UNIVERSITÄT SALZBURG

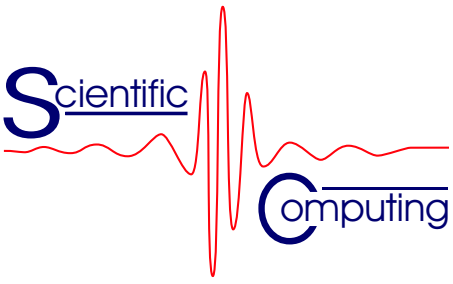
Accuracy of Distributed Data Retrieval on a Supercomputer Grid

Gabriel Okša[a] Marián Vajteršic

[a]Institute of Mathematics, Department of Informatics, Slovak Academy of Sciences, Bratislava, Slovak Republic

Technical Report 2005-09 December 2005

Department of Scientific Computing



Jakob-Haringer-Straße 2
5020 Salzburg
Austria
www.scicomp.sbg.ac.at

Technical Report Series

Accuracy of Distributed Data Retrieval on a Supercomputer Grid

Gabriel Okša* and Marián Vajteršic†

Abstract. *Using the paradigm of the Latent Semantic Indexing, two models for the encoding of a distributed database are described. In the first model, the computational system is organized into a grid of individual nodes, which possess only a restricted inter-communication. The distributed database consists of mutually uncoupled local databases, which are updated for new documents/terms, and scanned independently for documents matching a given query. There is no easy way of comparing the accuracy of retrieval from individual nodes, because there exists no global approximation of the whole database. In the second model (which can describe, for example, one node of a grid), the individual processors are connected by some sort of the communication network, so that the distributed computation and mutual communication are possible. We develop a two-stage model—the local and global levels of approximation—based on the Latent Semantic Indexing of documents for such a distributed system. Some interesting computational issues are discussed including the efficiency of a distributed singular value decomposition. Finally, it is possible to analyze the relationship between the local and global approximations with respect to the accuracy of retrieval of documents in this case.*

1 Introduction

Human knowledge is growing in all areas of human activity. Despite wars and terrorism, despite occasional nature catastrophes, despite all kinds of obstacles—and it seems that this growth is even accelerating. Deep inside

*Institute of Mathematics, Department of Informatics, Slovak Academy of Sciences, Bratislava, Slovak Republic, email: Gabriel.Oksa@savba.sk.

†Institute for Scientific Computing, University of Salzburg, Salzburg, Austria, email: marian@cosy.sbg.ac.at.

each of us lives a desire to know and understand more than our parents have known and understood ...

One of the main sources of information in our society is the written word. Since times of Sumerians a written document became the main tool to inform, to teach, to entertain, to archive the knowledge. Today, some 6000 years after Sumerians, nothing has changed with respect to the importance of written text. What has changed significantly, however, is the way people manipulate texts. Sumerians had large libraries consisting of vast amounts of cuneiform tables, which have been excavated from sand by archaeologists in the southern Iraq during last 200 years. We have the linear algebra and computers.

To become widely available, the knowledge must be manipulated in an easy and reliable way. The manipulation includes storage, updating and retrieval of texts stored in huge databases, which reside on some kind of a computer architecture.

To begin with, we need a mathematical model for the *storage* of written text in a computer. Here comes the linear algebra into play. In the vector space model, the collection of text documents is represented by a *term-document* matrix $A = (a_{ij}) \in \mathbb{R}^{m \times n}$, where a_{ij} is based on the number of times the term i appears in the document j, m is the number of terms, and n is the number of documents in the collection. Hence, a document becomes a column vector. *Updating* the term-document matrix means either to add new documents / delete some old documents, or to add new terms / delete some existing terms. *Retrieval* is connected with user's query that can also be represented as a vector of the same dimension as any document. The similarity between a query vector and a document vector is usually measured by the cosine of the angle between them, and for each query a list of documents ranked in a decreasing order of similarity is returned to the user.

Latent semantic indexing (LSI) is a concept-based automatic indexing method for overcoming the two fundamental problems which exist in the traditional lexical-matching retrieval schemes: synonymy and polysemy [3]. With respect to the synonymy, several different words can be used to express a concept and the keywords in a user's query may not match those in the relevant documents. On the other hand, the polysemy means that the words can have multiple meanings and the user's words may match those in the irrelevant documents. The LSI is an extension of the vector space model for information retrieval [4, 3].

The LSI modifies this vector space model by modeling the term-document relationship using a *reduced-dimension representation* (RDR) of term-document

matrix Acomputed by its singular value decomposition (SVD). Let

$$A = P\Sigma Q^T, \quad \Sigma = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_{\min\{m,n\}}), \quad \sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_{\min\{m,n\}},$$

be the SVD of A. Then the RDR is given by the best rank-k approximations $A_k = P_k \Sigma_k Q_k^T$, $k < \min\{m, n\}$, where P_k and Q_k consist of the first k columns of P and Q, respectively, and Σ_k is the kth leading principal submatrix of Σ. Each of the k reduced dimensions represents a so-called *pseudo-concept* [4], which may not have any explicit semantic content but helps to discriminate documents [4, 5].

In rapidly changing environments such as the World Wide Web, the document collection is frequently updated with new documents and terms constantly being added. Hence, the task arises to efficiently update the old LSI-generated RDR after an addition of new documents and terms. In addition, really huge databases can not be implemented on a single computer; instead, they use networking and documents are stored in a distributed way.

The number of possibilities how to organize a computational architecture is countless. We restrict ourselves to two basic types. A *grid* consists of individual computing nodes which are connected by network. However, individual nodes are independent and they do not normally communicate. They process the information independently of each other. A node can be another computer or even some kind of a local distributed system—e.g., a cluster of personal computers or a parallel supercomputer. In contrast, a *distributed system* consists of a set of nodes which are connected and communicate with each other. A good example is the cluster of personal computers (PCs).

In what follows, the implementation of the concept of LSI is discussed for both types of computational architectures. All three aspects of the LSI—storage, updating and retrieval—are discussed and compared.

2 LSI implemented on a grid

Let let us assume that the computational grid consists of p nodes. In context of the LSI, each node can store and maintain its individual database, which is represented by its own term-document matrix $A^{(i)}$, $i = 1, 2, \ldots, p$.

In the following two subsections we will describe the algorithms, which can be used for updating of individual databases and the retrieval of documents in individual nodes.

2.1 Storage and updating

Let the index i denote one of the nodes of the grid under consideration, $1 \leq i \leq p$. This node first accumulates its term-document matrix $A^{(i)}$ of order $m \times n_i$, then computes its RDR of order k_i and, finally, stores the matrices P_{k_i}, Q_{k_i} and Σ_{k_i}. Although the individual nodes of the grid work independently, we assume that their data-term matrices $A^{(i)}$ describe information from the same (or very close) areas of interests, so that the terms are the same for all nodes—hence, the number of rows m is the same across the nodes.

2.1.1 Updating documents

Since each node of the grid works independently, it can receive new documents from the outside world. Therefore, it must be capable to update its term-document matrix $A^{(i)}$. However, the original $A^{(i)}$ is not at our disposal anymore—all, that is left, is only its k_i-dimensional approximation in the factored form. Therefore, a natural question arises, how to compute a new approximation, which will incorporate a new information from new documents.

Let $D^{(i)} \in \mathbb{R}^{m \times w_i}$ be w_i new documents. The task is to compute the best rank-k_i approximation of the column partitioned matrix

$$B^{(i)} \equiv (A_{k_i}^{(i)},\, D^{(i)}) \ .$$

Using the factorization of $A_{k_i}^{(i)}$, the matrix $B^{(i)}$ can be written as

$$\begin{aligned} B^{(i)} &= \left(P_{k_i} \Sigma_{k_i} Q_{k_i}^T,\, D^{(i)}\right) \\ &= \left(P_{k_i},\, (I_m - P_{k_i} P_{k_i}^T)\, D^{(i)}\right) \cdot \begin{pmatrix} \Sigma_{k_i} & P_{k_i}^T D^{(i)} \\ 0 & I_{w_i} \end{pmatrix} \cdot \begin{pmatrix} Q_{k_i}^T & 0 \\ 0 & I_{w_i} \end{pmatrix} . \end{aligned}$$

Note that $I_m - P_{k_i} P_{k_i}^T$ is the matrix representation of the orthogonal projection, which maps the columns of matrix $D^{(i)}$ into the subspace $\mathcal{P}_{k_i}^{\perp}$ that is orthogonal to the column range of matrix P_{k_i}. Let $(I_m - P_{k_i} P_{k_i}^T) D^{(i)} = \hat{P}_{w_i} R_i$ be the QR decomposition of the matrix $(I_m - P_{k_i} P_{k_i}^T) D^{(i)}$. Then

$$B^{(i)} = (P_{k_i},\, \hat{P}_{w_i}) \cdot \begin{pmatrix} \Sigma_{k_i} & P_{k_i}^T D^{(i)} \\ 0 & R_i \end{pmatrix} \cdot \begin{pmatrix} Q_{k_i}^T & 0 \\ 0 & I_{w_i} \end{pmatrix} . \tag{1}$$

The crucial point in the above derivation is the observation that the w_i orthonormal columns of matrix $\hat{P}_{w_i}$ are mutually orthogonal to the k_i orthonormal columns of matrix P_{k_i} because the columns of $\hat{P}_{w_i}$ constitute the

orthonormal basis of the subspace $\mathcal{P}_{k_i}^{\perp}$. Note that two exterior matrices on the right hand side of Eq. (1) are orthogonal, but the inner matrix is not diagonal. Hence, from the computational point of view, the updating problem is reduced to the SVD of the inner matrix in Eq. (1).

Based on these facts, Zha and Simon [6] have derived a method for solving the problem of updating documents. Their approach is summarized in Algorithm 1.

Algorithm 1 Algorithm for updating documents

1: *Input:* k_i, $P_{k_i} \in \mathbb{R}^{m \times k_i}$, $\Sigma_{k_i} \in \mathbb{R}^{k_i \times k_i}$, $Q_{k_i} \in \mathbb{R}^{n \times k_i}$, $D^{(i)} \in \mathbb{R}^{m \times w_i}$.

2: Compute the projection: $\hat{D}^{(i)} = (I_m - P_{k_i} P_{k_i}^T) D^{(i)}$.

3: Compute the QR decomposition: $\hat{D}^{(i)} = \hat{P}_{w_i} R_i$, where $\hat{P}_{w_i} \in \mathbb{R}^{m \times w_i}$, $R_i \in \mathbb{R}^{w_i \times w_i}$.

4: Compute the SVD of matrix

$$\hat{B}^{(i)} \equiv \begin{pmatrix} \Sigma_{k_i} & P_{k_i}^T D^{(i)} \\ 0 & R_i \end{pmatrix} \in \mathbb{R}^{(k_i+w_i) \times (k_i+w_i)}$$

in the form:

$$\hat{B}^{(i)} = (U_{k_i},\, U_{k_i}^{\perp}) \cdot \mathrm{diag}(\hat{\Sigma}_{k_i},\, \hat{\Sigma}_{w_i}) \cdot (V_{k_i},\, V_{k_i}^{\perp})^T \,,$$

where U_{k_i}, $V_{k_i} \in \mathbb{R}^{(k_i+w_i) \times k_i}$ and $\hat{\Sigma}_{k_i} \in \mathbb{R}^{k_i \times k_i}$.

5: *Output:* The best rank-k_i approximation of $B^{(i)} = (A_{k_i},\, D^{(i)})$ is given by:

$$B_{k_i}^{(i)} \equiv \left[(P_{k_i},\, \hat{P}_{w_i})\, U_{k_i} \right] \cdot \hat{\Sigma}_{k_i} \cdot \left[\begin{pmatrix} Q_{k_i} & 0 \\ 0 & I_{w_i} \end{pmatrix} V_{k_i} \right]^T .$$

end

Notice that Step 4 in Algorithm 1 requires the SVD of the structured matrix $\hat{B}^{(i)}$, which is upper triangular with the diagonal left upper block of order $k_i \times k_i$. At the same time, this step represents the most intensive computation in Algorithm 1.

2.1.2 Updating terms

In this case, let $T^{(i)} \in \mathbb{R}^{q_i \times n_i}$ be the q_i new term vectors that should be added to the existing terms at the bottom of the old term-document matrix. The task is to compute the best rank-k_i approximation of the row partitioned matrix

$$C^{(i)} \equiv \begin{pmatrix} A_{k_i}^{(i)} \\ T \end{pmatrix} .$$

Using steps similar to those in the previous paragraph (see [6]), one gets the Algorithm 2 for the correct updating of terms.

Algorithm 2 Algorithm for updating terms

1: *Input:* k_i, $P_{k_i} \in \mathbb{R}^{m \times k_i}$, $\Sigma_{k_i} \in \mathbb{R}^{k_i \times k_i}$, $Q_{k_i} \in \mathbb{R}^{n_i \times k_i}$, $T^{(i)} \in \mathbb{R}^{q_i \times n_i}$.

2: Compute the projection: $\hat{T}^{(i)} = (I_{n_i} - Q_{k_i}\, Q_{k_i}^T)\, T^{(i)T} \in \mathbb{R}^{n_i \times q_i}$.

3: Compute the QR decomposition: $\hat{T}^{(i)} = \hat{Q}_{q_i}\, L_i^T$, where $\hat{Q}_{q_i} \in \mathbb{R}^{n_i \times q_i}$, $L_i \in \mathbb{R}^{q_i \times q_i}$.

4: Compute the SVD of matrix

$$\hat{C}^{(i)} \equiv \begin{pmatrix} \Sigma_{k_i} & 0 \\ T^{(i)} Q_{k_i} & L_i \end{pmatrix} \in \mathbb{R}^{(k_i+q_i)\times(k_i+q_i)}$$

in the form:

$$\hat{C}^{(i)} = (U_{k_i},\, U_{k_i}^{\perp}) \cdot \mathrm{diag}(\hat{\Sigma}_{k_i},\, \hat{\Sigma}_{q_i}) \cdot (V_{k_i},\, V_{k_i}^{\perp})^T\,,$$

where U_{k_i}, $V_{k_i} \in \mathbb{R}^{(k_i+q_i)\times k_i}$ and $\hat{\Sigma}_{k_i} \in \mathbb{R}^{k_i \times k_i}$.

5: *Output:* The best rank-k_i approximation of $C^{(i)} = \begin{pmatrix} A_k^{(i)} \\ T^{(i)} \end{pmatrix}$ is given by:

$$C_{k_i}^{(i)} \equiv \left[\begin{pmatrix} P_{k_i} & 0 \\ 0 & I_{q_i} \end{pmatrix} U_{k_i} \right] \cdot \hat{\Sigma}_{k_i} \cdot \left[(Q_{k_i},\, \hat{Q}_{q_i})\, V_{k_i} \right]^T\,.$$

end

Similarly to the problem of updating documents, the computationally most intensive step is the SVD of the lower triangular matrix $\hat{C}^{(i)}$ with the upper left diagonal block.

2.2 Retrieval of documents

A retrieval of relevant documents is based on the notion of a *query*, which is the m-dimensional binary vector q with ones at positions matching the terms that should be found and retrieved from the database. Notice that the dimension of a query is equal to the number of rows of $A_{k_i}^{(i)}$, i.e. to the size of the set of terms used for coding the documents into a database.

We assume that, despite the fact that the nodes of a grid do not communicate regularly, it is possible to send the same query through the connecting network to individual nodes. These nodes are capable to receive the query and search independently for relevant documents in their individual databases $A_{k_i}^{(i)}$, $i = 1, 2, \ldots, p$. Recall, however, that the low-rank approximation is stored in each node in its factored form given by matrices $P_{k_i}\, \Sigma_{k_i}$ and Q_{k_i}. Therefore, one has to work with these matrices and not with the explicit matrix $A_{k_i}^{(i)}$, which is never computed explicitly.

The *query matching* is based on the comparison of a query vector q to the columns of the approximation $A_{k_i}^{(i)}$ by means of the acute angle $\theta_j^{(i)}$ between them; i.e., for $j = 1, 2, \ldots, n_i$, one should compute

$$\cos\theta_j^{(i)} = \frac{(A_{k_i}^{(i)} e_j)^T q}{\|A_{k_i}^{(i)} e_j\|_2 \cdot \|q\|_2} = \frac{e_j^T P_{k_i} \Sigma_{k_i} (Q_{k_i}^T q)}{\|\Sigma_{k_i} Q_{k_i}^T e_j\|_2 \cdot \|q\|_2}\,.$$

For a fixed low-dimensional approximations, this computation can be made more efficient by pre-computing n_i values

$$s_j^{(i)} = \Sigma_{k_i} V_{k_i}^T e_j.$$

Then

$$\cos\theta_j^{(i)} = \frac{s_j^{(i)T} (P_{k_i}^T q)}{\|s_j^{(i)}\|_2 \cdot \|q\|_2}\,. \tag{2}$$

Producing a list of relevant documents is based on the geometric insight about alignment of two vectors in the k_i-dimensional Euclidean space: Two vectors are the more aligned (more "identical") the less is the acute angle between them. Since cosine is the decreasing function in the interval $[0,\, \pi/2]$, this allows for the *ordering* of retrieved documents by listing a non-increasing sequence of their cosines. Usually, some sort of *thresholding* is applied for retrieved documents—retrieved are only documents for which

$$\cos\theta_j^{(i)} \geq \alpha^{(i)},$$

where $\alpha^{(i)}$ is the constant, which can be specific for each node of the grid.

2.2.1 Comparison of retrieval between individual nodes

Since the individual nodes do not communicate, one can *not* build the "global" database which would represent the compound matrix $A = (A^{(1)}, A^{(2)}, \ldots, A^{(p)})$. Form the mathematical point of view, there is no way to represent the matrix A by some low, k-dimensional approximation, because there is no way to get the individual low-dimensional approximations of matrices A_i together and build upon them the low-dimensional approximation of A.

Therefore, the retrieval of documents is possible only on a local level of individual nodes. Moreover, because the global approximation is missing, we can *not*, strictly speaking, directly compare the results from individual retrievals of documents on individual nodes. In other words, since the individual low-dimensional approximations of A_i were built independently, there is no way how to compare the accuracy of retrievals coming from two different nodes.

Nevertheless, some general conclusions can be made in a special case. For the local approximation in each node, the most important parameters are: i) the number of encoded documents n_i, and ii) the dimension of the approximation vector space k_i. If the number of encoded documents is approximately the same in each node, and if the dimensions of approximations are also the same, than one can expect the same quality of encoding—i.e., when the same terms are used in each node, the structure of individual low-dimensional vector spaces will be very similar. In this case, one can use the same threshold in each node and merge and sort p individual lists of matched documents into one list according to, for example, non-increasing cosines. In other words, we can expect approximately the same accuracy of retrieval over the nodes in this special case.

The problem is, of course, how to manage the first requirement above during the updates. If the nodes of a grid do not communicate at all, there is no way how to ensure that the local databases will be built from the approximately same amount of documents. Therefore, next discussion is devoted to the second model of a distributed system where the inter-processor communication is available.

3 LSI implemented on a distributed system

We now consider the second possible paradigm with respect to the storage of documents and their retrieval. In contrast to the grid, in a *distributed* system the individual processors can communicate and mutually exchange data. Therefore, the term-document matrix A of order $m \times n$ can be distributed column-wise among, say, p processors in the form $A = (A_1, A_2, \ldots, A_p)$ where A_i is of order $m \times n_i$. This distributed system may even correspond to one node of a grid analyzed above.

Each processor builds its own k_i-dimensional approximation of the SVD of its block A_i as described above. Notice that these computations can be computed in parallel without any communication between processors. This means that all computations are perfectly local to processors and can be realized by some serial numerical library, e.g., using the LAPACK.

After this initial computation, however, comes the main difference between a grid and a distributed system. In contrast to a grid, a distributed system *can* build another "global" approximation of the original matrix A atop of individual approximations which were computed in individual processors. This is something completely new as compared to a grid. This *global approximation* is then used in the retrieval of documents.

Next we will describe how such a global approximation can be computed

and updated from individual approximations stored in individual processors.

3.1 Building a global approximation

For the sake of simplicity of exposition, let us first consider the case of two processors, PE1 and PE2. All following derivations can be easily extended to the case of p processors with $p > 2$.

Let us assume that PE1 has computed its $m \times k_1$ approximation of A_1 and stored the corresponding partial factors of SVD P_{11}, Σ_{11} and Q_{11}. Similarly, PE2 has the local approximation P_{21}, Σ_{21} and Q_{21} of A_2 at its disposal. Notice that the first index in these local approximations can be interpreted as the processor index while the second one denotes a local level of approximation. Since the original A_i has n_i columns, we must have $k_i \leq n_i$, $i = 1, 2$.

To build a global approximation of local factors, we must first choose the order k of that approximation. For that purpose, let us organize two local approximations into a global matrix G,

$$G = (P_{11}\Sigma_{11}Q_{11}^T, P_{21}\Sigma_{21}Q_{21}^T),$$

which is the matrix of order $m \times n$ (notice that the first matrix is of order $m \times n_1$, the second one is of order $m \times n_2$ and $n = n_1 + n_2$).

To compute the k-dimensional global approximation of this matrix, we must clearly have $k \leq k_1 + k_2$. Notice that

$$(P_{11}\Sigma_{11}Q_{11}^T, P_{21}\Sigma_{21}Q_{21}^T) = (P_{11}\Sigma_{11}, P_{21}\Sigma_{21}) \begin{pmatrix} Q_{11} & 0 \\ 0 & Q_{21} \end{pmatrix}^T,$$

where the rightmost matrix is of order $(k_1 + k_2) \times n$ with orthonormal rows (after transposition).

Now the k-dimensional global approximation of G is computed in two steps by Algorithm 3:

Algorithm 3 Algorithm for global approximation

1: Compute the full SVD of the $m \times (k_1 + k_2)$ matrix

$$(P_{11}\Sigma_{11}, P_{21}\Sigma_{21}) = (\tilde{U}_k, \tilde{U}_k^{\perp}) \begin{pmatrix} \tilde{\Sigma}_k & 0 \\ 0 & \tilde{\Sigma}' \end{pmatrix} (\tilde{V}_k, \tilde{V}_k^{\perp})^T,$$

where $\tilde{\Sigma}_k$ contains k largest singular values in a non-increasing order. Here, k denotes the dimension of a global approximation, which must be chosen (but see next step).

2: Now consider the matrix product

$$\tilde{U}_k \, \tilde{\Sigma}_k \left[\begin{pmatrix} Q_{11} & 0 \\ 0 & Q_{21} \end{pmatrix} \tilde{V}_k \right]^T \equiv \tilde{U}_k \, \tilde{\Sigma}_k \tilde{W}_k^T .$$

Notice that $\tilde{W}_k$ can be computed if and only if $k = k_1 + k_2$. In this special case it has orthonormal columns so that the above matrix product *is* the truncated k-dimensional SVD of G.

end

We have just shown that in the special case, when $k = k_1 + k_2$, the global approximation can be computed quite efficiently. Notice that the formation of G requires the scaling of the local columns of left singular vectors by local singular values. This scaling can be performed in parallel without any communication between processors. Then, the SVD of a distributed matrix G has to be computed. This can be achieved using the ScaLAPACK library, or using some new parallel block-Jacobi algorithm—see [1, 2]. Finally, a distributed matrix multiplication (e.g., by the ScaLAPACK routine`PDGEMM`) has to be performed for the computation of new global right singular vectors $\tilde{W}_k$.

In the case of p processors with $p > 2$, the above matrix G consists of p blocks with n_i columns, $i = 1, 2, \ldots p$. If we choose the special value of global approximation by $k = k_1 + k_2 + \ldots k_p$, then the local approximations are again not needed in full – only the locally scaled left vectors are needed in each processor. Hence, one has to compute explicitly only the SVD of matrix G of order $m \times k$ instead of a "full" matrix of order $m \times n$. Therefore, when $k \ll n$ the substantial saving in computation time can be achieved.

At this moment, the new global approximation is available in the form of a triple $\tilde{U}_k, \tilde{\Sigma}_k, \tilde{W}_k$, whereby each matrix is distributed through p processors. We can either collect this global approximation into one (or each) processor by using the procedure `GATHER` (or `ALLGATHER`), or leave the computed global k-dimensional approximation in the distributed form. The latter approach is more advantageous from the point of view of storage requirements, since each processor stores the $m \times k_i$ submatrix of the global left singular vectors, one vector of k_i global singular values and the $n \times k_i$ submatrix of the global right singular vectors. This means that no processor has to store the complete k-dimensional factors; since $k = \sum_{k=1}^{p} k_i$, this means a substantial saving in storage space per processor.

With respect to the global approximation, the requirements for each processor are the same if $k_1 = k_2 = \cdots = k_p$. The local approximation in each processor can require different amount of storage if k_i's differ across

the processors. However, if we consider a set of processors which process the qualitatively *same* database (i.e., documents from very similar areas of interest, e.g., mathematics, physics and astronomy), then there is no reason why the local orders of approximation k_i should differ too much. This is true provided that the individual k_i-dimensional approximations of local databases in individual processors are based on the roughly *same* amount of information, i.e., the starting column dimensions n_i of local databases are roughly the same. Hence, we require that $n_i \approx n/p$ where n is the initial number of documents in the whole (huge) database, which should be distributed among p processors. To provide an initial "portion" of information equally to each processor, we can randomly choose n/p items from the initial set of documents and send them to a given processor. This starting phase ensures that local approximations can be computed with the same dimension across the processors (i.e., $k_1 = k_2 = \cdots = k_p$), and the accuracy in approximating the original local databases will be approximately the same in each processor.

3.2 Updating

When new documents are to be inserted into an existing database, it is necessary to decide which processor(s) should receive either all documents or a portion of documents. From the computational point of view, it is not advisable to add documents into database one by one. Instead, new documents should be added in a batch of, say, d items with $d \gg 1$. When necessary, a new batch of documents can be divided among, say, p_1 processors with $p_1 \leq p$, where p is the total number of processors. Let us call these p_1 processors *locally active*.

After receiving new documents, locally active processors modify (update) their local databases in parallel using Algorithm 1. Notice that no interprocessor communication is needed at this stage of computation. After finishing local updates in locally active processors, however, the update on the global level is needed. *All* processors must participate in the global update and they perform Algorithm 3. After finishing the global update, a new global, k-dimensional approximation of the distributed database is available, whereby this global approximation is itself distributed.

Updating terms is a bit different. Here, even on the local level, all processors must be active because all processors built their local, k_i-dimensional approximations over the same set of terms (vocabulary). Hence, after receiving new terms, *all* processors perform Algorithm 2 in parallel. Therefore, there are *no* inactive processors as opposed to the local updating of documents, where some processors may be idle (if they do not receive a batch of new documents). After finishing local updates of terms, all processors are

involved in the computation of a new global k-dimensional approximation by performing Algorithm 3.

3.3 Retrieval of documents

In subsection 2.2, the retrieval of documents was described for the set of non-communicating processors in a grid. In the view of our two-stage procedure for building the database of documents for the distributed LSI, we can call this approach a *local retrieval*. Since the processors in a distributed system build their local as well as global representation of a database, there is in this case also the possibility of a two-stage retrieval of documents. This possibility opens a new, interesting approach to the estimation of the quality of retrieval.

The first possibility is—as in the case of a grid—the *local* retrieval of documents. The query q is sent to all p processors and all of them go through their k_i-dimensional approximations of local databases computing the cosines according to the Eq. 2. Each processor provides its own list of relevant documents according to the algorithm described in subsection 2.2, and there is no need for the inter-processor communication in this stage of retrieval from the local databases. When assuming that the local databases were built by encoding approximately the same amount of documents in each processor, then the thresholds α_i for cosines can be chosen the same (say, 0.5) in all processors, i.e., $\alpha_i = \beta$ for all i. Individual lists can be sent to a marked processor, which can then sort all matched documents into a final list $\mathcal{L}_1$ according to the local cosines obtained in processors.

However, since also the global k-dimensional approximation has been built in the case of a distributed system, one can try also the different retrieval of documents by using the distributed factors $\tilde{U}_k$ (order $m \times k$), $\tilde{\Sigma}_k$ (order $k \times k$) and $\tilde{W}_k$ (order $n \times k$). This is the *global* retrieval of documents. It proceeds by computing the cosines

$$\cos\tilde{\theta}_j = \frac{\tilde{s}_j^T(\tilde{U}_k^T q)}{\|\tilde{s}_j\|_2 \cdot \|q\|_2}, \quad j = 1, 2, \ldots, n, \tag{3}$$

where $\tilde{s}_j$ are n pre-computed values given by

$$\tilde{s}_j = \tilde{\Sigma}_k \tilde{W}_k^T e_j. \tag{4}$$

Both above equations require the clever data organization in computing the required matrix-vector products with distributed factors and final scalar products. Let us suppose that the matrices $\tilde{U}_k$ and $\tilde{W}_k^T$ are distributed column-wise, so that processor i contains the respective blocks of dimension

$m \times k_i$ and $k \times n_i$. Then, according to Eq. (4, $\tilde{s}_j$ is computed in two steps: i) take the jth column of $\tilde{W}_k^T$ residing as as a whole in some processor, and ii) scale its ℓth component by $\tilde{\sigma}_\ell$, $\ell = 1, 2, \ldots, k$. These two steps are most easily performed locally if each processor contains all k global singular values from $\tilde{\Sigma}_k$. Then, for the computation of $\tilde{\theta}_j$ according to Eq. (3), one has: i) to compute the distributed matrix-vector product $\tilde{y} = (\tilde{U}_k^T q)$ with the query q residing in each processor, ii) to compute the distributed scalar product $\tilde{s}_j^T \tilde{y}$, and, finally, iii) to scale the scalar product by $1/(\|\tilde{s}_j\|_2 \cdot \|q\|_2)$. All these computations can be performed by appropriate functions from the ScaLAPACK library.

The global retrieval ends with sorting of the set $\{\cos\tilde{\theta}_j\}$ and thresholding them by some threshold α. Assume that the same threshold is used as in the case of the local retrieval, i.e., $\alpha = \beta$. Thus, a list of matched documents $\mathcal{L}_2$ is produced.

Now comes an interesting part of the retrieval process—the comparison of lists $\mathcal{L}_1$ and $\mathcal{L}_2$. This comparison enables to make some conclusions with respect to the *accuracy* of local and global retrieval. It is assumed that the global and local threshold for retrieving documents are the same. Furthermore, it is assumed that all retrieved documents are indeed relevant, so that one can compare both lists without caring about wrongly matched documents.

In general, the following scenarios with respect to the number of items $|\mathcal{L}_1|$ and $|\mathcal{L}_2|$ and their contents are thinkable:

1. $|\mathcal{L}_1| = |\mathcal{L}_2|$ and both lists contain the same documents. This is the ideal situation, which says that both approximations on the local as well as global level are equally accurate.

2. $|\mathcal{L}_1| = |\mathcal{L}_2|$, but the lists do *not* contain the same documents. Hence, there are at least two different documents $\mathcal{D}_1$ and $\mathcal{D}_2$ such that $\mathcal{D}_1 \in \mathcal{L}_1$, $\mathcal{D}_1 \notin \mathcal{L}_2$ and $\mathcal{D}_2 \in \mathcal{L}_2$, $\mathcal{D}_2 \notin \mathcal{L}_1$. Since we assume $k = k_1 + k_2 + \cdots + k_p$ for the dimension of global approximation, it is unlikely that this discrepancy is based on the fact of wrong dimensions in the approximations. It is more likely that some documents with low ranking in the global list will be not found in the local list because of falling just below the local threshold (and vice versa). Lowering the threshold should help to achieve the matching of the same documents on both levels (both lists can be then larger than the original ones).

3. $|\mathcal{L}_2| > |\mathcal{L}_1|$. This situation tells us that the global approximation is more accurate that the local one. This can happen, for example, when

one (or more) local databases differ substantially in their column dimensions n_i, i.e., they locally encode widely differing numbers of documents. Then the use of the same dimensionality of approximation at the local level (i.e., $k_1 = k_2 = \cdots = k_p$) leads to the *under-estimate* of optimal dimension because this value must be derived from the portion A_{n_i} having the *least* number of documents (columns). It is best to prevent such a situation by keeping the number of documents encoded in individual processors approximately the same. In other words, at the beginning, n documents should be divided evenly among processors, and, at the updating, a new batch of documents of large enough size should be processed so that, again, each processor receives approximately the same number of documents for its local update. Consequently, the new global update will be computed from locally *balanced* updates.

4. $|\mathcal{L}_1| > |\mathcal{L}_2|$. Can the retrieval at local level yield more documents than that at global level? Since $k = k_1 + k_2 + \cdots + k_p$, each locally-approximating LSI space is the subspace of the globally-approximating LSI space. Hence, all the latent couplings between terms and documents, which exist in the local databases, exist automatically also in the global database. (Notice that the reverse is not true.) Therefore, this situation should not occur in practice.

4 Conclusion

We have described two models of a distributed database of documents that are encoded using the paradigm of the LSI. The first model is devoted to the grid, when the individual nodes have a very limited possibility of mutual communication. In this case, the documents are encoded on individual processors without any connection between them. All updates of documents / terms are also performed independently as well as the retrieval of documents. If the grid consists of p nodes, the retrieval of documents for a given query yields p lists, which have no mutual relations. In particular, one can not compare the accuracy of retrieval from individual nodes. In general, it is possible to say only that the accuracy of individual nodes will be approximately the same when each node encodes approximately the same number of relevant documents (provided that the dimensions in approximation are the same in each node).

The more interesting situation arises in the case of a distributed database, when p processors are connected with some sort of inter-processor network

(this can be, for example, one node of a large grid). In this case, we have developed a two-stage compression of the latent semantic information. In the first step, the local approximation of the whole database is constructed by dividing the whole database evenly among the processors. Since the processors can communicate, the global approximation of the whole database can be built in the second step. We have shown that the SVD computation at the global level can be made very efficient when the dimension of global approximation is equal to the sum of dimensions of local approximations. We have briefly discussed the implementation issues and shown that the retrieval of documents will require the use of distributed numerical libraries like ScaLAPACK. Finally, it is now possible compare the accuracy of the locally and globally encoded database by considering the lists produced when answering the same query. We have discussed some interesting scenarios that can arise in real life.

The next step should consist of implementation of a distributed database and of some experimental work in retrieving the documents at both levels—local and global.

References

[1] M. Bečka, G. Okša and M. Vajteršic, Dynamic ordering for a parallel block-Jacobi SVD algorithm, Parallel Computing 28 (2002) 243-262.

[2] M. Bečka and G. Okša, On variable blocking factor in a parallel dynamic block-Jacobi SVD algorithm, Parallel Computing 29 (2003) 1153-1174.

[3] M. W. Berry and M. Browne, *Understanding Search Engines: Mathematical Modeling and Text Retrieval*, First ed., SIAM, Philadelphia, PA, 1999.

[4] M. W. Berry, Z. Drmač and E. R. Jessup, Matrices, vector spaces, and information retrieval, SIAM Review, 41 (1999), pp. 335–362.

[5] H. Zha, A subspace-based model for information retrieval with applications in latent semantic indexing, in Proceedings of Irregular '98, Lecture Notes in Computer Science 1457, Springer Verlag, New York, NY, 1998, pp. 29–42.

[6] H. Zha and H. D. Simon, On updating problems in latent semantic indexing, SIAM J. Sci. Comput., 21 (1999), pp. 782–791.

[7] H. Zha and Z. Zhang, On matrices with low-rank-plus-shift structure: partial SVD and latent semantic indexing, SIAM J. Matrix Anal. Appl., 21 (1999), pp. 522–536.

[8] Z. Zhang and H. Zha, Structure and perturbation analysis of truncated SVD for column-partitioned matrices, SIAM J. Matrix Anal. Appl., 22 (2001), pp. 1245–1262.