for example, responsive to server-side update instructions.

[0071] In one or more embodiments, the browser data repository 106 may include any type of storage unit and/or device (e.g., a file system, database, collection of tables, or any other storage mechanism) for storing data. Further, the browser data repository 106 may include multiple different storage units and/or devices. The multiple different storage units and/or devices may or may not be of the same type or located at the same physical site. Further, the browser data repository 106 may be implemented or executed on the same computing system as the browser device 102. Additionally, or in the alternative, the browser data repository 106 may be implemented or executed on a computing system separate from the browser device 102.

[0072] The server data repository 108 may further include a content data corpus 130. The content data corpus 130 may include content that may be provided to the browser device 102, for example, for display by the browser 114 on a web page. In one example, the content may include

streaming data, such as audio and/or video streaming data. Additionally, or in the alternative, the content data corpus 130 may include web pages, URLs, text files, image files, audio files, video files, or data files, as well as combinations of these. The server 104 may search the content data corpus 130, for example, responsive to a query from the browser 114 executing on the browser device 102.

[0073] In one or more embodiments, the server data repository 108 may include any type of storage unit and/or device (e.g., a file system, database, collection of tables, or any other storage mechanism) for storing data. Further, the server data repository 108 may include multiple different storage units and/or devices. The multiple different storage units and/or devices may or may not be of the same type or located at the same physical site. Further, the server data repository 108 may be implemented or executed on the same computing system as the server 104. Additionally, or in the alternative, the server data repository 108 may be implemented or executed on a computing system separate from the server 104.

[0074] Referring again to FIG. 1, the system 100 may include a user device interface 132 communicatively coupled or couplable with the browser device 102. The user device interface 132 may include hardware and/or software configured to facilitate interactions between a user and various aspects of the system 100. The user device interface 132 may render user interface elements and receive input via user interface elements. For example, the user device interface 132 may display content from the browser 114, such as one or more web pages and associated content. Additionally, or in the alternative, the user device interface 132 may provide query inputs to the browser 114, such as via a web page displayed by the user device interface 132. The user device interface 132 may further display search results, such as via a web page displayed by the user device interface 132, responsive to a query input. Examples of interfaces include a GUI, a command line interface (CLI), a haptic interface, or a voice command interface. Examples of user interface elements include checkboxes, radio buttons, dropdown lists, list boxes, buttons, toggles, text fields, date and time selectors, command lines, sliders, pages, or forms. Any one or more of these interfaces or interface elements may be utilized by the user device interface 132.

[0075] In an embodiment, different components of a user device interface 132 are specified in different languages. The behavior of user interface elements is specified in a dynamic programming language, such as JavaScript. The content of user interface elements is specified in a markup language, such as hypertext markup language (HTML) or XML User Interface

Language (XUL). The layout of user interface elements is specified in a style sheet language, such as Cascading Style Sheets (CSS). Alternatively, the user device interface 132 may be specified in one or more other languages, such as Java, C, or C++.

[0076] Referring again to FIG. 1, the system 100 may include at least one communications interface 134 communicatively coupled or couplable with the browser device 102 and/or the server 104. Additionally, or in the alternative, the at least one communications interface 134 may be communicatively coupled or couplable with the browser data repository 106 and/or the server data repository 108. The at least one communications interface 134 may include hardware and/or software configured to transmit data between respective components of the system 100 and/or to transmit data to and/or from the system 100. For example, a communications interface 134 may transmit and/or receive data between and/or among at least one of: the browser device 102 and the server 104, the browser device 102 and the browser data repository 106, or the server 104 and the server data repository 108.

[0077] In one example, the browser device 102 and/or the server 104 may be respectively implemented on one or more digital devices. The term “digital device” generally refers to any hardware device that includes a processor. A digital device may refer to a physical device executing an application or a virtual machine. Examples of digital devices include a computer, a tablet, a laptop, a desktop, a netbook, a server, a web server, a network policy server, a proxy server, a generic machine, a function-specific hardware device, a hardware router, a hardware switch, a hardware firewall, a hardware firewall, a hardware network address translator (NAT), a hardware load balancer, a mainframe, a television, a content receiver, a set-top box, a printer, a mobile handset, a smartphone, a personal digital assistant (PDA), a wireless receiver and/or transmitter, a base station, a communication management device, a router, a switch, a controller, an access point, and/or a browser device.

[0078] In one or more embodiments, the system 100 may include more or fewer components than the components described with reference to FIG. 1. The components described with reference to FIG. 1 may be local to or remote from each other. The components described with reference to FIG. 1 may be implemented in software and/or hardware. Each component may be distributed over multiple applications and/or machines. Multiple components may be combined into one application and/or machine. Operations described with respect to one component may instead be performed by another component.

[0079] 4. EXAMPLE DOM TREE MANAGEMENT OPERATIONS

[0080] Referring now to FIGS. 2A-2D, example operations 200 associated with an example system are further described. The operations 200 described with reference to FIGS. 2A-2D may be associated with one or more components of the system. For example, the operations 200 described with reference to FIG. 2A may include operations of a browser device and/or operations of a browser executing on a browser device. Additionally, or in the alternative, the operations 200 described with reference to FIG. 2B may include operations of a server. The operations 200 described with reference to FIG. 2C and 2D may include operations of a browser and/or operations of a server. One or more operations described with reference to FIGS. 2A-2D may be modified, rearranged, or omitted all together. Accordingly, the particular sequence of operations described with reference to FIGS. 2A-2D should not be construed as limiting the scope of one or more embodiments.

[0081] A. Example Browser Operations.

[0082] As shown in FIG. 2A, example operations 200 may include operations associated with updating a browser-side DOM tree. In one example, the operations 200 may include, at block 202, identifying, by a browser executing on a browser device, a first browser-side update to a browser-side DOM tree stored by the browser. The first browser-side update may include a first set of one or more browser-side operations on the browser-side DOM tree corresponding to a browser event associated with content represented by the browser-side DOM tree. At block 204, the operations may include identifying, by the browser, a browser-side hash value corresponding to the browser-side DOM tree prior to the first browser-side update. The browser-side hash value may be identified in a browser data repository, such as in a browser-side hash tree data corpus. The browser-side hash value may be a root hash value of a browser-side hash tree representing the browser-side DOM tree prior to the first browser-side update. The browser-side hash tree may be generated by the browser and the root hash value of the browser-side hash tree may be selected as the browser-side hash value.

[0083] At block 206, the operations 200 may include computing, by the browser, a server-side update instruction for applying a first server-side update to a server-side DOM tree. The server-side update instruction may include a first set of one or more server-side operations on the server-side DOM tree. At block 208, the operations may include transmitting, by the browser to

the server, (a) the browser-side hash value, and (b) the server-side update instruction. The browser-side hash value may correspond to a server-side hash value stored by the server in a server data repository. The browser-side hash value may match the server-side hash value. Thus, the server may identify the server-side DOM tree based on the browser-side hash value. Upon identifying the server-side DOM tree, the server may execute the server-side update instruction to apply the first server-side update to the server-side DOM tree. The server-side DOM tree may correspond to the browser-side DOM tree. Upon having executed the server-side update instruction to apply the first server-side update to the server-side DOM tree, the server-side DOM tree may match the browser-side DOM tree resulting from the first browser-side update.

[0084] At block 210, the operations 200 may include receiving, by the browser from the server, (a) a server-side hash value representing the server-side DOM tree resulting from the first server-side update, and (b) a browser-side update instruction for applying a second browser-side update to the browser-side DOM tree. At block 212, the operations 200 may include browser identifying the browser-side DOM tree based on the server-side hash value. The browser-side DOM tree identified at block 212 may be the browser-side DOM tree resulting from the first browser-side update. The server-side hash value may be a root hash value of a server-side hash tree. The server-side hash tree may be generated by the server and the root hash value of the server-side hash tree may be selected as the server-side hash value. The server-side hash value may correspond to a browser-side hash value stored by the browser in the browser data repository. In one example, prior to receiving the server-side hash value from the server at block 210, the browser may compute the browser-side hash value that represents the browser-side DOM tree resulting from the first browser-side update. The browser-side hash value representing the browser-side DOM tree resulting from the first browser-side update may match the server-side hash value. Thus, the browser may identify the browser-side DOM tree based on the server-side hash value. For example, the browser-side DOM tree resulting from the first browser-side update may be stored in the browser data repository 106, such as in the browser-side DOM tree data corpus 122. Subsequently, the browser-side DOM tree resulting from the first browser-side update may be retrieved, for example, based on the corresponding server-side hash value, at block 212.

[0085] Upon identifying the browser-side DOM tree, at block 214, the operations may include the browser executing, based on the browser-side update instruction, the second browser-

side update to the browser-side DOM tree. The browser-side update instruction may include a second set of one or more browser-side operations on the browser-side DOM tree. The browser-side update instruction may be triggered by the server having executed the first server-side update on the server-side DOM tree. For example, the first server-side update on the server-side DOM tree may trigger a second server-side update on the server-side DOM tree, and responsive to the second server-side update, the server may compute the browser-side update instruction for applying the second browser-side update, for example, to synchronize the browser-side DOM tree with the server-side DOM tree following the second server-side update to the server-side DOM tree. Upon having executed the browser-side update instruction to apply the second browser-side update to the browser-side DOM tree, the browser-side DOM tree may match the server-side DOM tree resulting from the second browser-side update.

[0086] At block 216, the operations 200 may include determining whether there is another browser-side operation in the browser-side update instruction to be executed on the browser-side DOM tree. When there is another browser-side operation in the browser-side update instruction, the operations 200 may return to block 214, for example, to further execute the browser-side update instruction. When each of the browser-side operations have been executed, the operations 200 may return to block 202, for example, to identify another browser-side update to the browser-side DOM tree corresponding to another browser event associated with content represented by the browser-side DOM tree.

[0087] B. Example Server Operations.

[0088] Referring now to FIG. 2B, the example operations 200 may include operations associated with updating a server-side DOM tree. As shown in FIG. 2B, in one example, the operations 200 may include, at block 222, receiving, by a server from a browser executing on a browser device, (a) a browser-side hash value corresponding to a browser-side DOM tree stored by the browser, and (b) a server-side update instruction for applying a first server-side update to a server-side DOM tree stored by the server. The browser-side hash value may be a root hash value of a browser-side hash tree. The browser-side hash tree may be generated by the browser and the root hash value of the browser-side hash tree may be selected as the browser-side hash value. In one example, the browser-side hash value and the server-side update instruction received by the server may be transmitted to the server by the browser at block 206, as described with reference to FIG. 2A. In one example, the first server-side update may represent an

indication of a user interaction with the browser. The first server-side update may be configured to apply one or more updates to the server-side DOM tree to reflect the user interaction with the browser. Additionally, or in the alternative, the first server-side update may represent a content request from the browser.

[0089] At block 224, the operations 200 may include the server identifying the server-side DOM tree based on the browser-side hash value. The browser-side hash value may correspond to a server-side hash value stored by the server in a server data repository. The browser-side hash value may match the server-side hash value. Thus, the server may identify the server-side DOM tree based on the browser-side hash value.

[0090] Upon identifying the server-side DOM tree, at block 226, the operations 200 may include the server executing, based on the server-side update instruction, the first server-side update to the server-side DOM tree. The server-side update instruction may include a first set of one or more server-side operations on the server-side DOM tree. The server-side update instruction may be triggered by a browser event associated with content represented by the browser-side DOM tree. Responsive to the browser event, the browser may compute the server-side update instruction for applying the server-side update, for example, to synchronize the server-side DOM tree with the browser-side DOM tree following an update to the browser-side DOM tree corresponding to the browser event. The server-side DOM tree may correspond to the browser-side DOM tree. Upon having executed the server-side update instruction to apply the server-side update to the server-side DOM tree, the server-side DOM tree may match the browser-side DOM tree resulting from the update to the browser-side DOM tree corresponding to the browser event.

[0091] At block 228, the operations 200 may include determining whether there is another server-side operation in the server-side update instruction to be executed on the server-side DOM tree. When there is another server-side operation in the server-side update instruction, the operations 200 may return to block 226, for example, to further execute the server-side update instruction. When each of the server-side operations have been executed, the operations 200 may proceed to block 230.

[0092] At block 230, the operations 200 may include the server computing a server-side hash value representing the server-side DOM tree resulting from the first server-side update. The server-side hash value may be a root hash value of a server-side hash tree representing the

server-side DOM tree resulting from the first server-side update. The server-side hash tree may be generated by the server and the root hash value of the server-side hash tree may be selected as the server-side hash value. The server-side hash value computed at block 230 may be utilized to identify the server-side DOM tree resulting from the first server-side update. For example, the server-side DOM tree resulting from the first server-side update may be stored in the server data repository 108, such as in the server-side DOM tree data corpus 126. Subsequently, the server-side DOM tree resulting from the first server-side update may be retrieved based on the corresponding server-side hash value.

[0093] At block 232, the operations may include the server executing a second server-side update to the server-side DOM tree. The second server-side update may include a second set of one or more server-side operations on the server-side DOM tree. In one example, the second server-side update may be triggered by the first server-side update. For example, the first server-side update to the server-side DOM tree executed by the server at block 226 may represent a content request from the browser, as applied to the server-side DOM tree, and the second server-side update to the server-side DOM tree executed by the server at block 232 may represent a response to the content request as applied to the server-side DOM tree. The response as applied to the server-side DOM tree may include a change with respect to at least one item of content as represented by the server-side DOM tree.

[0094] At block 234, the operations 200 may include determining whether there is another server-side operation in the second server-side update instruction to be executed on the server-side DOM tree. When there is another server-side operation in the server-side update, the operations 200 may return to block 232, for example, to further execute the second server-side update. When each of the server-side operations have been executed, the operations 200 may proceed to block 236.

[0095] At block 236, the operations 200 may include the server computing a browser-side update instruction for applying a second browser-side update to the browser-side DOM tree. The browser-side update instruction may include a second set of one or more browser-side operations on the browser-side DOM tree. At block 238, the operations may include the server transmitting, to the browser executing on the browser device, (a) the server-side hash value representing the server-side DOM tree resulting from the first server-side update, and (b) the browser-side update

instruction. The server-side hash value may correspond to a browser-side hash value stored by the browser in the browser data repository.

[0096] In one example, the second browser-side update to the browser-side DOM tree may represent a response to a content request as applied to the browser-side DOM tree. The response as applied to the browser-side DOM tree may include a change with respect to the at least one item of content as represented by the browser-side DOM tree. For example, the first server-side update to the server-side DOM tree executed by the server at block 226 may represent a content request from the browser, as applied to the server-side DOM tree, and the second server-side update to the server-side DOM tree executed by the server at block 232 may represent a response to the content request as applied to the server-side DOM tree, and the second browser-side update to the browser-side DOM tree may represent a response to a content request as applied to the browser-side DOM tree.

[0097] In one example, prior to the server transmitting the server-side hash value to the browser at block 236, the browser may compute the browser-side hash value that represents the browser-side DOM tree resulting from the first browser-side update. The browser-side hash value representing the browser-side DOM tree resulting from the first browser-side update may match the server-side hash value transmitted by the server at block 236. Thus, the browser may identify the browser-side DOM tree based on the server-side hash value. For example, the browser-side DOM tree resulting from the first browser-side update may be stored in the browser data repository 106, such as in the browser-side DOM tree data corpus 122. Subsequently, the browser-side DOM tree resulting from the first browser-side update may be retrieved, for example, based on the corresponding server-side hash value transmitted by the server at block 236.

[0098] The server-side hash value transmitted to the browser at block 236 may match the browser-side hash value representing the browser-side DOM tree resulting from the first browser-side update. Thus, the browser may identify the browser-side DOM tree resulting from the first browser-side update based on the server-side hash value transmitted to the browser at block 236. Upon identifying the browser-side DOM tree, the browser may execute the browser-side update instruction to apply the second browser-side update to the browser-side DOM tree. Upon having executed the browser-side update instruction to apply the second server-side update to the browser-side DOM tree, the browser-side DOM tree may match the server-side DOM tree

resulting from the second server-side update. In one example, the server-side hash value and the browser-side update instruction transmitted by the server at block 238 may correspond to the server-side hash value and the browser-side update instruction received by the browser at block 210, as described with reference to FIG. 2A.

[0099] C. Computing Update Instructions.

[0100] As shown in FIG. 2C, example operations 200 may include operations associated with computing an update instruction for updating a DOM tree. The operations shown in FIG. 2C reflect computation of a browser-side update instruction. In one example, the operations described with reference to FIG. 2C may be performed in connection with computing the browser-side update instruction at block 236, as described with reference to Fig. 2B. The browser-side update instruction may be computed by a server and sent to a browser. Upon receiving the browser-side update instruction from the server, the browser may execute the browser-side update instruction to apply one or more browser-side updates to a browser-side DOM tree. Additionally, or in the alternative, it will be appreciated that operations similar to those shown in FIG. 2C may be performed by a browser to compute a server-side update instruction to be send to a server, and that upon receiving a server-side update instruction from the browser, the server may execute the server-side update instruction to apply one or more server-side updates to a server-side DOM tree. For example, the operations described with reference to FIG. 2C may be performed in connection with computing the server-side update instruction at block 206, as described with reference to Fig. 2A.

[0101] As shown in FIG. 2C, in one example, the operations 200 may include, at block 252, generating a first server-side hash tree representing a server-side DOM tree resulting from a first server-side update. The first server-side hash tree may include a first set of leaf nodes. At block 254, the operations 200 may include generating a second server-side hash tree representing the server-side DOM tree resulting from a second server-side update. The second server-side hash tree may include a second set of leaf nodes.

[0102] At block 256, the operations 200 may include comparing a first leaf node, of the first set of leaf nodes, to a second leaf node, of the second set of leaf nodes. The first leaf node may occupy a first leaf position of the first server-side hash tree, the second leaf node may occupy a second leaf position of the second server-side hash tree, and the first leaf position may correspond to the second leaf position.

[0103] At block 258, the operations 200 may include determining whether the first leaf node matches the second leaf node. When the first leaf node matches the second leaf node, the operations 200 may return to block 256, for example, to compare another set of leaf nodes. When the first leaf node differs from the second leaf node, the operations 200 may proceed to block 260. For example, at block 258, the operations may include determining a first difference between a first leaf node, of the first set of leaf nodes, and a second leaf node, of the second set of leaf nodes. Based on at least the first difference between the first leaf node and the second leaf node, the operations 200 may proceed to block 260.

[0104] At block 260, the operations 200 may include identifying, in the server-side DOM tree resulting from the first server-side update, a first server-side DOM tree-node corresponding to the first leaf node of the first server-side hash tree. At block 262, the operations 200 may include identifying, in the server-side DOM tree resulting from the second server-side update, a second server-side DOM tree-node corresponding to the second leaf node of the second server-side hash tree.

[0105] At block 264, the operations 200 may include determining a second difference between the first server-side DOM tree-node and the second server-side DOM tree-node. At block 266, the operations 200 may include determining a browser-side update to the browser-side DOM tree based on the second difference between the first server-side DOM tree-node and the second server-side DOM tree-node. At block 268, the operations 200 may include generating a browser-side update instruction that includes the browser-side update.

[0106] The browser-side update instruction may include at least a first instruction configured to cause the browser to apply the second browser-side update. In one example, upon the browser having applied the browser-side update to the browser-side DOM tree, a first browser-side DOM tree-node, corresponding to the first server-side DOM tree-node, is modified to match the second server-side DOM tree-node.

[0107] Referring now to FIG. 2D, example operations 200 may include operations associated with determining a difference between hash trees, for example, in connection with computing an update instruction for updating a DOM tree, such as described with reference to FIG. 2C. The operations shown in FIG. 2D reflect server-side hash trees, for example, in association with computation of a browser-side update instruction. In one example, the operations described with reference to FIG. 2D may be performed in connection with computing the browser-side update

instruction, for example, as described with reference to block 236 of FIG. 2B. Additionally, or in the alternative, it will be appreciated that operations similar to those shown in FIG. 2D may be performed in connection with computing the server-side update instruction, for example, as described with reference to block 206 of FIG. 2A.

[0108] In one example, the operations described with reference to FIG. 2D may be performed with respect to a server determining a difference between a first server-side hash tree, representing a server-side DOM tree resulting from a first server-side update, and a second server-side hash tree, representing the server-side DOM tree resulting from a second server-side update. In one example, based on the difference between the first server-side hash tree and the second server-side hash tree, the server may generate a browser-side update instruction for a browser to apply an update a browser-side DOM tree. The server may transmit the browser side update instruction to a browser, and the browser may execute the browser-side update instruction to apply the update to the browser-side DOM tree. In one example, the first server-side hash tree may be the first server-side hash tree described with reference to block 252 of FIG. 2C. In one example, the second server-side hash tree may be the second server-side hash tree described with reference to block 254 of FIG. 2C.

[0109] As shown in FIG. 2D, the operations 200 may include, at block 272, comparing a root node of a first server-side hash tree to a corresponding root node of a second server-side hash tree. At block 274, the operations may include determining whether the root node of the first server-side hash tree matches the root node of the second server-side hash tree. When the root nodes match one another, the first server-side hash tree matches the second server-side hash tree. When the root nodes are different from one another, the first server-side hash tree is different from the second server-side hash tree with respect to at least one leaf node and/or at least one intermediate node. When the first root node matches the second root node, the operations 200 may return to block 272, for example, to compare another set of root nodes corresponding to another set of hash trees. When the first root node differs from the second root node, the operations 200 may proceed to block 276, for example, to determine one or more leaf nodes and/or intermediate nodes that differ as between the first server-side hash tree and the second server-side hash tree.

[0110] In one example, the first server-side hash tree may include a first server-side root node having a first root node-value, and the second server-side hash tree may include a second

server-side root node having a second root node-value. The operations 200 may include, at block 272, comparing the first root node-value of the first server-side root node to the second root node-value of the second server-side root node. At block 274, the operations 200 may include determining that the first root node-value differs from the second root node-value. Based on the first root node-value differing from the second root node-value, the operations 200 may proceed to block 276.

[0111] At block 276, the operations 200 may include comparing an intermediate node of the first server-side hash tree to a corresponding intermediate node of the second server-side hash tree. At block 278, the operations 200 may include determining whether the intermediate nodes match one another or whether the intermediate nodes differ from one another. When the intermediate nodes match, the operations 200 may return to block 276, for example, to compare another set of intermediate nodes. When the intermediate nodes differ, the operations may proceed to block 280. Additionally, or in the alternative, when there are no further intermediate nodes, the operations 200 may return to block 272.

[0112] In one example, the first server-side hash tree may include at least a first server-side intermediate node and a second server-side intermediate node respectively extending from the first server-side root node. The first server-side intermediate node may have a first intermediate node-value, and the second server-side intermediate node may have a second intermediate node-value. The second server-side hash tree may include at least a third server-side intermediate node and a fourth server-side intermediate node respectively extending from the second server-side root node. The third server-side intermediate node may have a third intermediate node-value, and the fourth server-side intermediate node may have a fourth intermediate node-value. The first server-side intermediate node may occupy a first intermediate position of the first server-side hash tree, and the third server-side intermediate node may occupy a third intermediate position of the second server-side hash tree. The first intermediate position of the first server-side hash tree may correspond to the third intermediate position of the second server-side hash tree. The second server-side intermediate node may occupy a second intermediate position of the first server-side hash tree, and the fourth server-side intermediate node may occupy a fourth intermediate position of the second server-side hash tree. The second intermediate position of the first server-side hash tree may correspond to the fourth intermediate position of the second server-side hash tree.

[0113] In one example, at block 276, the operations 200 may include comparing the first intermediate node-value of the first server-side intermediate node to the third intermediate node-value of the third server-side intermediate node, and at block 278, the operations 200 may include determining that the first intermediate node-value matches the third intermediate node-value. Based on the first intermediate node-value matching the third intermediate node-value, the operations 200 may return to block 276.

[0114] In one example, the first server-side hash tree and the second server-side hash tree may respectively include one or more lower-level intermediate nodes and/or one or more leaf nodes respectively extending from the first server-side intermediate node and the third server-side intermediate node. Based on the first server-side intermediate node matching the third server-side intermediate node, the operations 200 may include determining that any lower-level intermediate nodes and/or any leaf nodes respectively extending from the first server-side intermediate node and the third server-side intermediate node also respectively match one another. For example, based on the first server-side intermediate node matching the third server-side intermediate node, the operations 200 may omit comparing any lower-level intermediate nodes and/or any leaf nodes respectively extending from the first server-side intermediate node and the third server-side intermediate node.

[0115] Upon returning to block 276, the operations 200 may include comparing the second intermediate node-value of the first server-side intermediate node to the fourth intermediate node-value of the fourth server-side intermediate node, and at block 278, determining that the second intermediate node-value differs from the fourth intermediate node-value. Based on the second intermediate node-value differing from the fourth intermediate node-value, the operations may proceed to block 280.

[0116] At block 280, the operations 200 may include comparing a leaf node of the first server-side hash tree to a corresponding leaf node of the second server-side hash tree. At block 282, the operations 200 may include determining whether the leaf nodes match one another or whether the leaf nodes differ from one another. When the leaf nodes match, the operations 200 may return to block 280, for example, to compare another set of leaf nodes. When the leaf nodes differ, the operations may proceed to block 284. Additionally, or in the alternative, when there are no further leaf nodes, the operations 200 may return to block 276.

[0117] In one example, the first server-side hash tree may include a first server-side leaf node and a second server-side leaf node extending from the second server-side intermediate node. The first server-side leaf node may have a first leaf node-value, and the second server-side leaf node may have a second leaf node-value. The second server-side hash tree may include a third server-side leaf node and a fourth server-side leaf node extending from the fourth server-side intermediate node. The third server-side leaf node may have a third leaf node-value, and the fourth server-side leaf node may have a fourth leaf node-value. The first server-side leaf node may occupy a first leaf position of the first server-side hash tree, and the third server-side leaf node may occupy a third leaf position of the second server-side hash tree. The first leaf position of the first server-side hash tree may correspond to the third leaf position of the second server-side hash tree. The second server-side leaf node may occupy a second leaf position of the first server-side hash tree, and the fourth server-side leaf node may occupy a fourth leaf position of the second server-side hash tree. The second leaf position of the first server-side hash tree may correspond to the fourth leaf position of the second server-side hash tree.

[0118] In one example, at block 280, the operations 200 may include comparing the first leaf node-value of the first server-side leaf node to the third leaf node-value of the third server-side leaf node, and at block 282, the operations 200 may include determining that the first leaf node-value matches the third leaf node-value. Based on the first leaf node-value matching the third leaf node-value, the operations 200 may return to block 280.

[0119] Upon returning to block 280, the operations 200 may include comparing the second leaf node-value of the second server-side leaf node to the fourth leaf node-value of the fourth server-side leaf node, and at block 282, the operations 200 may include determining that the second leaf node-value differs from the fourth leaf node-value. Based on the second leaf node-value differing from the fourth leaf node-value, the operations 200 may proceed to block 284.

[0120] At block 284, the operations 200 may include generating a browser-side update instruction for a browser to update a browser-side DOM tree corresponding to the server-side DOM tree represented by the first server-side hash tree. The browser-side update instruction may include a browser-side update generated based on the difference between a leaf node of the first server-side hash tree and the corresponding leaf node of the second server-side hash tree. Upon having generated the browser-side update instruction, the operations 200 may proceed to block 286. At block 286, the operations 200 may include determining whether there is another leaf

node associated with the first server-side hash tree and/or the second server-side hash tree. When there is another leaf node, the operations 200 may return to block 280, for example, to compare another set of leaf nodes. When there are no further leaf nodes, the operations 200 may proceed to block 288. At block 288, the operations 200 may include determining whether there is another intermediate node associated with the first server-side hash tree and/or the second server-side hash tree. When there is another intermediate node, the operations 200 may return to block 276, for example, to compare another set of intermediate nodes. When there are no further intermediate nodes, the operations 200 may return to block 272, for example, to compare another set of root nodes corresponding to another set of hash trees.

[0121] In one example, at block 284, the operations 200 may include generating a browser-side update instruction for a browser to apply a browser-side update to a browser-side DOM tree based on the difference between the second leaf node-value of the second server-side leaf node and the fourth leaf node-value of the fourth server-side leaf node. The operations 200 for generating the browser-side update instruction may include one or more of the operations 200 described with reference to FIG. 2C, including, for example, the operations described with reference to blocks 260-268.

[0122] 4. COMPUTER NETWORKS AND CLOUD NETWORKS

[0123] In one or more embodiments, a computer network provides connectivity among a set of nodes. The nodes may be local to and/or remote from each other. The nodes are connected by a set of links. Examples of links include a coaxial cable, an unshielded twisted cable, a copper cable, an optical fiber, and a virtual link.

[0124] A subset of nodes implements the computer network. Examples of such nodes include a switch, a router, a firewall, and a network address translator (NAT). Another subset of nodes uses the computer network. Such nodes (also referred to as “hosts”) may execute a client process and/or a server process. A client process makes a request for a computing service (such as, execution of a particular application, and/or storage of a particular amount of data). A server process responds by executing the requested service and/or returning corresponding data.

[0125] A computer network may be a physical network, including physical nodes connected by physical links. A physical node is any digital device. A physical node may be a function-specific hardware device, such as a hardware switch, a hardware router, a hardware firewall, and

a hardware NAT. Additionally or alternatively, a physical node may be a generic machine that is configured to execute various virtual machines and/or applications performing respective functions. A physical link is a physical medium connecting two or more physical nodes. Examples of links include a coaxial cable, an unshielded twisted cable, a copper cable, and an optical fiber.

[0126] A computer network may be an overlay network. An overlay network is a logical network implemented on top of another network (such as, a physical network). Each node in an overlay network corresponds to a respective node in the underlying network. Hence, each node in an overlay network is associated with both an overlay address (to address to the overlay node) and an underlay address (to address the underlay node that implements the overlay node). An overlay node may be a digital device and/or a software process (such as, a virtual machine, an application instance, or a thread). A link that connects overlay nodes is implemented as a tunnel through the underlying network. The overlay nodes at either end of the tunnel treat the underlying multi-hop path between them as a single logical link. Tunneling is performed through encapsulation and decapsulation.

[0127] In an embodiment, a client may be local to and/or remote from a computer network. The client may access the computer network over other computer networks, such as a private network or the Internet. The client may communicate requests to the computer network using a communications protocol, such as Hypertext Transfer Protocol (HTTP). The requests are communicated through an interface, such as a client interface (such as a web browser), a program interface, or an application programming interface (API).

[0128] In an embodiment, a computer network provides connectivity between clients and network resources. Network resources include hardware and/or software configured to execute server processes. Examples of network resources include a processor, a data storage, a virtual machine, a container, and/or a software application. Network resources are shared amongst multiple clients. Clients request computing services from a computer network independently of each other. Network resources are dynamically assigned to the requests and/or clients on an on-demand basis. Network resources assigned to each request and/or client may be scaled up or down based on, for example, (a) the computing services requested by a particular client, (b) the aggregated computing services requested by a particular tenant, and/or (c) the aggregated

computing services requested of the computer network. Such a computer network may be referred to as a “cloud network.”

[0129] In an embodiment, a service provider provides a cloud network to one or more end users. Various service models may be implemented by the cloud network, including but not limited to Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS). In SaaS, a service provider provides end users the capability to use the service provider’s applications, which are executing on the network resources. In PaaS, the service provider provides end users the capability to deploy custom applications onto the network resources. The custom applications may be created using programming languages, libraries, services, and tools supported by the service provider. In IaaS, the service provider provides end users the capability to provision processing, storage, networks, and other fundamental computing resources provided by the network resources. Any arbitrary applications, including an operating system, may be deployed on the network resources.

[0130] In an embodiment, various deployment models may be implemented by a computer network, including but not limited to a private cloud, a public cloud, and a hybrid cloud. In a private cloud, network resources are provisioned for exclusive use by a particular group of one or more entities (the term “entity” as used herein refers to a corporation, organization, person, or other entity). The network resources may be local to and/or remote from the premises of the particular group of entities. In a public cloud, cloud resources are provisioned for multiple entities that are independent from each other (also referred to as “tenants” or “customers”). The computer network and the network resources thereof are accessed by clients corresponding to different tenants. Such a computer network may be referred to as a “multi-tenant computer network.” Several tenants may use a same particular network resource at different times and/or at the same time. The network resources may be local to and/or remote from the premises of the tenants. In a hybrid cloud, a computer network comprises a private cloud and a public cloud. An interface between the private cloud and the public cloud allows for data and application portability. Data stored at the private cloud and data stored at the public cloud may be exchanged through the interface. Applications implemented at the private cloud and applications implemented at the public cloud may have dependencies on each other. A call from an application at the private cloud to an application at the public cloud (and vice versa) may be executed through the interface.

[0131] In an embodiment, tenants of a multi-tenant computer network are independent of each other. For example, a business or operation of one tenant may be separate from a business or operation of another tenant. Different tenants may demand different network requirements for the computer network. Examples of network requirements include processing speed, amount of data storage, security requirements, performance requirements, throughput requirements, latency requirements, resiliency requirements, Quality of Service (QoS) requirements, tenant isolation, and/or consistency. The same computer network may need to implement different network requirements demanded by different tenants.

[0132] In one or more embodiments, in a multi-tenant computer network, tenant isolation is implemented to ensure that the applications and/or data of different tenants are not shared with each other. Various tenant isolation approaches may be used.

[0133] In an embodiment, each tenant is associated with a tenant ID. Each network resource of the multi-tenant computer network is tagged with a tenant ID. A tenant is permitted access to a particular network resource only if the tenant and the particular network resources are associated with a same tenant ID.

[0134] In an embodiment, each tenant is associated with a tenant ID. Each application, implemented by the computer network, is tagged with a tenant ID. Additionally or alternatively, each data structure and/or dataset, stored by the computer network, is tagged with a tenant ID. A tenant is permitted access to a particular application, data structure, and/or dataset only if the tenant and the particular application, data structure, and/or dataset are associated with a same tenant ID.

[0135] As an example, each database implemented by a multi-tenant computer network may be tagged with a tenant ID. Only a tenant associated with the corresponding tenant ID may access data of a particular database. As another example, each entry in a database implemented by a multi-tenant computer network may be tagged with a tenant ID. Only a tenant associated with the corresponding tenant ID may access data of a particular entry. However, the database may be shared by multiple tenants.

[0136] In an embodiment, a subscription list indicates which tenants have authorization to access which applications. For each application, a list of tenant IDs of tenants authorized to access the application is stored. A tenant is permitted access to a particular application only if the

tenant ID of the tenant is included in the subscription list corresponding to the particular application.

[0137] In an embodiment, network resources (such as digital devices, virtual machines, application instances, and threads) corresponding to different tenants are isolated to tenant-specific overlay networks maintained by the multi-tenant computer network. As an example, packets from any source device in a tenant overlay network may only be transmitted to other devices within the same tenant overlay network. Encapsulation tunnels are used to prohibit any transmissions from a source device on a tenant overlay network to devices in other tenant overlay networks. Specifically, the packets received from the source device are encapsulated within an outer packet. The outer packet is transmitted from a first encapsulation tunnel endpoint (in communication with the source device in the tenant overlay network) to a second encapsulation tunnel endpoint (in communication with the destination device in the tenant overlay network). The second encapsulation tunnel endpoint decapsulates the outer packet to obtain the original packet transmitted by the source device. The original packet is transmitted from the second encapsulation tunnel endpoint to the destination device in the same particular overlay network.

[0138] 5. MICROSERVICE APPLICATIONS

[0139] According to one or more embodiments, the techniques described herein are implemented in a microservice architecture. A microservice in this context refers to software logic designed to be independently deployable, having endpoints that may be logically coupled to other microservices to build a variety of applications. Applications built using microservices are distinct from monolithic applications, which are designed as a single fixed unit and generally comprise a single logical executable. With microservice applications, different microservices are independently deployable as separate executables. Microservices may communicate using HyperText Transfer Protocol (HTTP) messages and/or according to other communication protocols via API endpoints. Microservices may be managed and updated separately, written in different languages, and be executed independently from other microservices.

[0140] Microservices provide flexibility in managing and building applications. Different applications may be built by connecting different sets of microservices without changing the source code of the microservices. Thus, the microservices act as logical building blocks that may be arranged in a variety of ways to build different applications. Microservices may provide

monitoring services that notify a microservices manager (such as If-This-Then-That (IFTTT), Zapier, or Oracle Self-Service Automation (OSSA)) when trigger events from a set of trigger events exposed to the microservices manager occur. Microservices exposed for an application may alternatively or additionally provide action services that perform an action in the application (controllable and configurable via the microservices manager by passing in values, connecting the actions to other triggers and/or data passed along from other actions in the microservices manager) based on data received from the microservices manager. The microservice triggers and/or actions may be chained together to form recipes of actions that occur in optionally different applications that are otherwise unaware of or have no control or dependency on each other. These managed applications may be authenticated or plugged in to the microservices manager, for example, with user-supplied application credentials to the manager, without requiring reauthentication each time the managed application is used alone or in combination with other applications.

[0141] In one or more embodiments, microservices may be connected via a GUI. For example, microservices may be displayed as logical blocks within a window, frame, other element of a GUI. A user may drag and drop microservices into an area of the GUI used to build an application. The user may connect the output of one microservice into the input of another microservice using directed arrows or any other GUI element. The application builder may run verification tests to confirm that the output and inputs are compatible (e.g., by checking the datatypes, size restrictions, etc.)

[0142] TRIGGERS

[0143] The techniques described above may be encapsulated into a microservice, according to one or more embodiments. In other words, a microservice may trigger a notification (into the microservices manager for optional use by other plugged in applications, herein referred to as the “target” microservice) based on the above techniques and/or may be represented as a GUI block and connected to one or more other microservices. The trigger condition may include absolute or relative thresholds for values, and/or absolute or relative thresholds for the amount or duration of data to analyze, such that the trigger to the microservices manager occurs whenever a plugged-in microservice application detects that a threshold is crossed. For example, a user may request a trigger into the microservices manager when the microservice application detects a value has crossed a triggering threshold.

[0144] In one embodiment, the trigger, when satisfied, might output data for consumption by the target microservice. In another embodiment, the trigger, when satisfied, outputs a binary value indicating the trigger has been satisfied, or outputs the name of the field or other context information for which the trigger condition was satisfied. Additionally or alternatively, the target microservice may be connected to one or more other microservices such that an alert is input to the other microservices. Other microservices may perform responsive actions based on the above techniques, including, but not limited to, deploying additional resources, adjusting system configurations, and/or generating GUIs.

[0145] ACTIONS

[0146] In one or more embodiments, a plugged-in microservice application may expose actions to the microservices manager. The exposed actions may receive, as input, data or an identification of a data object or location of data, that causes data to be moved into a data cloud.

[0147] In one or more embodiments, the exposed actions may receive, as input, a request to increase or decrease existing alert thresholds. The input might identify existing in-application alert thresholds and whether to increase or decrease, or delete the threshold. Additionally, or alternatively, the input might request the microservice application to create new in-application alert thresholds. The in-application alerts may trigger alerts to the user while logged into the application, or may trigger alerts to the user using default or user-selected alert mechanisms available within the microservice application itself, rather than through other applications plugged into the microservices manager.

[0148] In one or more embodiments, the microservice application may generate and provide an output based on input that identifies, locates, or provides historical data, and defines the extent or scope of the requested output. The action, when triggered, causes the microservice application to provide, store, or display the output, for example, as a data model or as aggregate data that describes a data model.

[0149] 6. HARDWARE OVERVIEW

[0150] According to one embodiment, the techniques described herein are implemented by one or more special-purpose computing devices. The special-purpose computing devices may be hard-wired to perform the techniques, or may include digital electronic devices such as one or more application-specific integrated circuits (ASICs), field programmable gate arrays (FPGAs), or network processing units (NPU) that are persistently programmed to perform the techniques,

or may include one or more general purpose hardware processors programmed to perform the techniques pursuant to program instructions in firmware, memory, other storage, or a combination. Such special-purpose computing devices may also combine custom hard-wired logic, ASICs, FPGAs, or NPUs with custom programming to accomplish the techniques. The special-purpose computing devices may be desktop computer systems, portable computer systems, handheld devices, networking devices or any other device that incorporates hard-wired and/or program logic to implement the techniques.

[0151] For example, FIG. 3 is a block diagram that illustrates a computer system 300 upon which an embodiment of the invention may be implemented. Computer system 300 includes a bus 302 or other communication mechanism for communicating information, and a hardware processor 304 coupled with bus 302 for processing information. Hardware processor 304 may be, for example, a general-purpose microprocessor.

[0152] Computer system 300 also includes a main memory 306, such as a random-access memory (RAM) or other dynamic storage device, coupled to bus 302 for storing information and instructions to be executed by processor 304. Main memory 306 also may be used for storing temporary variables or other intermediate information during execution of instructions to be executed by processor 304. Such instructions, when stored in non-transitory storage media accessible to processor 304, render computer system 300 into a special-purpose machine that is customized to perform the operations specified in the instructions.

[0153] Computer system 300 further includes a read only memory (ROM) 308 or other static storage device coupled to bus 302 for storing static information and instructions for processor 304. A storage device 310, such as a magnetic disk or optical disk, is provided and coupled to bus 302 for storing information and instructions.

[0154] Computer system 300 may be coupled via bus 302 to a display 312, such as a cathode ray tube (CRT), for displaying information to a computer user. An input device 314, including alphanumeric and other keys, is coupled to bus 302 for communicating information and command selections to processor 304. Another type of user input device is cursor control 316, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 304 and for controlling cursor movement on display 312. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), that allows the device to specify positions in a plane.

[0155] Computer system 300 may implement the techniques described herein using customized hard-wired logic, one or more ASICs or FPGAs, firmware and/or program logic which in combination with the computer system causes or programs computer system 300 to be a special-purpose machine. According to one embodiment, the techniques herein are performed by computer system 300 in response to processor 304 executing one or more sequences of one or more instructions contained in main memory 306. Such instructions may be read into main memory 306 from another storage medium, such as storage device 310. Execution of the sequences of instructions contained in main memory 306 causes processor 304 to perform the process steps described herein. In alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions.

[0156] The term “storage media” as used herein refers to any non-transitory media that store data and/or instructions that cause a machine to operate in a specific fashion. Such storage media may comprise non-volatile media and/or volatile media. Non-volatile media includes, for example, optical or magnetic disks, such as storage device 310. Volatile media includes dynamic memory, such as main memory 306. Common forms of storage media include, for example, a floppy disk, a flexible disk, hard disk, solid state drive, magnetic tape, or any other magnetic data storage medium, a CD-ROM, any other optical data storage medium, any physical medium with patterns of holes, a RAM, a PROM, and EPROM, a FLASH-EPROM, NVRAM, any other memory chip or cartridge, content-addressable memory (CAM), and ternary content-addressable memory (TCAM).

[0157] Storage media is distinct from but may be used in conjunction with transmission media. Transmission media participates in transferring information between storage media. For example, transmission media includes coaxial cables, copper wire and fiber optics, including the wires that comprise bus 302. Transmission media can also take the form of acoustic or light waves, such as those generated during radio-wave and infra-red data communications.

[0158] Various forms of media may be involved in carrying one or more sequences of one or more instructions to processor 304 for execution. For example, the instructions may initially be carried on a magnetic disk or solid-state drive of a remote computer. The remote computer can load the instructions into its dynamic memory and send the instructions over a telephone line using a modem. A modem local to computer system 300 can receive the data on the telephone line and use an infra-red transmitter to convert the data to an infra-red signal. An infra-red

detector can receive the data carried in the infra-red signal and appropriate circuitry can place the data on bus 302. Bus 302 carries the data to main memory 306, from which processor 304 retrieves and executes the instructions. The instructions received by main memory 306 may optionally be stored on storage device 310 either before or after execution by processor 304.

[0159] Computer system 300 also includes a communication interface 318 coupled to bus 302. Communication interface 318 provides a two-way data communication coupling to a network link 320 that is connected to a local network 322. For example, communication interface 318 may be an integrated services digital network (ISDN) card, cable modem, satellite modem, or a modem to provide a data communication connection to a corresponding type of telephone line. As another example, communication interface 318 may be a local area network (LAN) card to provide a data communication connection to a compatible LAN. Wireless links may also be implemented. In any such implementation, communication interface 318 sends and receives electrical, electromagnetic, or optical signals that carry digital data streams representing various types of information.

[0160] Network link 320 typically provides data communication through one or more networks to other data devices. For example, network link 320 may provide a connection through local network 322 to a host computer 324 or to data equipment operated by an Internet Service Provider (ISP) 326. ISP 326 in turn provides data communication services through the world-wide packet data communication network now commonly referred to as the “Internet” 328. Local network 322 and Internet 328 both use electrical, electromagnetic, or optical signals that carry digital data streams. The signals through the various networks and the signals on network link 320 and through communication interface 318, which carry the digital data to and from computer system 300, are example forms of transmission media.

[0161] Computer system 300 can send messages and receive data, including program code, through the network(s), network link 320 and communication interface 318. In the Internet example, a server 330 might transmit a requested code for an application program through Internet 328, ISP 326, local network 322 and communication interface 318.

[0162] The received code may be executed by processor 304 as it is received, and/or stored in storage device 310, or other non-volatile storage for later execution.

[0163] 7. MISCELLANEOUS; EXTENSIONS

[0164] Embodiments are directed to a system with one or more devices that include a hardware processor and that are configured to perform any of the operations described herein and/or recited in any of the claims below. In an embodiment, a non-transitory computer readable storage medium comprises instructions which, when executed by one or more hardware processors, causes performance of any of the operations described herein and/or recited in any of the claims.

[0165] Any combination of the features and functionalities described herein may be used in accordance with one or more embodiments. In the foregoing specification, embodiments have been described with reference to numerous specific details that may vary from implementation to implementation. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense. The sole and exclusive indicator of the scope of the invention, and what is intended by the applicants to be the scope of the invention, is the literal and equivalent scope of the set of claims that issue from this application, in the specific form in which such claims issue, including any subsequent correction.

CLAIMS

What is claimed is:

1. One or more non-transitory computer-readable media storing instructions, which when executed by one or more hardware processors, cause performance of operations comprising:
receiving, by a server from a browser executing on a browser device, (a) a first hash value corresponding to a browser-side DOM tree stored by the browser, and (b) a server-side update instruction for applying a first server-side update to a server-side DOM tree stored by the server, wherein the first server-side update to the server-side DOM tree reflects a first browser-side update by the browser to the browser-side DOM tree;
identifying, by the server, based on the first hash value, the server-side DOM tree;
executing, by the server, responsive to the server-side update instruction, the first server-side update to the server-side DOM tree, wherein the first server-side update comprises a first set of one or more server-side operations on the server-side DOM tree; and
computing, by the server, a second hash value representing the server-side DOM tree resulting from the first server-side update; and
storing, by the server, the second hash value in association with the server-side DOM tree resulting from the first server-side update.
2. The media of Claim 1, wherein the first server-side update represents an indication of a user interaction with the browser.
3. The media of claim 1, wherein the operations further comprise:
subsequent to the server executing the first server-side update to the server-side DOM tree,
numbering a first set of server-side nodes of the server-side DOM tree according to a node numbering convention;
wherein a second set of browser-side nodes of the browser-side DOM tree are numbered according to the node numbering convention; and
wherein each of the first set of server-side nodes has a server-side number that matches a browser-side number of a corresponding browser-side node.

4. One or more non-transitory computer-readable media storing instructions, which when executed by one or more hardware processors, cause performance of operations comprising: receiving, by a server from a browser executing on a browser device, (a) a first hash value corresponding to a browser-side DOM tree stored by the browser, and (b) a server-side update instruction for applying a first server-side update to a server-side DOM tree stored by the server, wherein the first server-side update to the server-side DOM tree reflects a first browser-side update by the browser to the browser-side DOM tree; identifying, by the server, based on the first hash value, the server-side DOM tree; executing, by the server, responsive to the server-side update instruction, (a) the first server-side update to the server-side DOM tree, wherein the first server-side update comprises a first set of one or more server-side operations on the server-side DOM tree, and (b) a second server-side update to the server-side DOM tree, wherein the second server-side update comprises a second set of one or more server-side operations on the server-side DOM tree, wherein the second server-side update is triggered by the first server-side update; computing, by the server, a browser-side update instruction for applying a second browser-side update to the browser-side DOM tree, wherein the second browser-side update comprises a second set of one or more browser-side operations on the browser-side DOM tree, wherein the second browser-side update to the browser-side DOM tree reflects the second server-side update to the server-side DOM tree; and transmitting, by the server to the browser executing on the browser device, the browser-side update instruction, wherein, responsive to the browser-side update instruction, the browser executes the second browser-side update to the browser-side DOM tree.
5. The media of claim 4, wherein computing the browser-side update instruction for applying the second browser-side update to the browser-side DOM tree comprises: generating a first server-side hash tree representing the server-side DOM tree resulting from the first server-side update, wherein the first server-side hash tree comprises a first set of leaf nodes; generating a second server-side hash tree representing the server-side DOM tree resulting from the second server-side update, wherein the second server-side hash tree comprises a second set of leaf nodes;

determining a first difference between a first leaf node, of the first set of leaf nodes, and a second leaf node, of the second set of leaf nodes,
wherein the first leaf node occupies a first leaf position of the first server-side hash tree,
wherein the second leaf node occupies a second leaf position of the second server-side hash tree, and
wherein the first leaf position corresponds to the second leaf position;
based on the first difference between the first leaf node and the second leaf node:
identifying, in the server-side DOM tree resulting from the first server-side update, a first server-side DOM tree-node corresponding to the first leaf node of the first server-side hash tree,
identifying, in the server-side DOM tree resulting from the second server-side update, a second server-side DOM tree-node corresponding to the second leaf node of the second server-side hash tree,
determining a second difference between the first server-side DOM tree-node and the second server-side DOM tree-node,
determining the second browser-side update to the browser-side DOM tree based on the second difference between the first server-side DOM tree-node and the second server-side DOM tree-node,
wherein the browser-side update instruction comprises a first instruction configured to cause the browser to apply the second browser-side update;
wherein upon the browser having applied the second browser-side update to the browser-side DOM tree, a first browser-side DOM tree-node, corresponding to the first server-side DOM tree-node, is modified to match the second server-side DOM tree-node.

6. The media of claim 4, wherein the operations further comprise:
computing, by the server, a second hash value representing the server-side DOM tree resulting from the first server-side update; and
transmitting, by the server to the browser executing on the browser device, the second hash value, wherein, the browser identifies the browser-side DOM tree based on the second hash value.

7. The media of claim 6, wherein computing the second hash value comprises:
generating a server-side hash tree representing the server-side DOM tree resulting from the first server-side update;
determining a server-side root hash value of the server-side hash tree; and
selecting the server-side root hash value as the second hash value.

8. The media of claim 7, wherein the server-side hash tree comprises a plurality of server-side leaf nodes, and wherein the server-side DOM tree comprises a plurality of server-side DOM tree-nodes, and wherein each of the plurality of server-side leaf nodes comprises a server-side leaf hash value that represents a corresponding server-side DOM tree-node of the plurality of server-side DOM tree-nodes.

9. The media of claim 7,
wherein the browser computes a third hash value, representing the browser-side DOM tree resulting from the first browser-side update, at least by:
generating a browser-side hash tree representing the browser-side DOM tree resulting from the first browser-side update;
determining a browser-side root hash value of the browser-side hash tree; and
selecting the browser-side root hash value as the third hash value;
wherein the second hash value matches the third hash value; and
wherein the browser identifies the browser-side DOM tree based on the second hash value matching the third hash value.

10. The media of claim 9,
wherein the server-side hash tree comprises a plurality of server-side hash tree-nodes, wherein each of the plurality of server-side hash tree-nodes comprises a server-side node-value;
and
wherein the operations further comprise:
assigning a server-side node identification number to each of the plurality of server-side hash tree-nodes of the server-side hash tree at least by numbering the plurality of

server-side hash tree-nodes of the server-side hash tree according to a numbering convention,

wherein for at least a first server-side hash tree-node of the plurality of server-side hash tree-nodes, a first server-side node identification number assigned to the first server-side hash tree-node replaces a second server-side hash tree-node identification number having been previously assigned to the first server-side hash tree-node;

wherein the browser-side hash tree comprises a plurality of browser-side hash tree-nodes,

wherein each of the plurality of browser-side hash tree-nodes comprises a browser-side node identification number having been assigned according to the numbering convention, and a browser-side node-value;

wherein the server-side node-value of each server-side hash tree-node identifiable by a corresponding server-side node identification number, matches the browser-side node-value of a corresponding browser-side hash tree-node identifiable by a corresponding browser-side node identification number.

11. The media of claim 4,

wherein the first server-side update to the server-side DOM tree represents a content request from the browser;

wherein the second server-side update to the server-side DOM tree represents a response to the content request as applied to the server-side DOM tree, wherein the response as applied to the server-side DOM tree comprises a change with respect to at least one item of content as represented by the server-side DOM tree; and

wherein the second browser-side update to the browser-side DOM tree represents the response to the content request as applied to the browser-side DOM tree, wherein the response as applied to the browser-side DOM tree comprises the change with respect to the at least one item of content as represented by the browser-side DOM tree.

12. The media of claim 4, wherein the first hash value matches a second hash value corresponding to the server-side DOM tree, the second hash value computed by the server prior to receiving the first hash value and the server-side update instruction.

13. The media of claim 4, wherein the first set of one or more server-side operations of the first server-side update comprises:

a first node identification number corresponding to a first server-side node of the server-side DOM tree; and

a first node operation to be executed upon the first server-side node of the server-side DOM tree;

wherein the first node identification number corresponds to a node numbering convention

applicable to (i) a first set of server-side nodes, including the first server-side node, represented by the server-side DOM tree, and (ii) a second set of browser-side nodes represented by the browser-side DOM tree.

14. The media of claim 4, wherein the operations further comprise:

subsequent to the server executing the first server-side update to the server-side DOM tree,

numbering a first set of server-side nodes of the server-side DOM tree according to a node numbering convention;

wherein a second set of browser-side nodes of the browser-side DOM tree are numbered

according to the node numbering convention;

wherein each of the first set of server-side nodes has a server-side number that matches a

browser-side number of a corresponding browser-side node; and

wherein the second set of one or more browser-side operations of the second browser-side update comprises:

a first node identification number corresponding to a first browser-side node of the browser-side DOM tree; and

a first node operation to be executed upon the first browser-side node of the browser-side DOM tree.

15. The media of claim 4, wherein computing the browser-side update instruction for

applying the second browser-side update to the browser-side DOM tree comprises:

generating a first server-side hash tree representing the server-side DOM tree resulting from the first server-side update, wherein the first server-side hash tree comprises:

a first server-side root node having a first root node-value,

a first set of server-side leaf nodes, and
at least a first server-side intermediate node and a second server-side intermediate node
respectively extending from the first server-side root node, the first server-side
intermediate node having a first intermediate node-value, and the second server-
side intermediate node having a second intermediate node-value,
wherein at least a first server-side leaf node and a second server-side leaf node of
the first set of server-side leaf nodes extend from the second server-side
intermediate node, the first server-side leaf node having a first leaf node-
value, and the second server-side leaf node having a second leaf node-
value;
generating a second server-side hash tree representing the server-side DOM tree resulting from
the second server-side update, wherein the second server-side hash tree comprises:
a second server-side root node having a second root node-value,
a second set of server-side leaf nodes, and
at least a third server-side intermediate node and a fourth server-side intermediate node
respectively extending from the second server-side root node, the third server-side
intermediate node having a third intermediate node-value, and the fourth server-
side intermediate node having a fourth intermediate node-value,
wherein at least a third server-side leaf node and a fourth server-side leaf node of
the second set of server-side leaf nodes extend from the fourth server-side
intermediate node, the third server-side leaf node having a third leaf node-
value, and the fourth server-side leaf node having a fourth leaf node-value;
comparing the first root node-value of the first server-side root node to the second root node-
value of the second server-side root node, and determining that the first root node-value
differs from the second root node-value;
based on the first root node-value differing from the second root node-value, comparing the first
intermediate node-value of the first server-side intermediate node to the third
intermediate node-value of the third server-side intermediate node, and determining that
the first intermediate node-value matches the third intermediate node-value,
wherein the first server-side intermediate node occupies a first intermediate position of
the first server-side hash tree,

wherein the third server-side intermediate node occupies a third intermediate position of the second server-side hash tree, and

wherein the first intermediate position corresponds to the third intermediate position;

based on the first intermediate node-value matching the third intermediate node-value,

comparing the second intermediate node-value of the first server-side intermediate node to the fourth intermediate node-value of the fourth server-side intermediate node, and

determining that the second intermediate node-value differs from the fourth intermediate node-value,

wherein the second server-side intermediate node occupies a second intermediate position of the first server-side hash tree,

wherein the fourth server-side intermediate node occupies a fourth intermediate position of the second server-side hash tree, and

wherein the second intermediate position corresponds to the fourth intermediate position;

based on the second intermediate node-value differing from the fourth intermediate node-value,

comparing the first leaf node-value of the first server-side leaf node to the third leaf node-value of the third server-side leaf node, and determining that the first leaf node-value matches the third leaf node-value,

wherein the first server-side leaf node occupies a first leaf position of the first server-side hash tree,

wherein the third server-side leaf node occupies a third leaf position of the second server-side hash tree, and

wherein the first leaf position corresponds to the third leaf position;

based on the first leaf node-value matching the third leaf node-value, comparing the second leaf node-value of the second server-side leaf node to the fourth leaf node-value of the fourth server-side leaf node, and determining that the second leaf node-value differs from the fourth leaf node-value,

wherein the second server-side leaf node occupies a second leaf position of the first server-side hash tree,

wherein the fourth server-side leaf node occupies a fourth leaf position of the second server-side hash tree, and

wherein the second leaf position corresponds to the fourth leaf position;

based on the second leaf node-value differing from the fourth leaf node-value:

identifying, in the server-side DOM tree resulting from the first server-side update, a first server-side DOM tree-node corresponding to the second server-side leaf node of the first server-side hash tree,

identifying, in the server-side DOM tree resulting from the second server-side update, a second server-side DOM tree-node corresponding to the fourth server-side leaf node of the second server-side hash tree,

determining a difference between the first server-side DOM tree-node and the second server-side DOM tree-node,

determining the second browser-side update to the browser-side DOM tree based on the difference between the first server-side DOM tree-node and the second server-side DOM tree-node,

wherein the browser-side update instruction comprises a first instruction configured to cause the browser to apply the second browser-side update;

wherein upon the browser having applied the second browser-side update to the browser-side DOM tree, a first browser-side DOM tree-node, corresponding to the first server-side DOM tree-node, is modified to match the second server-side DOM tree-node.

16. A method, comprising operations as recited in any of Claims 1-15.

17. A system, comprising:

at least one hardware processor;

the system being configured to use the at least one hardware processor to execute operations as recited in any of Claims 1-15.

18. A computer program product comprising instructions which, when executed by one or more hardware processors, causes performance of operations as recited in any of Claims 1-15.

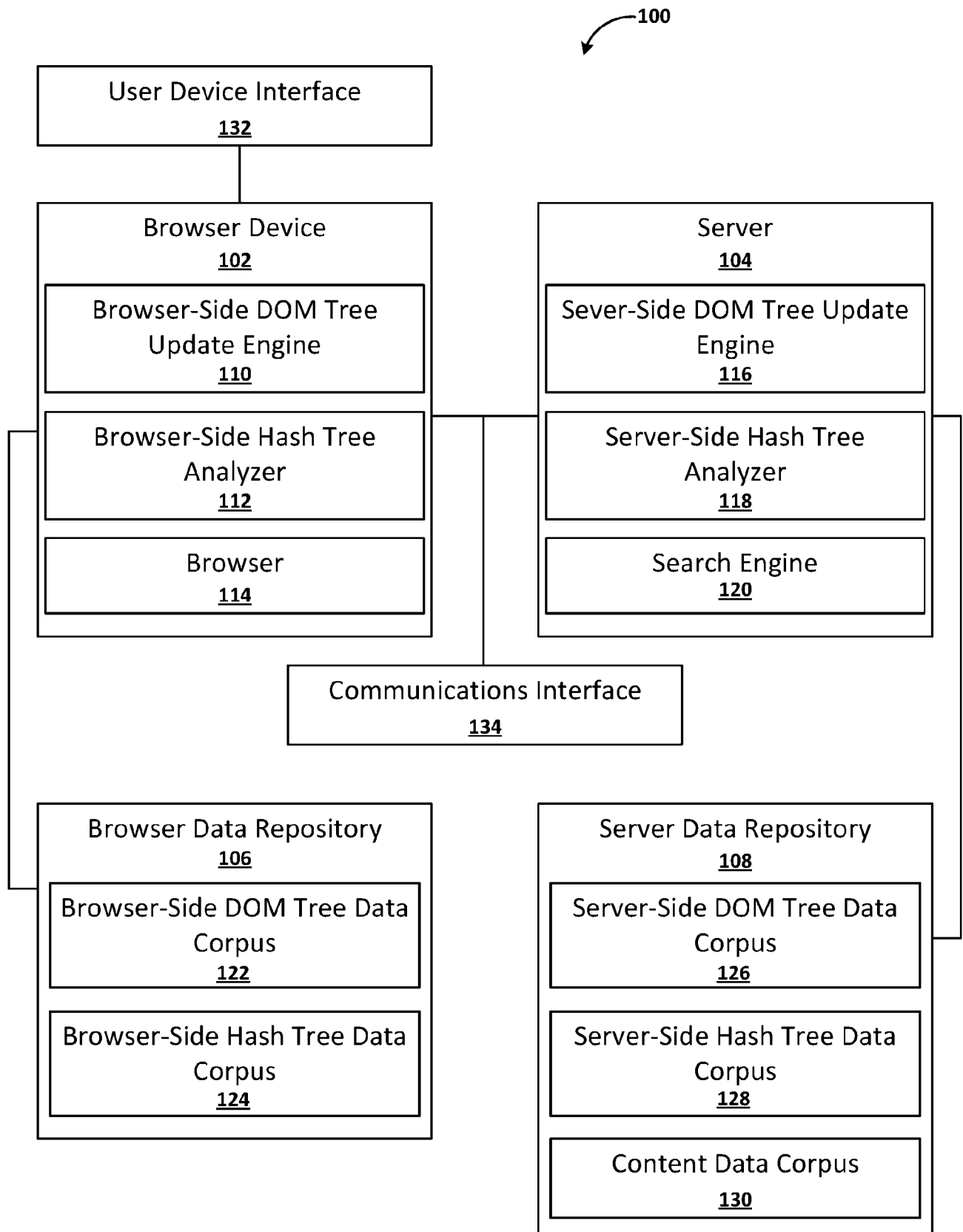


FIG. 1

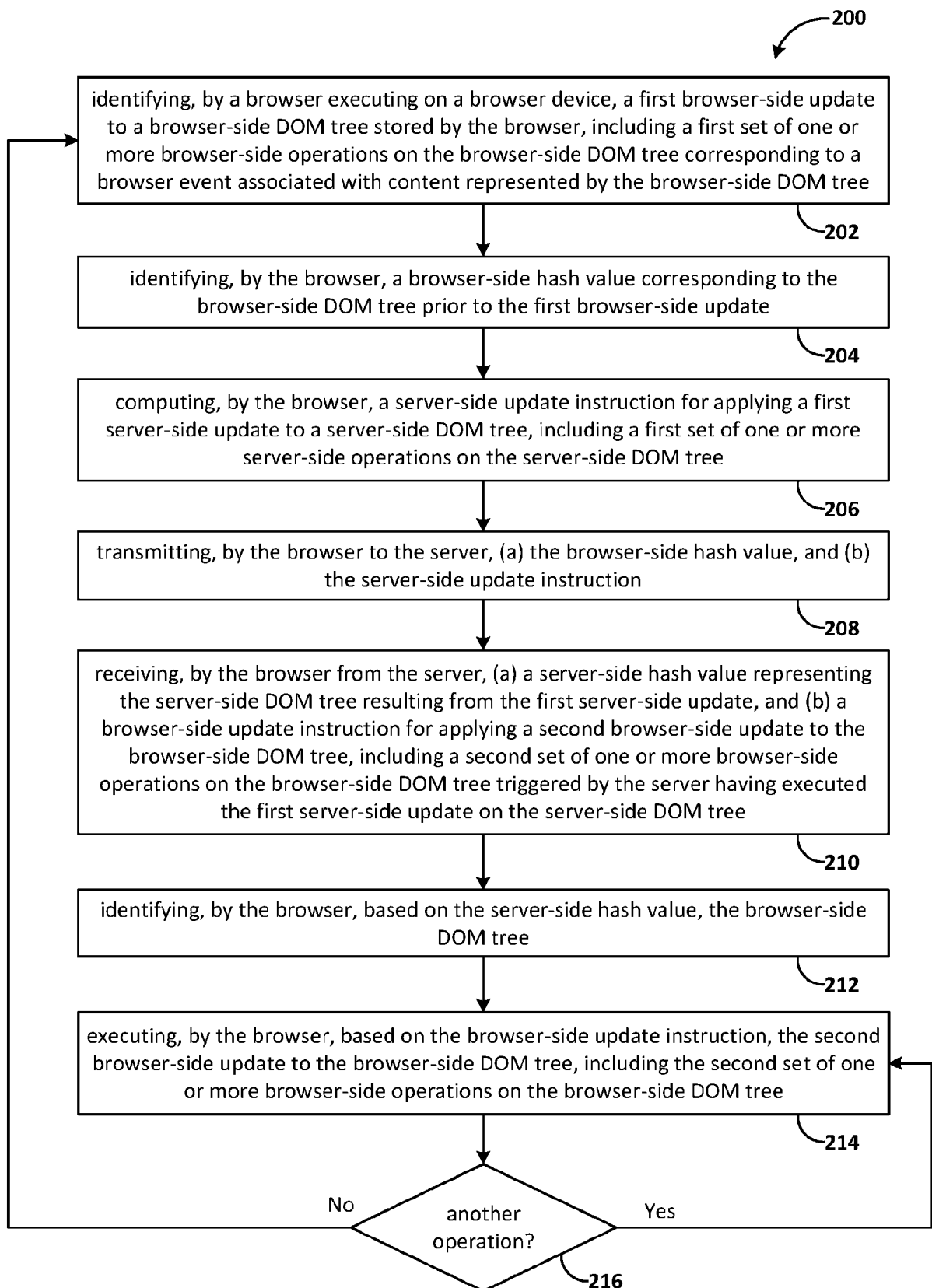


FIG. 2A

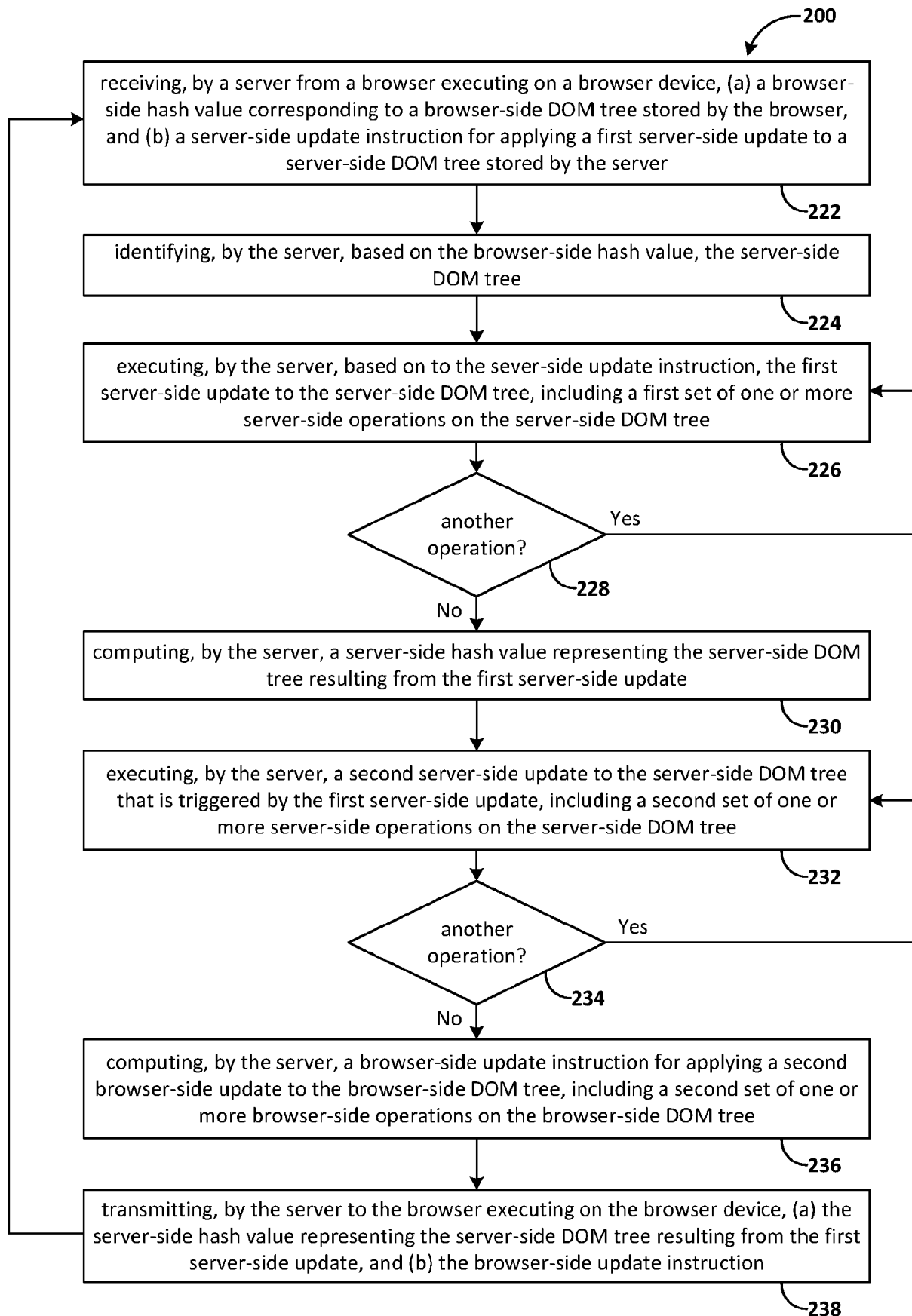


FIG. 2B

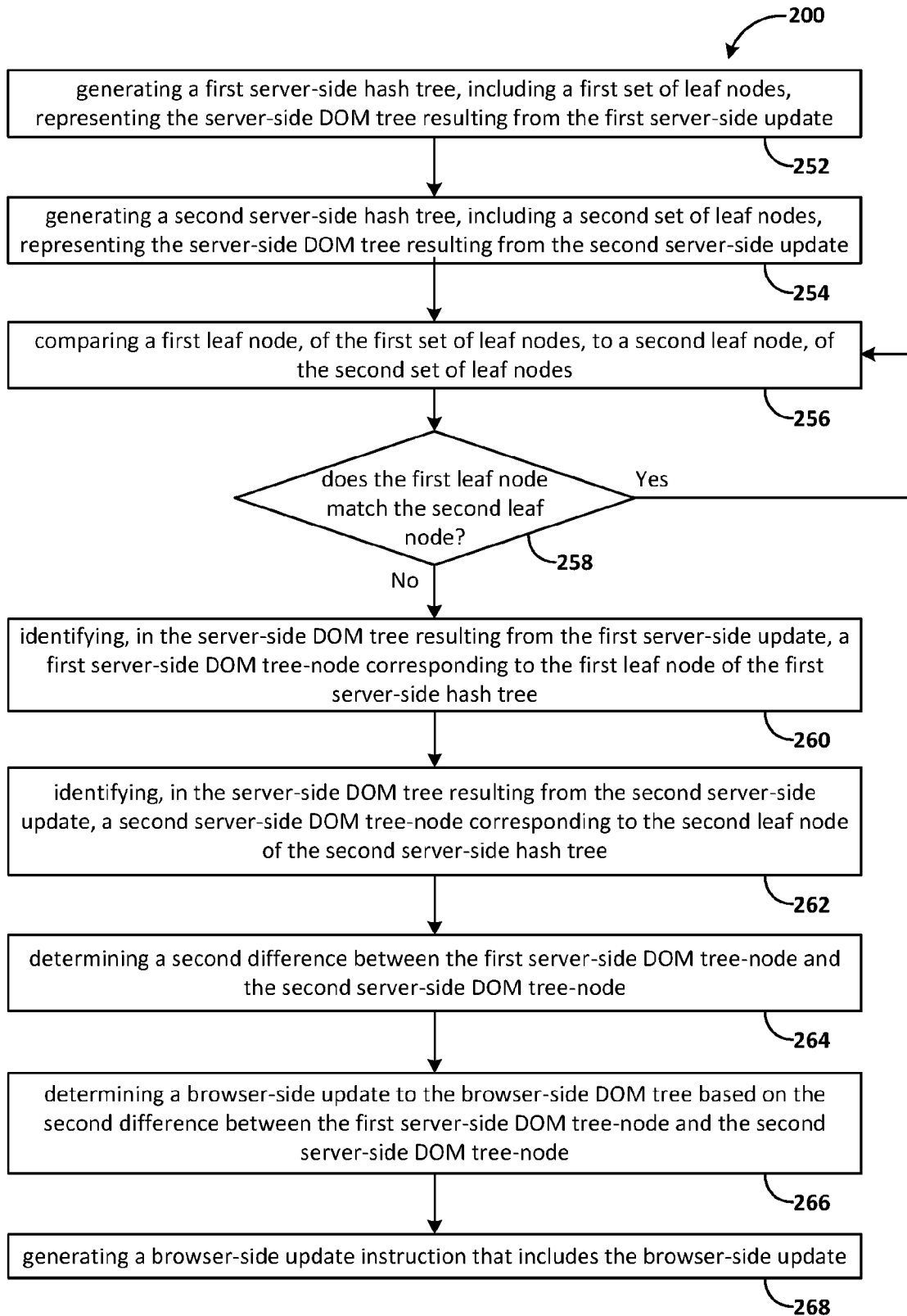


FIG. 2C

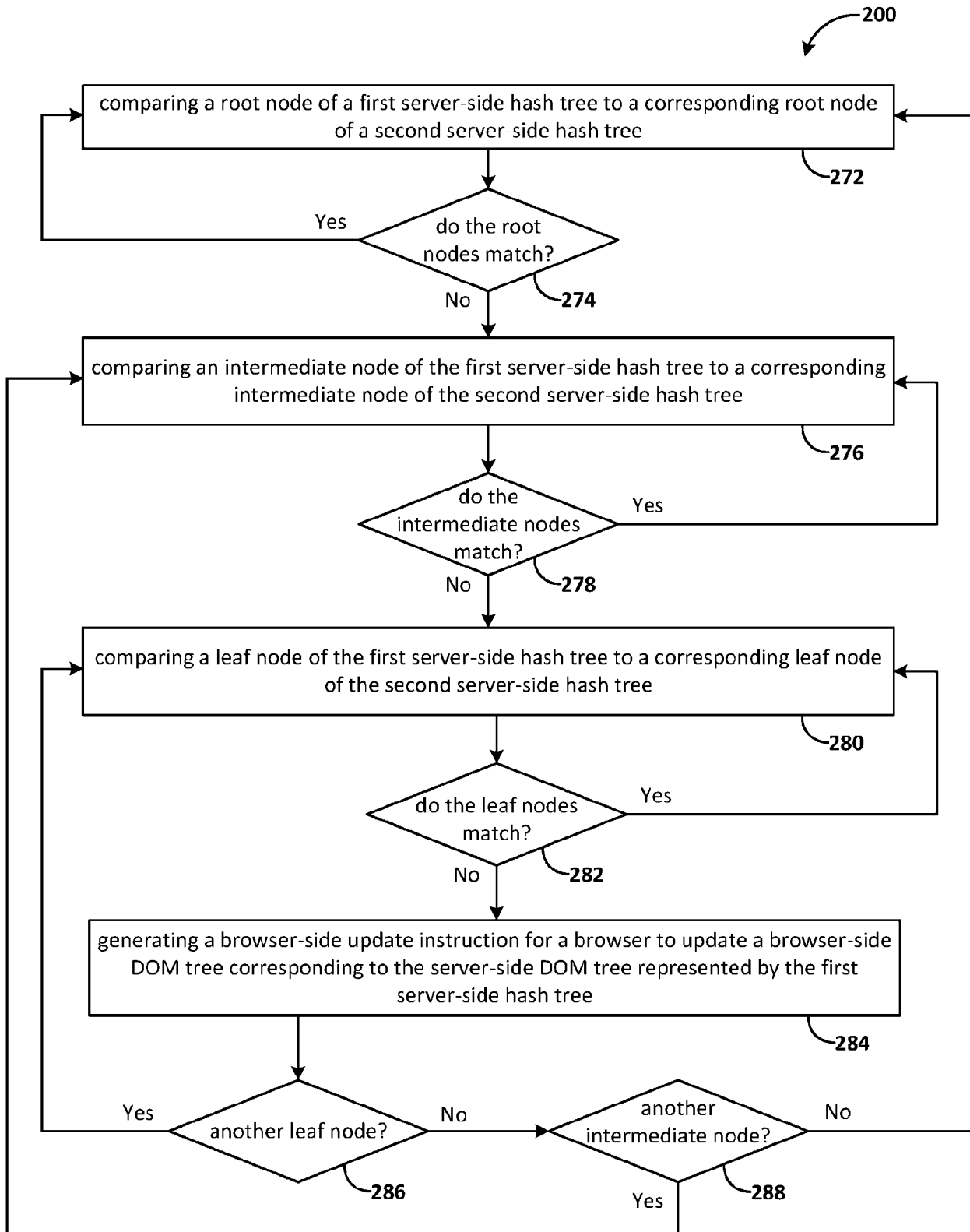


FIG. 2D

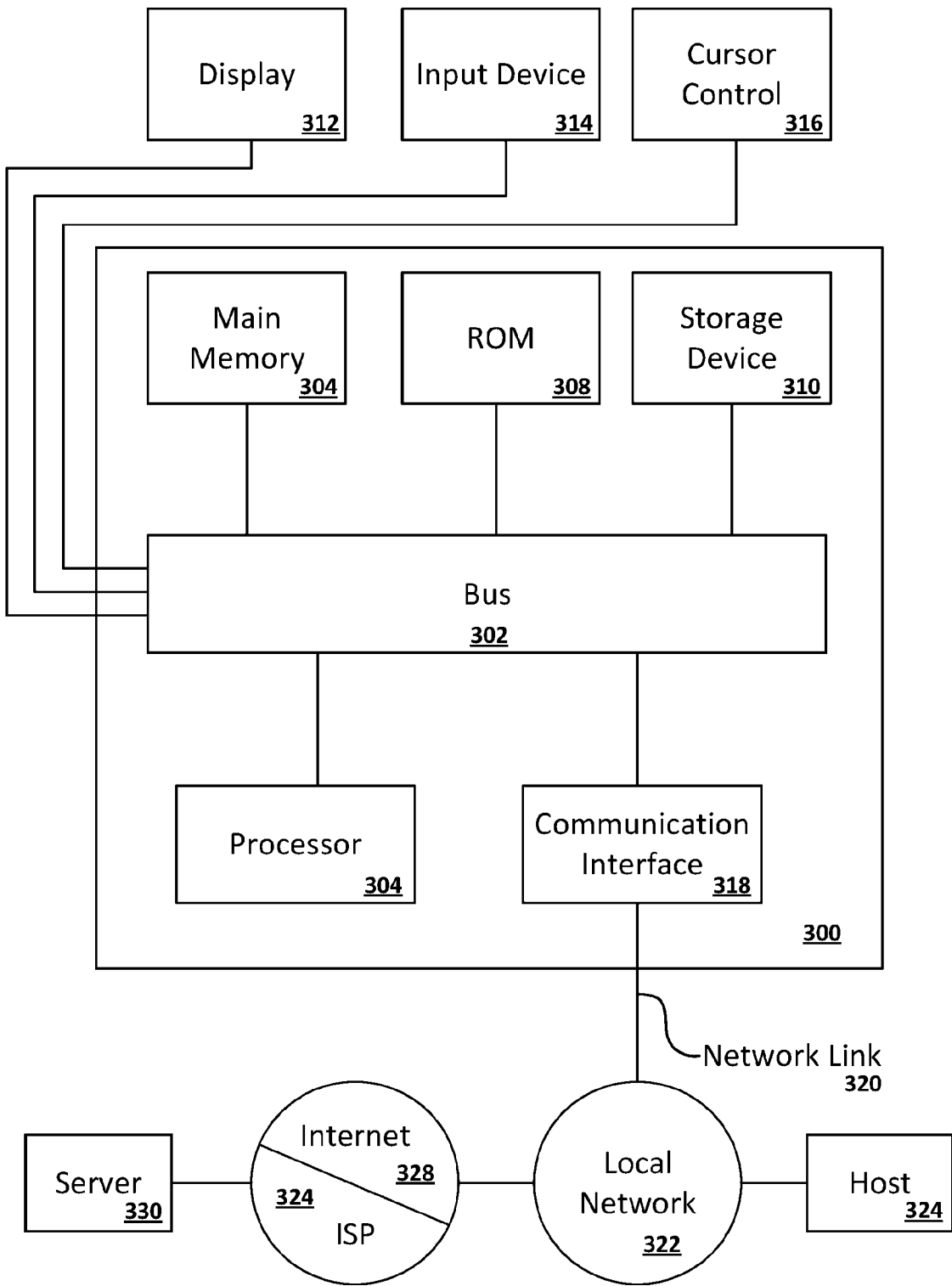


FIG. 3

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2024/037407

A. CLASSIFICATION OF SUBJECT MATTER

INV. G06F16/27 G06F16/901 G06F16/957
ADD.

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

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2012/260157 A1 (ZHU BIN BENJAMIN [US] ET AL) 11 October 2012 (2012-10-11) abstract paragraph [0057] - paragraph [0082] figures 1-6 ----- -/-	1-18



Further documents are listed in the continuation of Box C.



See patent family annex.

* 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

"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

3 October 2024

Date of mailing of the international search report

22/10/2024

Name and mailing address of the ISA/

European Patent Office, P.B. 5818 Patentlaan 2
NL - 2280 HV Rijswijk
Tel. (+31-70) 340-2040,
Fax: (+31-70) 340-3016

Authorized officer

Dumitrescu, Cristina

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2024/037407

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT		
Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	HIROSI MARUYAMA KENT TAMURA NAOHIKO URAMOTO IBM: "Digest Values for DOM (DOMHASH) ----- Hiroshi Maruyama Kent Tamura Naohiko Uramoto; draft-ietf-trade-hiroshi-dom-hash-03.txt", DIGEST VALUES FOR DOM (DOMHASH) ----- ----- HIROSHI MARUYAMA KENT TAMURA NAOHIKO URAMOTO; DRAFT-IETF-TRADE-HIROSHI-DOM-HASH-03.TXT, INTERNET ENGINEERING TASK FORCE, IETF; STANDARDWORKINGDRAFT, INTERNET SOCIETY (ISOC) 4, RUE DES FA, vol. trade, no. 3, 1 September 1999 (1999-09-01), XP015029259, the whole document -----	1-18

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No

PCT/US2024/037407

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2012260157	A1	11-10-2012	NONE
