US 20140173397A1

(19) United States

(12) Patent Application Publication

(10) Pub. No.: US 2014/0173397 A1

(43) Pub. Date: Jun. 19, 2014

Pereira et al.

(54) AUTOMATED DOCUMENT COMPOSITION USING CLUSTERS

(76) Inventors: **Jose Bento Ayres Pereira**, Palo Alto, CA (US); **Keyen Liu**, Beijing (CN); **Lei Wang**, Beijing (CN); **Niranjan Damera-Venkata**, Chennai, Tamil Nadu (IN)

(21) Appl. No.: **14/234,154**

(22) PCT Filed: **Jul. 22, 2011**

(86) PCT No.: **PCT/CN2011/001203**

§ 371 (c)(1), (2), (4) Date: **Jan. 22, 2014**

Publication Classification

(51) **Int. Cl.**
G06F 17/24 (2006.01)

(52) **U.S. Cl.**
CPC ***G06F 17/248*** (2013.01)
USPC **715/202**

(57) **ABSTRACT**

Systems and methods of automated document composition using clusters are disclosed. In an example, a method comprises determining a plurality of composition scores $\Phi_A(A, B)$, the composition scores each computing separately on a plurality of worker nodes in the cluster. The method also includes determining coefficients ($\tau_i(A)$ at a master node in the cluster based on the composition scores (Φ_i) from each of the worker nodes. The method also includes outputting an optimal document (D*) using the coefficients (τ_i).

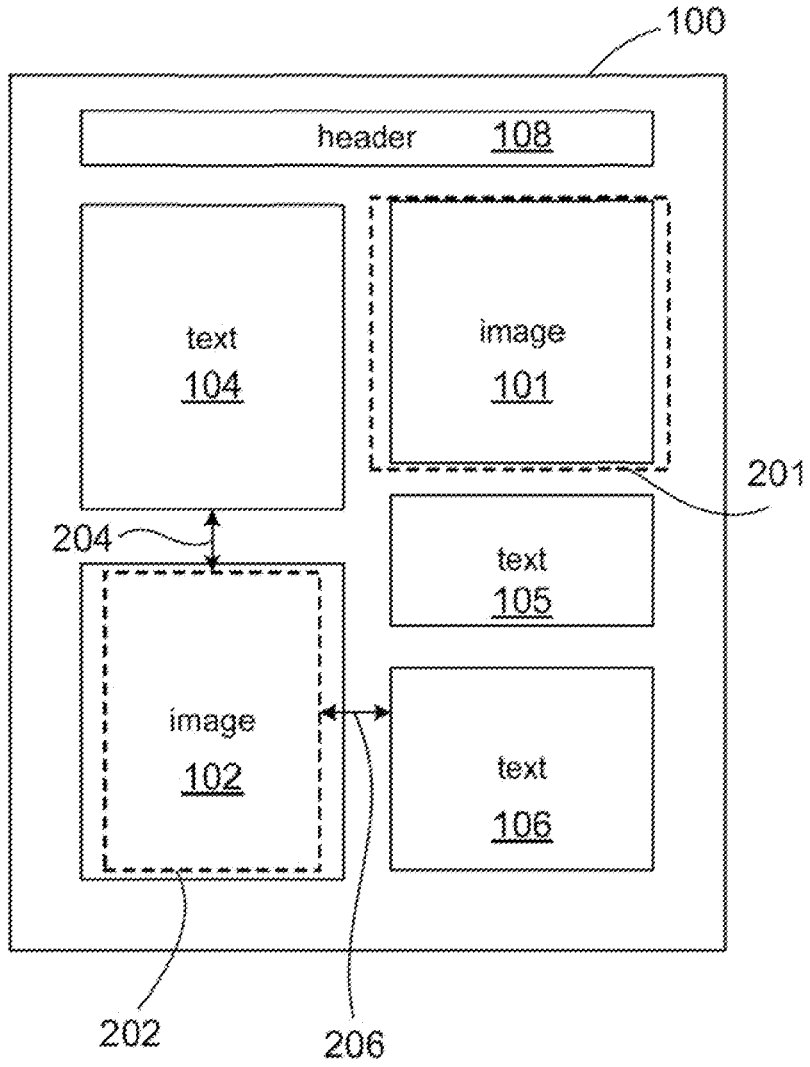


Fig. 1

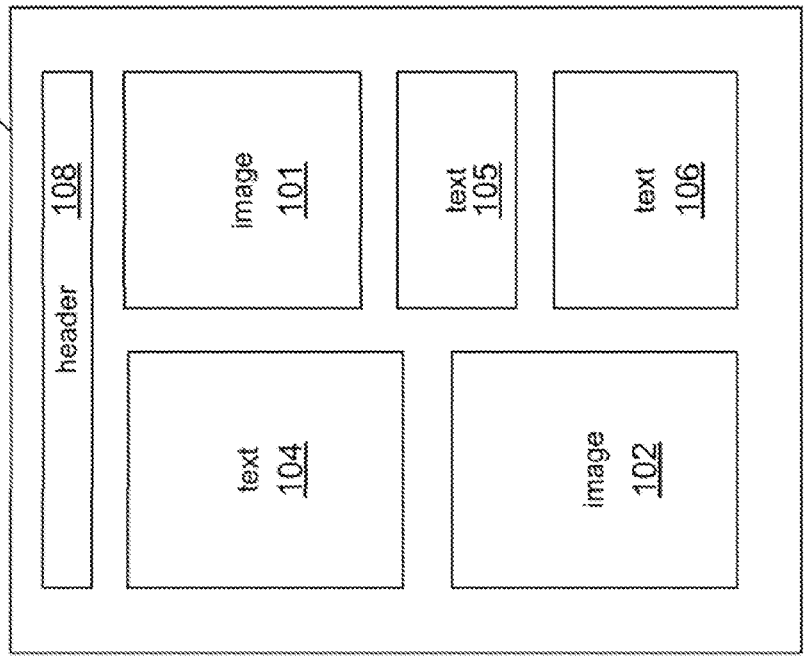
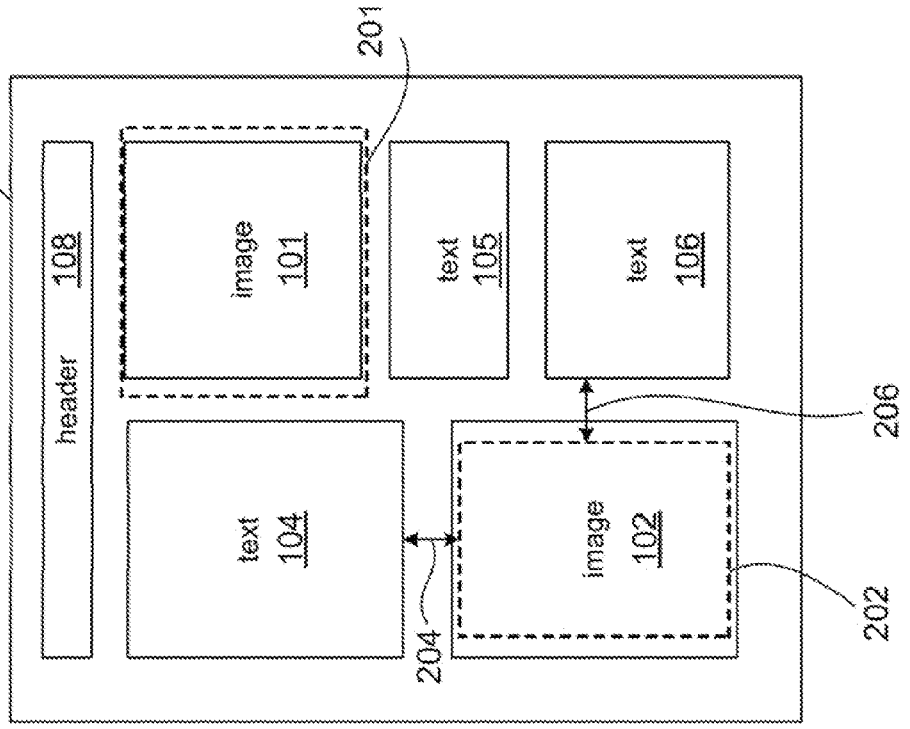

100
header 108
image 101
text 105
text 106
text 104
image 102

Fig. 2


100
201
header 108
image 101
text 105
text 106
206
text 104
image 102
204
202

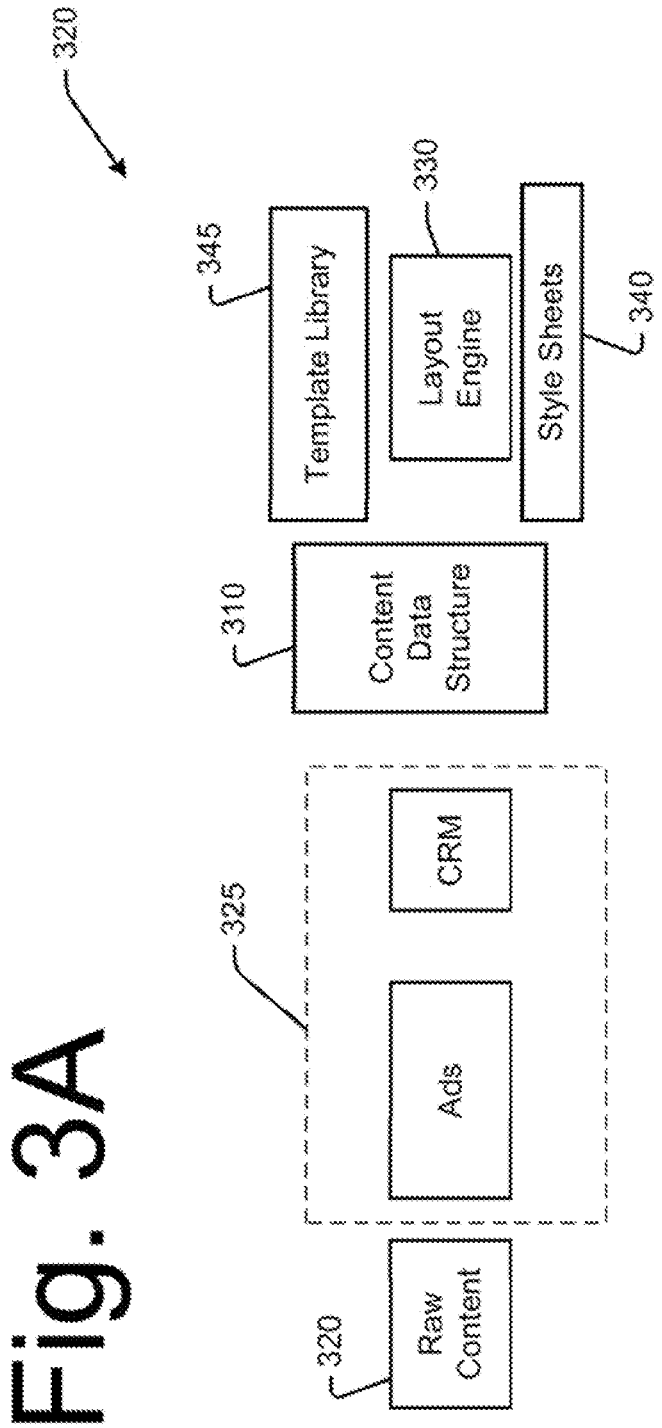

320
345
Template Library
330
Layout Engine
Style Sheets
340
310
Content Data Structure
325
CRM
Ads
320
Raw Content

Fig. 3A

Fig. 3B

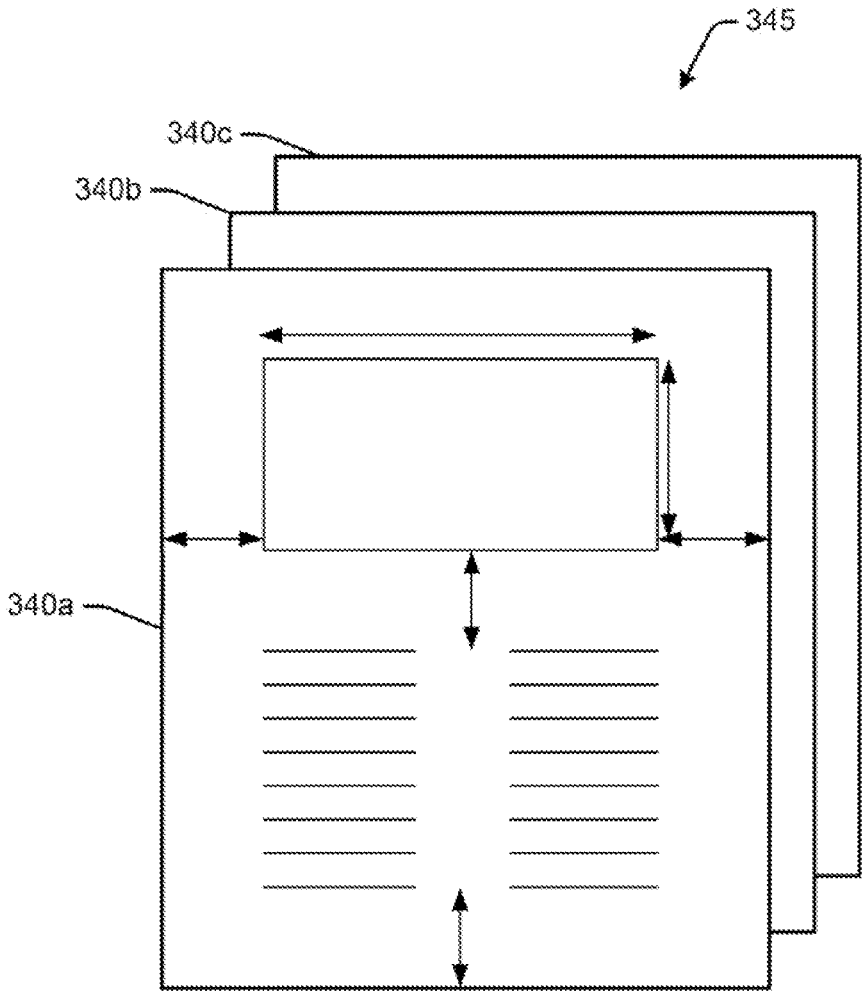


Fig. 4A

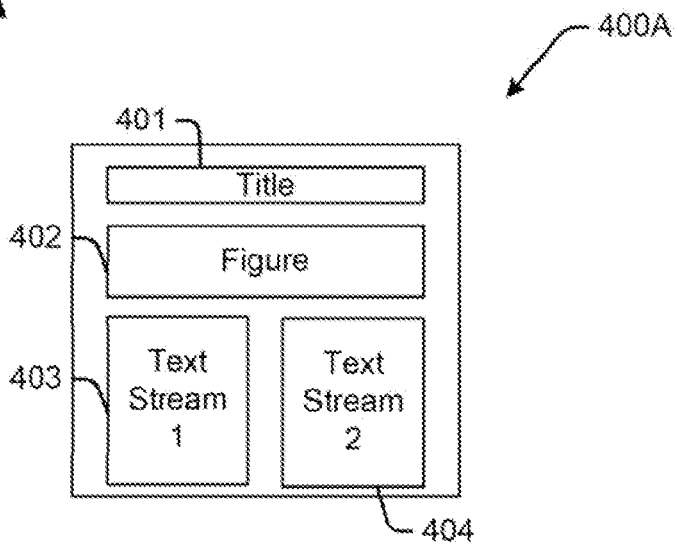

400A
401
Title
402
Figure
403
Text Stream 1
Text Stream 2
404

Fig. 4B

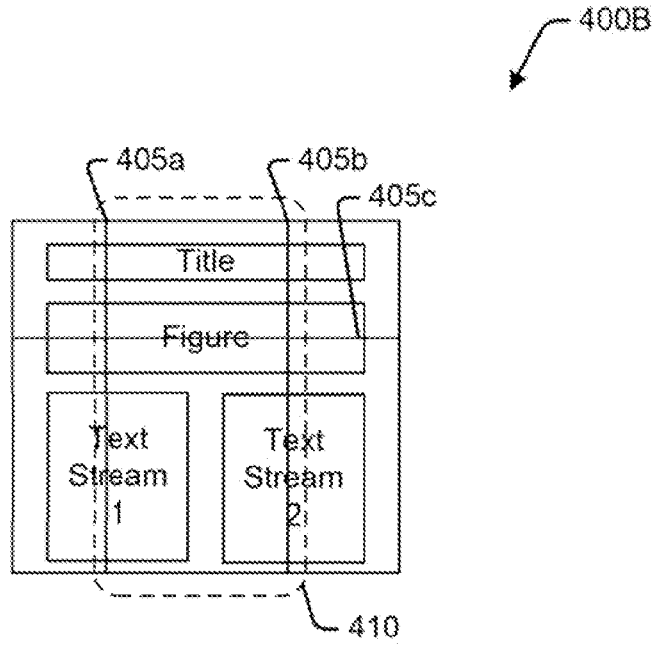

400B
405a
405b
405c
Title
Figure
Text Stream 1
Text Stream 2
410

Fig. 4C

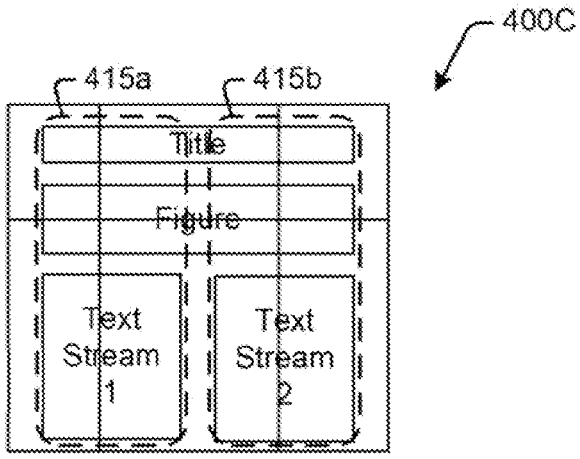
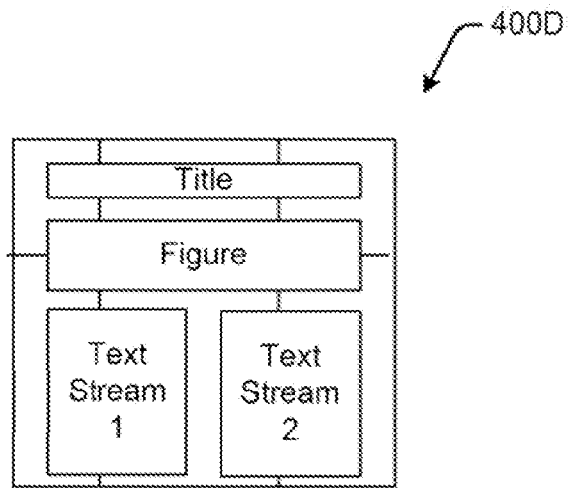

400C
415a
415b
Title
Figure
Text Stream 1
Text Stream 2

Fig. 4D


400D
Title
Figure
Text Stream 1
Text Stream 2

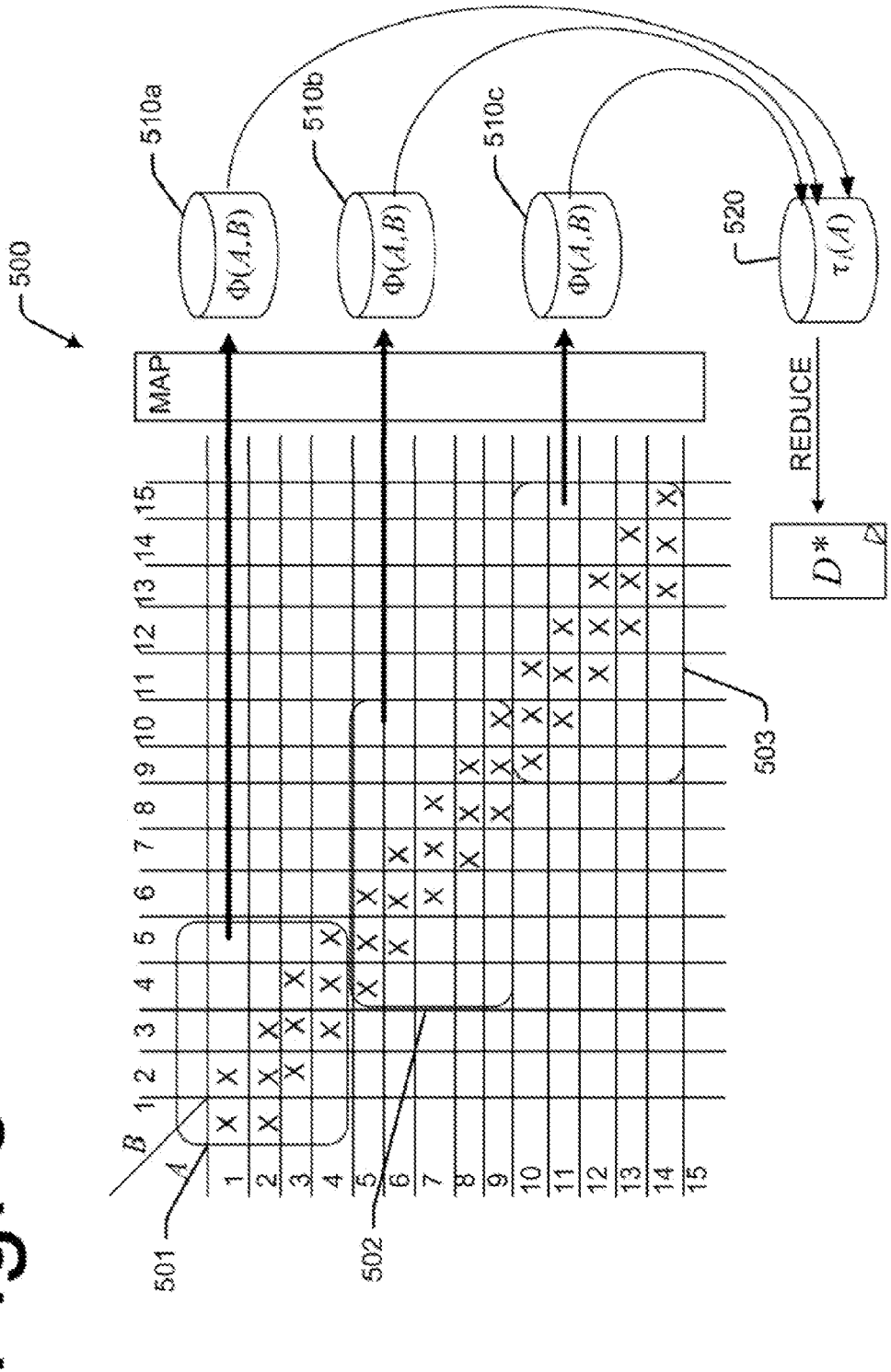

500
510a
510b
510c
520
Φ(A,B)
Φ(A,B)
Φ(A,B)
τ(A)
MAP
REDUCE
D*
503
501
502
B
A

Fig. 5

Fig. 6

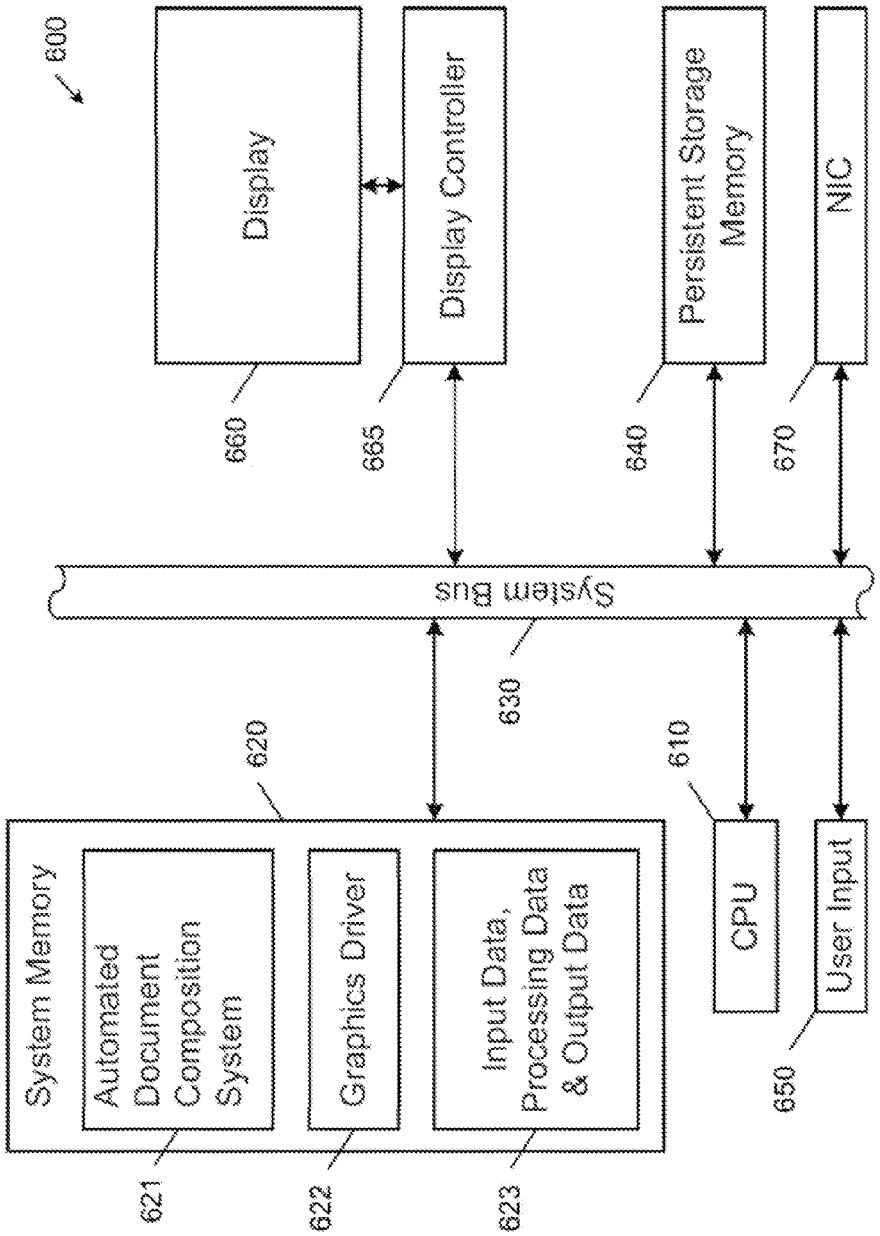
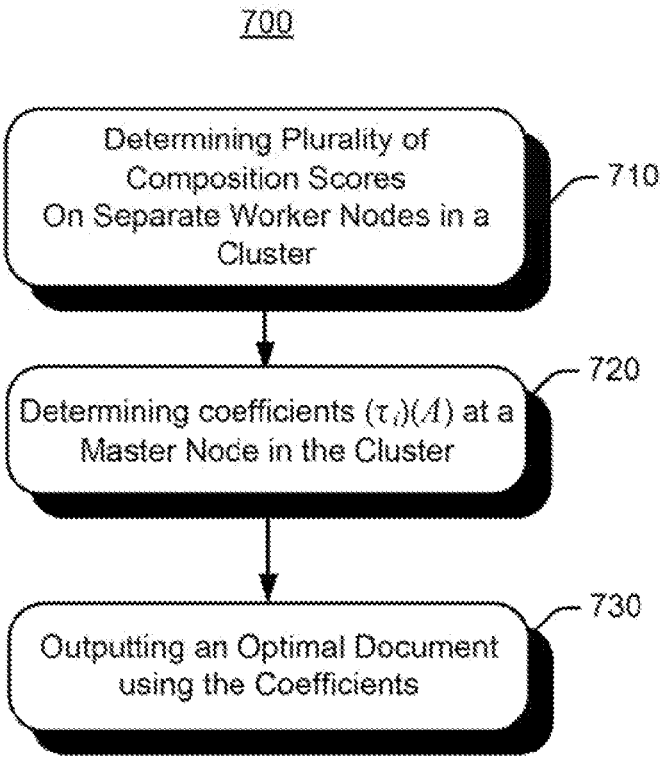

600
Display
Display Controller
Persistent Storage Memory
NIC
660
665
640
670
System Bus
630
620
610
System Memory
Automated Document Composition System
Graphics Driver
Input Data, Processing Data & Output Data
CPU
User Input
621
622
623
650

Fig. 7


700
Determining Plurality of Composition Scores On Separate Worker Nodes in a Cluster
710
Determining coefficients $(\tau_i)(A)$ at a Master Node in the Cluster
720
Outputting an Optimal Document using the Coefficients
730

AUTOMATED DOCUMENT COMPOSITION USING CLUSTERS

BACKGROUND

[0001] Micro-publishing has exploded on the Internet, as evidenced by a staggering increase in the number of blogs and social networking sites. Personalizing content allows a publisher to target content for the readers (or subscribers), allowing the publisher to focus on advertising and tap this increased value as a premium. But while these publishers may have the content, they often lack the design skill to create compelling print magazines, and often cannot afford expert graphic design. Manual publication design is expertise intensive, thereby increasing the marginal design cost of each new edition. Having only a few subscribers does not justify high design costs. And even with a large subscriber base, macro-publishers can find it economically infeasible and logistically difficult to manually design personalized publications for all of the subscribers. An automated document composition system could be beneficial.

BRIEF DESCRIPTION OF THE DRAWINGS

[0002] FIG. **1** shows an example of a template for a single page of a mixed-content document.

[0003] FIG. **2** shows the example template in FIG. **1** where two images are selected for display in the image fields.

[0004] FIG. **3**A is a high-level diagram showing an example implementation of automated document composition using PDM.

[0005] FIG. **3**B is a high-level diagram showing an example template library.

[0006] FIGS. **4**A-D show an example variable template in a template library.

[0007] FIG. **5** is a high-level illustration of example automated document composition in server clusters.

[0008] FIG. **6** is a high-level block diagram showing example hardware that may be implemented for automated document composition in server clusters.

[0009] FIG. **7** is a flowchart showing example operations for automated document composition in server clusters.

DETAILED DESCRIPTION

[0010] Automated document composition is a compelling solution for micro-publishers, and even macro-publishers. Both benefit by being able to deliver high-quality, personalized publications (e.g., newspapers, books and magazines), while reducing the time and associated costs for design and layout. In addition, the publishers do not need to have any particular level of design expertise, allowing the micro-publishing revolution to be transferred from being strictly "online" to more traditional printed publications.

[0011] Mixed-content documents used in both online and traditional print publications are typically organized to display a combination of elements that are dimensioned and arranged to display information to a reader (e.g., text, images, headers, sidebars), in a coherent, informative, and visually aesthetic manner. Examples of mixed-content documents include articles, flyers, business cards, newsletters, website displays, brochures, single or multi page advertisements, envelopes, and magazine covers, just to name a few examples. In order to design a layout for a mixed-content document, a document designer selects for each page of the document a number of elements, element dimensions, spacing between elements called "white space," font size and style for text, background, colors, and an arrangement of the elements.

[0012] Arranging elements of varying size, number, and logical relationship onto multiple pages in an aesthetically pleasing manner can be challenging, because there is no known universal model for human aesthetic perception of published documents. Even if the published documents could be scored on quality, the task of computing the arrangement that maximizes aesthetic quality is exponential to the number of pages and is generally regarded as intractable.

[0013] The Probabilistic Document Model (PDM) overcomes these classical challenges by allowing aesthetics to be encoded by human graphic designers into elastic templates, and efficiently computing the best layout while also maximizing the aesthetic intent. While the computational complexity of the serial PDM is linear in the number of pages and in content units, the performance is insufficient for interactive applications, where either a user is expecting a preview before placing an order, or is expecting to interact with the layout in a semi-automatic fashion.

[0014] Advances in computing devices have accelerated the growth and development of software-based document layout design tools and, as a result, have increased the efficiency with which mixed-content documents can be produced. A first type of design tool uses a set of gridlines that can be seen in the document design process but are invisible to the document reader. The gridlines are used to align elements on a page, allow for flexibility by enabling a designer to position elements within a document, and even allow a designer to extend portions of elements outside of the guidelines, depending on how much variation the designer would like to incorporate into the document layout. A second type of document layout design tool is a template. Typical design tools present a document designer with a variety of different templates to choose from for each page of the document.

[0015] FIG. **1** shows an example of a template **100** for a single page of a mixed-content document. The template **100** includes two image fields **101** and **102**, three text fields **104-106**, and a header field **108**. The text, image, and header fields are separated by white spaces. A white space is a blank region of a template separating two fields, such as white space **110** separating image field **101** from text field **105**. A designer can select the template **100** from a set of other templates, input image data to fill the image fields **101** and text data to fill the text fields **104-106** and the header **108**.

[0016] However, many procedures in organizing and determining an overall layout of an entire document continue to require numerous tasks that are to be completed by the document designer. For example, it is often the case that the dimensions of template fields are fixed, making it difficult for document designers to resin images and arrange text to fill particular fields creating image and text overflows, cropping, or other unpleasant scaling issues.

[0017] FIG. **2** shows the template **100** where two images, represented by dashed-line boxes **201** and **202**, are selected for display in the image fields **101** and **102**. As shown in the example of FIG. **2**, the images **201** and **202** do not fit appropriately within the boundaries of the image fields **101** and **102**. With regard to the image **201**, a design tool may be configured to crop the image **201** to fit within the boundaries of the image field **101** by discarding what it determines as peripheral portions of the image **201**, or the design tool may attempt to fit the image **201** within the image field **101** by rescaling the aspect ratio of the image **201**, resulting in a

visually displeasing distorted image **201**. Because image **202** fits within the boundaries of image field **102** with room to spare, white spaces **204** and **206** separating the image **202** from the text fields **104** and **106** exceed the size of the white spaces separating other elements in the template **100** resulting in a visually distracting uneven distribution of the elements. The design tool may attempt to correct for this by rescaling the aspect ratio of the image **202** to fit within the boundaries of the image field **102**, also resulting in a visually displeasing distorted image **202**.

[0018] The systems and methods described herein use automated document composition for generating mixed-content documents. Automated document composition can be used to transform marked-up raw content into aesthetically-pleasing documents. Automated document composition may involve pagination of content, determining relative arrangements of content blocks and determining physical positions of content blocks on the pages.

[0019] FIG. **3**A is a high-level diagram **300** showing an example implementation of automated document composition using PDM. The content data structure **310** represents the input to the layout engine. In an example, the content data structure is an XML file. In a typical magazine example, there may be a stream of text, a stream of figures, a stream of sidebars, a stream of pull quotes, a stream of advertisements, and logical relationships between them. For purposes of illustration, FIG. **3**A shows a stream of text blocks, a stream of figures, and the logical linkages.

[0020] In the example shown in FIG. **3**A, the content **320** is decoupled from the presentation **325** which allows variation in the size, number and relationship among content blocks, and is the input to the automated publishing engine **330**. Adding or deleting elements may be accomplished by addition or deletion of sub-trees in the XML structure **310**. Content modifications simply amount to changing the content of an XML leaf-node.

[0021] Each content data structure **310** (e.g., an XML file) is coupled with a template or document style sheet **340** from a template library **345**. Content blocks within the XML file **310** have attributes that denote type. For example, text blocks may be tagged as head, subhead, list, pare, caption. The document style sheet **340** defines the type definitions and the formatting for these types. Thus the style sheet **340** may define a head to use Arial bold font with a specified font size, line spacing, etc. Different style sheets **340** apply different formatting to the same content data structure **310**.

[0022] It is noted that type definitions may be scoped within elements, so that two different types of sidebars may have different text formatting applied to text with a subhead attribute. The style sheet also defines overall document characteristics such as, margins, bleeds, page dimensions, spreads, etc. Multiple section of the same document may be formatted with different style sheets.

[0023] Graphic designers may design a library of variable templates. An example template library **345** is shown in high-level in FIG. **38**. Having human-developed templates **340***a-c* addresses creating an overarching model for human aesthetic perception. Different styles can be applied to the same template via style sheets as discussed above.

[0024] FIGS. **4**A-D show an example variable template in the template library. The template parameters (Θ's) represent white space, figure scale factors, etc. The design process to crests a template may include content block layout, specification of dimension (x and y) optimization paths and path groups, and specification of prior probability distributions for individual parameters.

[0025] A content block layout is illustrated in FIG. **4**A. A designer may place content rectangles **401-404** on the design canvas **400**. Three types of content blocks are supported in this example, including title **401**, figure **402**, and text blocks **403-404**. It is noted that text blocks **403-404** represent streams of text sub-blocks, and may include headings, subheadings, list items, etc. The types and formatting of sub-blocks that go in a text stream are defined in the document style sheet. Each template has attributes, such as background color, background image, first page template flag, last page template flag etc. that allow for common template customizations.

[0026] To specify paths and path groups, the designer may draw vertical and horizontal lines **405***a-c* across the page indicating paths what the layout engine optimizes. Specification of a path indicates the designer goal that content blocks and whitespace along the path conform to specified path heights (widths). These path lengths may be set to the page height (width) to encourage the layout engine to produce full pages with minimized under and overfill. Paths may be grouped together to indicate that text flow from one path to the next. FIG. **4**B is a design canvas **400**B showing an example path **405***a-c* and path group **410** specification. Further, content may be grouped together as a sidebar. FIG. **4**C is a design canvas **400**C showing a sidebar grouping **415***a-b* where the figure and text stream are grouped together into a sidebar. Thus FIG. **4**B shows two Y paths grouped into a single Y-path group **410**, and FIG. **4**C shows two Y paths grouped into two Y-Path groups **415***a-b*. The second Y-path group **415***b* contains a sidebar grouping. Text is not allowed to flow outside a sidebar or from one Y-path group to the next.

[0027] When the designer selects variable entry (e.g., in the user interface), the figure areas and X and Y whitespaces are highlighted for parameter specification (e.g., as illustrated by design canvas **400**D in FIG. **4**D). The parameters are set to fixed values inferred from the position on the canvas. The designer clicks on parameters that are to be variable and enters a minimum value, a maximum value, a mean value and a precision value for each desired variable. This process specifies a "prior" Gaussian distribution for each of the template parameters. It is a "prior" Gaussian distribution in the sense that it is specified before seeing actual content. For figures, width and height ranges, and a precision value for the scale factor are specified. The mean value of the scale parameter is automatically determined by the layout engine based on the aspect ratio of en actual image so as to make the figure as large as possible without violating the specified range conditions on width and height. Thus the scale parameter of a figure has a truncated Gaussian distribution with truncation at the mean. The designer can make aesthetic judgments regarding relative block placement, whitespace distribution, figure scaling etc. The layout engine strives to respect this designer "knowledge" as encoded into the prior parameter distributions.

[0028] The layout engine includes three components. A parser parses style sheets, templates, and input content into internal data structures. An inference engine computes the optimal layouts, given content. A rendering engine renders the final document.

[0029] There are three parsers, one each for style sheets, content, and templates. The style sheet parser reads the style

sheet for each content stream and creates a style structure that includes document style and font styles. The content parser reads the content stream and creates an array of structures for figures, text and sidebars respectively.

[0030] The text structure array (also referred to herein as a "chunk array") includes information about each independent "chunk" of text that is to be placed on the page. A single text block in the content stream may be chunked as a whole if text cannot flow across columns or pages (e.g., headings and text within sidebars). However, if the text block is allowed to flow (e.g., paragraphs and lists), the text is first decomposed into smaller chunks that are rendered atomically. Each structure in the chunk array can include an index in the array, chunk height, whether a column or page break is allowed at the chunk, the identity of the content block to which the chunk belongs, the block type and an index into the style array to access the style to render the chunk. The height of a chunk is determined by rendering the text chunk at all possible text widths using the specified style in an off screen rendering process. In en example, the number of lines and information regarding the font style and line spacing is used to calculate the rendered height of a chunk.

[0031] Each figure structure in the figure array encapsulates the figure properties of an actual figure in the content stream such as width, height, source filename, caption and the text block identity of a text block which references the figure. Figure captions are handled similar to a single text chunk described above allowing various caption widths based on where the caption actually occurs in a template. For example, full width captions span text columns, while column width captions span a single text column.

[0032] Each content sidebar may appear in any sidebar template slot (unless explicitly restricted), so the sidebar array has elements that are themselves arrays with individual elements describing allocations to different possible sidebar styles. Each of these structures has a separate figure array and chunk array for figures and text that appear within a particular template sidebar.

[0033] The inference engine is part of the layout engine. Given the content, style sheet, and template structures, the inference engine solves for a desired layout of the given content. In en example, the inference engine simultaneously allocates content to a sequence of templates chosen from the template library, and solves for template parameters that allow maximum page fill while incorporating the aesthetic judgements of the designers encoded in the prior parameter distributions. The inference engine is based on a framework referred to as the Probabilistic Document Model (PDM), which models the creation and generation of arbitrary multi-page documents.

[0034] A given set of all units of content to be composed (e.g., images, units of text, and sidebars) is represented by a finite set c that is a particular sample of content from a random set C with sample space comprising sets of all possible content input sets. Text units may be words, sentences, lines of text, or whole paragraphs. Text units may be words, sentences, lines of text, or whole paragraphs. To use lines of text as an atomic unit for composition, each paragraph is decomposed first into lines of fixed column width. This can be done if text column widths are known and text is not allowed to wrap around figures. This method is used in all examples due to convenience and efficiency.

[0035] The term c' denotes a set comprising all sets of discrete content allocation possibilities over one or more pages, starting with and including the first page. Content subsets that do not form valid allocations (e.g., allocations of non-contiguous lines of text) do not exist in c'. If there are 3 lines of text and 1 floating figure to be composed, e.g., $c=\{l_1, l_2, l_2, f_1\}$ while $c'=\{\{l_1,\}, \{l_1, l_2\}, \{l_1, l_2, l_3\}, \{f_1\}, \{l_1 f_1\}, \{l_1, l_2, f_1\}, \{l_1 l_2, l_3, f_1\}\} \cup \{0\}$. It is noted that the specific order of elements within an allocation set is not necessary, because $\{l_1, l_2, f_1\}$ and $\{l_2, f_1, l_2\}$ refer to an allocation of the same content. However an allocation $\{l_1, l_3, f_1\} \notin c'$ means that lines 1 and 3 cannot be in the same allocation without including line 2. In addition, c' includes the empty set to allow for the possibility of a null allocation.

[0036] The index of a page is represented by $i \geq 0$. C_i is a random set representing the content allocated to page i. $C_{\leq i} \in c'$ is a random set of content allocated to pages with index 0 through i. Hence:

$$C_{\leq i}=\cup_{j=0}{}^{i}C_j$$

[0037] If $C_{si}=C_{si-1}$, then $C_l=0$ (i.e., page i has no content allocated). For convenience of this discussion, $C_{\leq i}=0$ and all pages $i \geq 0$ have valid content allocations to the previous i-l pages.

[0038] The probabilistic document model (PDM) is a probabilistic framework for adaptive document layout that supports automated generation of paginated documents for variable content. PDM encodes soft constraints (aesthetic priors) on properties, such as, whitespace, image dimensions, and image resealing preferences, and combines all of these preferences with probabilistic formulations of content allocation and template choice into a unified model According to PDM, the i^{th} page of a probabilistic document may be composed by first sampling random variable T_i from a set of template indices with a number of possible template choices (representing different relative arrangements of content), sampling a random vector θ_i of template parameters representing possible edits to the chosen template, and sampling a random set C_i of content representing content allocation to that page (or "pagination"). Each of these tasks is performed by sampling from an underlying probability distribution.

[0039] Thus, a random document can be generated from the probabilistic document model by using the following sampling process for page $i \geq 0$ with $C_{\leq -l}=0$;

[0040] sample template t, from $\mathbb{P}_i(T_i)$

[0041] sample parameters θ_i from $\mathbb{P}(\Theta_i|t_i)$

[0042] sample content $c_{\leq i}$ from $\mathbb{P}(C_{\leq i}|c_{\leq i-1}, \theta_t, l_i)$

$$c_i=c_{\leq i}-c_{\leq i-1}$$

[0043] The sampling process naturally terminates when the content runs out. Since this may occur at different random page counts each time the process is initiated, the document page count I is itself a random variable defined by the minimal page number at which $C_{\leq 1}=c$. A document V in PDM is thus defined by a triplet D of random variables representing the various design choices made in the above equations.

[0044] For a specific content c, the probability of producing document D of I pages via the sampling process described in this section is simply the product of the probabilities of all design (conditional) choices made during the sampling process. Thus,

$$\mathbb{P}(\mathcal{D}; I) = \prod_{i=0}^{I-i} \mathbb{P}(C_{\leq i} \mid C_{\leq i-1}, \Theta_i, T_i)\mathbb{P}(\Theta_i \mid T_i)\mathbb{P}_i(T_i)$$

[0045] The task of computing the optimal page count and the optimizing sequences of templates, template parameters, content allocations that maximize overall document probability is referred to herein as the model inference task, which can be expressed as:

$$(\mathcal{D}^*, I^*) = \underset{\mathcal{D}, I \geq 1}{\operatorname{argmax}} \mathbb{P}(\mathcal{D}; I)$$

[0046] The optimal document composition may be computed in two passes. In the forward pass, the following coefficients are recursively computed, for all valid content allocation sets $A \supseteq B$ as follows

$$\Psi(\mathcal{A}, \mathcal{B}, T) = \max_{\Theta} \mathbb{P}(\mathcal{A} \mid \mathcal{B}, \Theta, T)\mathbb{P}(\Theta \mid T)$$

$$\Phi_i(\mathcal{A}, \mathcal{B}) = \max_{T \in \Omega_s} \Psi(\mathcal{A}, \mathcal{B}, T)\mathbb{P}_i(T),\ i \geq 0,$$

$$\tau_i(\mathcal{A}) = \max_{\mathcal{B}} \Phi, (\mathcal{A}, \mathcal{B})\tau_{i-1}(\mathcal{B}),\ i \geq 1$$

[0047] In the equations above, $\tau_0(A)=\Phi_0(A, 0)$. Computation of $\tau_i(A)$ depends on $\Phi_i(A, B)$, which in turn depends on $\psi(A, B, T)$. In the backward pass, the coefficients computed in the forward pass are used to infer the optimal document. This process is very fast, involving arithmetic and lookups. The entire process is dynamic programming with the coefficients $\tau_i(A)$, $\Phi_i(A, B)$ and $\psi(A, B, T)$ playing the role of dynamic programming tables. The following discussion focuses on parallelizing the forward pass of PDM inference, which is the most computationally intensive part.

[0048] The innermost function $\psi(A, B, T)$ can be determined as a score of how we content in the set A-B is suited for template T. This function is the maximum of a product of two terms. The first term $\mathbb{P}$ (A|B, Θ, T) represents how we content fills the page and respects figure references, while the second term $\mathbb{P}$ (ƒ|T) assesses how close, the parameters of a template are to the designer's aesthetic preference. Thus the overall probability (or "score") is a tradeoff between page fill and a designer's aesthetic intent. When there are multiple parameters settings that fill the page equally well, the parameters that maximize the prior (and hence are closest to the template designer's desired values) are favored.

[0049] The function $\Phi_i(A, B)$ scores how well content A-B can be composed onto the i^{th} page, considering all possible relative arrangements of content (templates) allowed for that page. $\mathbb{P}_i(T)$ allows the score of certain templates to be increased, thus increasing the chance that these templates are used in the final document composition.

[0050] Finally function $\tau_i(A)$ is a pure pagination score of the allocation A to the first i pages. The recursion $\tau_i(A)$ means that the pagination score for an allocation A to the first i pages, $\tau_i(A)$ is equal to the product of the best pagination score over all possible previous allocations B to the previous (i−1) pages with the score of the current allocation A-B to the i^{th} page (A, B).

[0051] The PDM process can be used to back out the optimal templates to compose each page of the document composition. The way in which these calculations are distributed among different computational units in a server cluster processing environment has to do with the degree of dependency and synchronization mechanisms. Three types of degrees of dependency can be distinguished among the computations: (a) independent computations, (b) dependent computations, and (c) partially dependent computations.

[0052] An example of independent computations is the sums involved in the component-wise sum of two vectors (a, b). The sum of each component, (a_i+b_i) is unrelated to the sum the other components. Therefore, it does not matter if the threads to which each of these sums is assigned can communicate with each other.

[0053] An example of dependent computations is the calculations involved in obtaining all the values of a recursion, such as $x_{i+1}=f(x_i)$. Proceeding to compute x_{10} occurs after computing x_9. Hence, all of these computations can be computed by the same thread sequentially. There can be less benefit in having different threads to compute these different x_i, either inside different thread-blocks or using the same thread-blocks.

[0054] An example of partially dependent computations is the comparisons involved in determining the maximum value over a set of values using parallel reduction, e.g., $\max_{i\in(1, 2, \ldots 32)} \theta_i$. At an initial stage, b1 is computed as $b_1=\max(a_1, a_{17})$, $b2=\max(a_2, a_{18}), \ldots b_{16}=\max(a_{16}, a_{32})$. However, computations cannot proceed to the next process, e.g., computing $c_1=\max\{b_1, b_8\}$, $c_2=\max\{b_2, b_9\}, \ldots c_s=\max\{b_8, b_6\}$), until all b's have been calculated. In short, there is some dependency among the computations, and although at a given level (e.g., b_is level) each comparison can be done in a separate thread, all threads should belong to the same block so that after each process the output can synchronize before going to the next process in the reduction.

[0055] The automated publishing can be executed in a server cluster processing environment using these general notions of dependency. In an example, serial procedures (e.g., shown herein as algorithms) may be mapped to multiple server nodes using a computational paradigm known as "MAP-REDUCE." MAP-REDUCE is a software framework first introduced in the computing industry to support distributed computing on large data sets on clusters of computers. MAP-REDUCE is now available on many commercial cloud computing offerings.

[0056] In a MAP operation, a master node converts an input "problem" into smaller "sub-problems," and distributes those sub-problems to "worker" nodes. The worker node processes the sub-problem, and passes a result back to a master node. In the REDUCE operation the master node then takes the results from all of the sub-problems and combines the results to obtain a solution to the input problem.

[0057] FIG. **5** is a high-level illustration of example automated document composition in server clusters. In this example it can be seen how the computation of Φs may be distributed to the worker nodes. It can also be seen how the collected data can be "REDUCED" to compute the rs on the master node.

[0058] In an example, the sub-problems sent to the server nodes are the computation of the $\Phi_i(A, B)$ for all:

$$A, B \in C'$$

[0059] The set A-B can be effectively bound to represent the content allocated to a page. This implies that all legal subsets A and B do not need to considered in building $\Phi_i(A, B)$, but those that are close enough are considered so that the content A-B can reasonably be expected to fit on a page. The computation of (A, B) depends on i since the maximization over allowed templates for each page in $\Phi_i(A, B)$ occurs over sub-libraries that depend on i. However, since in practice the number of distinct template sub-libraries is quite small (typically first, last, odd and even page templates are drawn from distinct libraries), the computation of $\Phi_f(A, B)$ for any i can be reduced to computation of $\Phi_{first}(A, B)$, $\Phi_{last}(A, B)$, $\Phi_{odd}(A, B)$ and $\Phi_{even}(A, B)$. This means that each distributed server node essentially computes odd (A, B) and even (A, B) for most content. As a simplification (without loss of generality) all templates for all pages are sampled from a single template library, so the subscript can he dropped and $\Phi_f(A, B)$ can be written as$\Phi(A, B)$.

[0060] FIG. **5** shows how the computation of the Φs can be distributed to the worker nodes, and shows how the collected data may be reduced to compute the τs on the master node. To provide intuition about the mapping, each content allocation set in c' is associated with a number. Close numbers represent close sets, and supersets receive larger numbers than subsets. Therefore, a grid of possible content allocations (A, B) can be assumed, as shown in FIG. **1**. Because A-B represents the content allocated to a page, it is bounded by page dimensions.

[0061] Accordingly, relatively few diagonal and neighboring elements are actually computed (regions designated "X" in FIG. **5**), although each node **510***a-c* receives a block of computation (blocks inside boundaries **501-503** without an "X" designation in FIG. **5**). The content allocations lie along the diagonal of the grid if there is a single possible content ordering (no floating elements).

[0062] It is noted that the illustration shown in FIG. **5** is intended to provide a visual representation showing that a small portion of the entire grid has meaningful allocations for which (A, B) are computed. In general, for each A the allowed B's are in a neighborhood which can be expressed as:

$$N_f(A)=\{B:d(A-B)\leq f\}$$

[0063] The function d(A-B) returns a vector of the counts of various page elements in the set A-B. f is a vector that expresses what is meant to be close by bounding the numbers of various page elements allowed on a page. For example $f=[100(\text{lines}), 2(\text{figures}), 1(\text{sidebar})]^T$. This eliminates an allocation where $d(A-B)=[110(\text{lines}), 2(\text{figures}), 1(\text{sidebar})]^T$.

[0064] The master node **520** receives all the computed Φs from worker nodes **510***a-c*, and computes the $\tau_f(A)$ coefficients. Master node **520** also performs a sequential backward pass algorithm (associated with the procedure) to obtain the final document D*. Pseudo code for the Map and Reduce functions is shown for an example below by Algorithms 2 and 3. With reference to FIG. **5**, instead of a full block decomposition, a row-based decomposition is used for the Map operation. Thus each Map computes (A, B) for a given A for B's in the neighborhood of A. Line 3 in the example Algorithm 1 may be computed efficiently if the distributions are parameterized.

Algorithm 1 Code to compute Φ(A, B) in Map step

```
1: Φ(A, B) = 0
2: for all T ϵ Ω do
3:    Ψ(A, B, T) = max_Θ ℙ(A|B, Θ, T) ℙ(Θ|T)
4:    if Φ(A, B) < Ψ(A, B, T) ℙ(T) then
5:       Φ(A, B) = Ψ(A, B, T) ℙ(T)
6:    end if
7: end for
```

Algorithm 2 Map(key = A, value = f)

```
1: for all B ϵ c^l : A − B ϵ N_f(A) do
2:    Emit key = "l", value = (A, B, Φ(A, B))
3: end for
```

Algorithm 3 Reduce(key = "l", values = (A, B, Φ(A, B))) ∀ A, B

```
1: τ_0(A) = Φ_0(A, ∅), ∀ A ϵ c^l
2: τ_i(A) = 0, ∀ A ϵ c^1, ∀i ≥ 1
3: for all A do
4:    for all B corresponding to specific A do
5:       for i = 1 to I do
6:          if τ_i(A) ≤ Φ(A, B)τ_{i−1}(B) then
7:             τ_i(A) = Φ(A, B)τ_{i−1}(B)
8:          end if
9:       end for
10:   end for
11:   Emit key=(i,A) value = τ_i(A)
12: end for
```

[0065] The information that each computer receives initially is a data structure containing the layout information of each piece involved in composing the document. This structure includes the dimensions of each picture, the layout of each template, the structure of each side bar and the size of each line of text, it is noted, however, that this structure does not include the actual lines of text or images that go into composing the final document. The structures therefore a small byte size.

[0066] A simple formula is deduced that shows how the theoretical total operation time depends on the number of computers, N, among which the work is distributed. Let the number of sets A for which to compute (A, B) be N_C, a constant. Now assume A is fixed, since there is a restriction on the maximum content per page, the number of sets B for which are going to compute (A, B), is bounded by a constant. In the beginning, the same data structure is broadcast to all of the nodes. This takes a fixed time tD. After that, each of the N nodes computes a set of coefficients. This computation is done in parallel among all nodes, and takes a time proportional to N_CI N. After all the coefficients are computed, the coefficients are transmitted to the (N+1)th node. Since there is one receiving node, and because the amount of information to be transmitted by each node is proportional to the number of coefficients, this takes a time that is proportional to $N\times(N_C/N)$. After the Reducer receives all the coefficients, this node computes the $\tau_i(A)$ coefficients and determines the optimal document.

[0067] FIG. **6** is a high-level block diagram **600** showing example hardware that may be implemented for automated

document composition. In this example, a computer system **600** is shown that can implement any of the examples of the automated document composition system **621** that are described herein. The computer system **600** includes a processing unit **710** (CPU), a system memory **620**, and a system bus **630** that couples processing unit **610** to the various components of the computer system **600**. The processing unit **610** typically includes one or more processors, each of which may be in the form of any one of various commercially available processors. The system memory **620** typically includes a read only memory (ROM) that stores a basic input/output system (BIOS) that contains start-up routines for the computer system **600** and a random access memory (RAM). The system bus **146** may be a memory bus, a peripheral bus or a local bus, and may be compatible with any of a variety of bus protocols, including PCI, VESA, Microchannel, ISA, and EISA. The computer system **600** also includes a persistent storage memory **640** (e.g., a hard drive, a floppy drive, a CD ROM drive, magnetic tape drives, flash memory devices, and digital video disks) that is connected to the system bus **630** and contains one or more computer-readable media disks that provide non-volatile or persistent storage for data, data structures and computer-executable instructions.

[0068] A user may interact (e.g., enter commands or data with the computer system **600** using one or more input devices **650** (e.g., a keyboard, a computer mouse, a microphone, joystick, and touch pad). Information may be presented through a user interface that is displayed to a user on the display **660** (implemented by, e.g., a display monitor), that is controlled by a display controller **665** (implemented by, e.g., a video graphics card). The computer system **600** also typically includes peripheral output devices, such as a printer. One or more remote computers may be connected to the computer system **600** through a network interface card (NIC) **670**.

[0069] As shown in FIG. **6**, the system memory **620** also stores the automated document composition system **621**, a graphics driver **622**, and processing information **623** that includes input data, processing data, and output data.

[0070] The automated document composition system **621** can include discrete data processing components, each of which may be in the form of any one of various commercially available data processing chips. In some implementations, the automated document composition system **621** is embedded in the hardware of any one of a wide variety of digital and analog computer devices, including desktop, workstation, and server computers. In some examples, the automated document composition system **621** executes process instructions machine-readable instructions, such as but not limited to computer software and firmware) in the process of implementing the methods that are described herein. These process instructions, as well as the data generated in the course of their execution, are stored in one or more computer-readable media. Storage devices suitable for tangibly embodying these instructions and data include ail forms of non-volatile computer-readable memory, including, for example, semiconductor memory devices, such as EPROM, EEPROM, and flash memory devices, magnetic disks such as internal hard disks and removable hard disks, magneto-optical disks, DVD-ROM/RAM, and CD-ROM/RAM.

[0071] FIG. **7** is a flowchart showing example operations for automated document composition in server clusters. Operations **700** may be embodied as machine readable instructions on one or more computer-readable medium. When executed on a processor, the instructions cause a general purpose computing device to be programmed as a special-purpose machine that implements the described operations. In an example implementation, the components and connections depicted in the figures may be used.

[0072] An example of a method of automated document composition in server clusters may be carried out by program code stored on non-transient computer-readable medium and executed by processor(s).

[0073] In operation **710**, determining a plurality of composition scores $\Phi_i(A, B)$, the composition scores each computing separately on a plurality of worker nodes in the duster.

[0074] In operation **720**, determining coefficients $(\tau_i)(A)$ at a master node in the cluster based on the composition scores (Φ_i) from each of the worker nodes.

[0075] In operation **730**, outputting an optimal document (D*) using the coefficients (τ_i).

[0076] The operations shown and described herein are provided to illustrate example implementations, it is noted that the operations are not limited to the ordering shown. Still other operations may also be implemented.

[0077] In an example of further operation, A and B may be subsets of original content a (C). The composition scores may be for allocating content (A) to the first i pages in a document, and allocating content (B) to the first i−1 pages in the document. The composition scores may represent how well content A-B fits the ith page over templates T from a library of templates used to lay out original content (C).

[0078] In further operations, all Bs are computed for a given A by a single worker node.

[0079] In another example of further operations, all worker nodes may receive a data structure including layout information of each component for composing the document. The layout information may include dimensions of each component for composing the document. The layout information may include layout of each template for composing the document. The layout information may include structure of each component for composing the document. The layout information may not include actual text or images.

[0080] It is noted that the example embodiments shown and described are provided for purposes of illustration and are not intended to be limiting. Still other embodiments are also contemplated.

1. A method of automated document composition using clusters, comprising:

determining a plurality of composition scores $\Phi_i(A, B)$, the composition scores each computing separately on a plurality of worker nodes in the cluster;

determining coefficients $(\tau_i)(A)$ at a master node in the duster based on the composition scores (Φ_i) from each of the worker nodes; and

outputting an optimal document (D*) using the coefficients (τ_i).

2. The method of claim **1**, wherein A and B are subsets of original content (C).

3. The method of claim **1**, wherein the composition scores are for allocating content (A) to the first i pages in a document, and allocating content (B) to the first i−1 pages in the document.

4. The method of claim **1**, wherein the composition scores represent how well content A-B fits the ith page over templates T from a library of templates used to lay out original content (C).

5. The method of claim **1**, wherein all Bs are computed for a given A by a single worker node.

6. The method of claim **1**, wherein all worker nodes receive a data structure including layout information of each component for composing the document.

7. The method of claim **6**, wherein the layout information includes dimensions of each component for composing the document.

8. The method of claim **6**, wherein the layout information includes layout of each template for composing the document.

9. The method of claim **6**, wherein the layout layout information includes structure of each component for composing the document.

10. The method of claim **6**, wherein the layout information does not include actual text or images.

11. A system comprising a computer readable storage to store program code executable for automated document composition using clusters, the program code comprising instructions to:

determine a plurality of composition scores $\Phi_i(A, B)$ on a plurality of worker nodes in the cluster;

determine coefficients $(\tau_i)(A)$ at a master node in the cluster based on the composition scores (Φ_i) from each of the worker nodes; and

output an optimal document (D*) using the coefficients (τ_i).

12. The system of claim **11**, wherein the worker nodes are provided in a cloud computing environment.

13. The system of claim **11**, wherein serial operations are mapped to multiple worker nodes using "MAP-REDUCE."

14. The system of claim **13**, wherein in a MAP operation, the master node converts input into sub-problems and distributes the subproblems to the worker nodes.

15. The system of claim **14**, wherein the worker nodes process the sub-problem, and return results back to the master node.

16. The system of claim **15**, wherein in a REDUCE operation the master node combines the results from all of the worker nodes to determine the coefficients (τ_j).

17. A system comprising a computer readable storage to store program code executable by a multi-core processor to:

separately compute a plurality of composition scores $\Phi_i(A, B)$ on a plurality of worker nodes in a cluster;

compute coefficients $(\tau_i)(A)$ at a master node in the cluster based on the composition scores (Φ_i) from each of the worker nodes; and

output an optimal document (D*) using the coefficients (τ_i).

18. The system of claim **17**, wherein the worker nodes execute "MAP-REDUCE" in a cloud computing environment.

19. The system of claim **17**, wherein all Bs ace computed for a given A by a single worker node.

20. The system of claim **17**, wherein all worker nodes receive a data structure including layout information of each component of the document.

* * * * *