Scalable Computing:
Practice and Experience

Scientific International Journal
for Parallel and Distributed Computing

ISSN: 1895-1767

Universitatea de Vest din Timişoara

Volume 15(1) March 2014

SUBSCRIPTION INFORMATION: please visit http://www.scpe.org

Scalable Computing: Practice and Experience

Volume 15, Number 1, March 2014

TABLE OF CONTENTS

DOI 10.12694/scpe.v15i1.961
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 15, Number 1, pp. iii–iv. http://www.scpe.org

INTRODUCTION TO THE SPECIAL ISSUE ON SELECTED PAPERS FROM SYNASC 2013 WORKSHOPS

Dear SCPE readers,

It is a pleasure to present a special issue covering subjects related with high-performance computing and cloud computing. Although there are distinct research interests, approaches that are relevant for both topics could be identified. In the context of Big Data research, HPC-based approaches and algorithms are becoming increasingly relevant in the context of cloud computing. On the other hand, the access to a wider pool of computing resources, together with the expected integration of new resource types, transform the cloud into an interesting candidate for supporting some HPC-related tasks.

This special issue is based on several contributions presented at workshops that were organized in conjunction with the 15$^{\text{th}}$ *International Symposium on Symbolic and Numeric Algorithms for Scientific Computing* (SYNASC 2013). We selected four papers from the second *Workshop on Management of resources and services in Cloud and Sky computing* (MICAS 2013) and the *Workshop on HPC for Scientific Problems* (HPCSP 2013), together with a fifth paper that is highly relevant for the topics of interest that were covered by this special issue.

Challenging problems related with sparse matrices are discussed in [3, 5]. In the paper of Šimeček et al., different memory-efficient storage formats are investigated for very large sparse matrices which, due to their requirement, require strong support from massively parallel computer systems. Specific large sparse matrices are considered in their research, with a focus on the compression of the information describing the structure of these matrices [5]. This specific approach is required in order to minimize the memory footprint of stored matrices.

The results from [5] are complemented by the work of Langr et al., where, in the context of the problem of visualization of large sparse matrices emerging in HPC applications, a novel algorithm for parallel acquisition of visualization data is discussed. The algorithm was designed to cope with the heterogeneous aspects of modern HPC systems [3].

A third paper coming from the HPCSP 2013 workshop was developed with an important support from different Blue Gene/P installations, including the supercomputer at HPC Center from West University of Timişoara[1] [2]. The research from the work of Frisch et al., is built on top of the outstanding requirements coming from computational fluid mechanics computations, which are memory- and time-intensive, and require an impressive range of computational resources.

Two papers covering subjects that are relevant to cloud computing conclude this special issue. The first one is coming from the MICAS 2013 workshop, and offer a perspective on developing and modeling secure cloud applications [4], with an interesting case study on securing mOSAIC-based[2] applications. The approach used by Rak et al., is based on the currently low perception on security issues in cloud environments, which affect cloud adoption, especially by small and medium sized enterprises.

The second contribution, and last paper of this special issue, offer a different point of view in the context of cloud computing: brokering of cloud resources [1]. The multi-agent approach described by Amato and Venticinque is built in relation with the mOSAIC Cloud Agency. As different performance limitations were identified both in the case of the aforementioned solution and in the case of a centralized approach, a new set of requirements were identified and investigated in this work in order to support advances for the brokering problem.

REFERENCES

[1] A. Amato and S. Venticinque, *A distributed agent-based decision support for cloud brokering*, Scalable Computing: Practice and Experience, 15(1), pp. 65-78 (2014).

[2] J. Frisch, R.-P. Mundani, and E. Rank, *Adaptive multi-grid methods for parallel CFD applications*, Scalable Computing: Practice and Experience, 15(1), pp. 33-48 (2014).

[1] http://hpc.uvt.ro/infrastructure/bluegenep/

[2] http://www.mosaic-cloud.eu

[3] D. Langr, I. Šimeček, P. Tvrdík, and T. Dytrych, *Large-scale visualization of sparse matrices*, Scalable Computing: Practice and Experience, 15(1), pp. 21-31 (2014).
[4] M. Rak, M. Ficco, E. Battista, V. Casola, and N. Mazzocca, *Developing secure cloud applications*, Scalable Computing: Practice and Experience, 15(1), pp. 49-62 (2014).
[5] I. Šimeček, D. Langr, and P. Tvrdík, *Tree-based space efficient formats for storing the structure of sparse matrices*, Scalable Computing: Practice and Experience, 15(1), pp. 1-20 (2014).

Teodor-Florin Fortiş, *West University of Timişoara, Romania* (fortis@info.uvt.ro)

DOI 10.12694/scpe.v15i1.962
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 15, Number 1, pp. 1–20. http://www.scpe.org

TREE-BASED SPACE EFFICIENT FORMATS FOR STORING THE STRUCTURE OF SPARSE MATRICES *

I. ŠIMEČEK† D. LANGR† AND P. TVRDÍK†

Abstract.
Sparse storage formats describe a way how sparse matrices are stored in a computer memory. Extensive research has been conducted about these formats in the context of performance optimization of the sparse matrix-vector multiplication algorithms, but memory efficient formats for storing sparse matrices are still under development, since the commonly used storage formats (like COO or CSR) are not sufficient. In this paper, we propose and evaluate new storage formats for sparse matrices that minimize the space complexity of information about matrix structure. The first one is based on arithmetic coding and the second one is based on binary tree format. We compare the space complexity of common storage formats and our new formats and prove that the latter are considerably more space efficient.

Key words: sparse matrix representation; parallel execution; space efficiency; arithmetical-coding-based format; minimal binary tree format; minimal quadtree format;

AMS subject classifications. 68M14, 68W10, 68P05, 68P20, 94A17

1. Introduction. The paper investigates memory-efficient storage formats for very large sparse matrices (LSMs). By LSMs, we mean matrices that due to their sizes must be stored and processed by massively parallel computer systems (MPCSs) with distributed memory architecture consisting of tens or hundreds of thousands of processor cores.

Within our previous work [9, 12, 11, 8, 7], we have addressed weaknesses of previously developed solutions for space-efficient formats for storing of large sparse matrices. The space complexity of the representation of sparse matrices depends strongly on the used matrix storage format. A matrix of order n is considered to be *sparse* if it contains much less nonzero elements than n^2. Some alternative definitions of sparse matrix can be found in [22]. In practice, a matrix is considered sparse if the ratio of nonzero elements drops bellow some threshold. Our research addresses computations with LSMs satisfying at least one of the following conditions:

1. The LSM is used repeatedly and the computation of its elements is slow and it takes more time than its later reading from a file system.
2. Construction of a LSM is memory-intensive. It needs significant amount of memory for auxiliary data structures, typically of the same order of magnitude as the amount of memory required for storing the LSM itself.
3. A solver requires the LSM in another format than is produced by a matrix generator and the conversion between these formats cannot be performed effectively on-the-fly.
4. Computational tasks with LSMs need check-pointing and recovery from failures of the MPCSs. We assume that a distributed-memory parallel computation with a LSM needs longer time. To avoid recomputations in case of a system failure, we need to save a state of these long-run processes to allow fast recovery. This is especially important nowadays (and will be more in the future) when MPCSs consist of tens or hundreds of thousands of processor cores.

If at least one of these conditions is met, we might need to store LSMs into a file system. And since the file system access is of orders of magnitude slower compared to the memory access, we want to store matrices in a way that minimizes their memory requirements.

In this paper, we focus only on the compression of the information describing the *structure* of LSMs (i.e., the locations of nonzero elements). The values of the nonzero elements are unchanged, because their compression depends strongly on the application. For some application areas, the values of nonzero elements are implicit and only the information about the structure of a LSM is stored (for example, incident matrices of unweighed

*This research has been supported by GACR grant No. P202/12/2011.
†Department of Computer Systems, Faculty of Information Technology, Czech Technical University in Prague, Thákurova 9, 160 00, Praha, Czech Republic. E-mail: mailto:xsimecek,langrd@fit.cvut.cz.

graphs). Alternatively, we can interleave computations with reading of nonzero elements. For example, we can divide the process of a sparse matrix factorization into these steps:

1. read the matrix structure,
2. do in parallel: perform the symbolic factorization and read the values of nonzero elements of the matrix,
3. perform the numeric factorization.

This paper is an extended version of our previous results [9]. We present updated versions of the algorithms and derivation of lower and upper bounds of space and computational complexity. We also provide more detailed analysis of the computational, space, and communication complexities of parallel implementation of conversion to the new MBT format.

1.1. Terminology and notation. We consider a LSM A of order n. The number of its nonzero elements is denoted by N, the average number of nonzero elements per rows is N/n and it is denoted by *avg_per_row*.

- We assume that $1 \ll N \ll M = n^2$.
- The pattern of nonzero elements in A is unknown or random.
- Indexes of all vectors and matrices start from zero.
- The number of nonzero elements in submatrix B of matrix A is denoted by $nnz(B)$.
- Let P be the number of processors. The matrix A is partitioned among P processors $p_1, \ldots, p_P$ of a given massive parallel computer system (MPCS).
- The MPCS uses some variant of parallel I/O that allows to read/write a separate file for each process independently. Parallel I/O is a bottleneck of typical MPCS. Therefore we require that the new format should be space-efficient, in order to keep resulting file sizes as low as possible.
- We assume that nonzero elements are stored using a distributed version of a common sparse storage format (SSF). This initial distribution we called an *input mapping*.

This work is inspired by some real applications, for example ab initio calculations of medium-mass atomic nuclei (for future details see [1, 2].

1.2. Representing indexes in binary codes. Let us have an array of ξ elements indexed from 0 to $\xi - 1$. The minimum number of bits of an *unsigned* indexing data type is

$$S^{\mathrm{MIN}}(\xi) = \left\lceil \log_2 \xi \right\rceil.$$

The value S^{MIN} is the minimum number of bits, but it is usually padded to whole bytes (S^{BYTE} bits)

$$S^{\mathrm{BYTE}}(\xi) = 8 \cdot \left\lceil S^{\mathrm{MIN}}(\xi)/8 \right\rceil,$$

or it is padded to the nearest power of 2 bytes (S^{POW} bits)

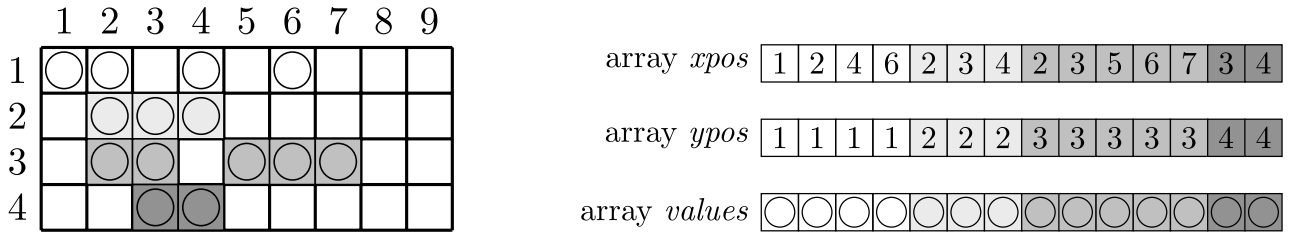
$$S^{\mathrm{POW}}(\xi) = 2^{\eta}, \quad \text{where } \eta = \left\lceil \log_2 S^{\mathrm{MIN}}(\xi) \right\rceil.$$

When we describe a matrix storage format, we use simply $S(\xi)$ instead of $S^{\mathrm{MIN}}(\xi)$.

2. State-of-the-art.

2.1. The Coordinate (COO) Format. The coordinate (COO) format is the simplest SSF (see [19, 3]). The matrix A is represented by three linear arrays *values*, *xpos*, and *ypos* (see Fig. 2.1 (b)). The array *values*$[1, \ldots, N]$ stores the nonzero values of A, arrays *xpos*$[1, \ldots, N]$ and *ypos*$[1, \ldots, N]$ contain column and row indexes, respectively, of these nonzero values. The space complexity of the structure of matrix A (the size of the array *values* is not counted) of this format is

$$S_{\mathrm{COO}}(n, N) = 2 \cdot N \cdot S(n). \tag{2.1}$$



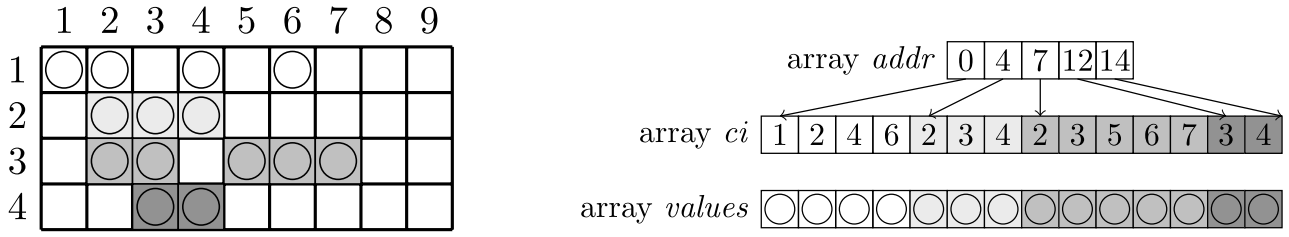
(a) An example of the sparse matrix

(b) Representation of this matrix in the COO format

FIG. 2.1. *An example of representation of sparse matrix in the COO format*

2.2. The Compressed Sparse Row (CSR) format. The most common SSF is the *compressed sparse row* (CSR) format (see [19, 3] for details). The matrix A stored in the CSR format is represented by three linear arrays *values*, *addr*, and *ci* (see Fig. 2.2 (b)). The array $values[1, \ldots, N]$ stores the nonzero elements of A, the array $addr[1, \ldots, n+1]$ contains indexes of initial nonzero elements of rows of A; if row i does not contain any nonzero element, then $addr[i] = addr[i+1]$. It is obvious that $addr[1] = 1$ and $addr[n+1] = N$. The array $ci[1, \ldots, N]$ contains column indexes of nonzero elements of A. Hence, the first nonzero element of the row j is stored at index $addr[j]$ in array *values*. The space complexity of the structure of matrix A (array *values* is not counted) in this format is

$$S_{\mathrm{CSR}}(n, N) = N \cdot S(n) + n \cdot S(N). \tag{2.2}$$



(a) An example of the sparse matrix

(b) Representation of this matrix in the CSR format

FIG. 2.2. *An example of representation of sparse matrix in the CSR format*

2.3. Register blocking formats. Widely-used SSFs are easy to understand, however, sparse operations (like matrix-vector or matrix-matrix multiplication) using these formats are slow (mainly due to indirect addressing). Sparse matrices often contain dense submatrices (blocks), so various blocking SSFs were designed to accelerate matrix operations. Compared to the CSR format, the aim of these formats (like SPARSITY [6] or [16] or CARB [20, 10]) is to allow a better use of registers and more efficient computations. But these specialized SSFs have usually large transformation overhead and consume approximately the same amount of memory as the CSR format.

2.4. Minimal quadtree (MQT) format. The *Quadtree* (QT) is a tree data structure in which all inner nodes have exactly four child nodes. Since our aim is to minimize the space complexity of QT-based formats, in [7] we proposed a new QT format called *minimal quadtree (MQT) format*. Instead of pointers, each node of the MQT contains only 4 flags (i.e., 4 bits only) indicating whether given subquadtrees are nonempty.

2.5. Other state-of-art SSFs. There are several other SSFs specialized for given areas (e.g., compression of text, picture or video). They can be used for compression of sparse matrices, but none of them satisfies all these four requirements:

1. non-lossy compression,
2. possibility of massively parallel execution,
3. space efficiency (high compression rate),
4. high speed compression/decompression.

Only few research results have been published about SSFs in the context of minimization of the required memory, which is the optimization criterion for a file I/O of LSMs. Some recent research of hierarchical blocking SSFs, though primarily aimed at optimization of matrix-vector multiplication, also addresses optimization of memory requirements [13, 14, 15]. We have published several papers about space-efficient SSFs suitable for storing sparse matrices [8, 11, 7].

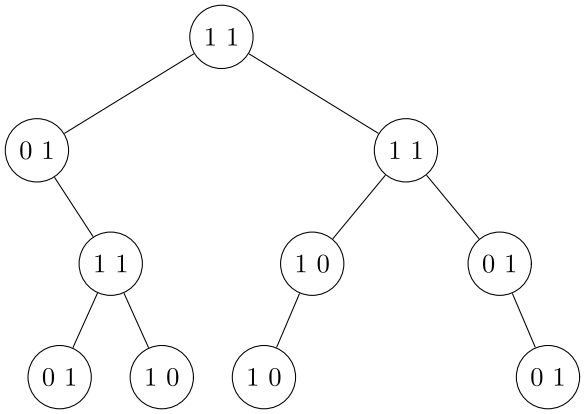


FIG. 3.1. *Visualization of the binary tree from the Example in Sect. 3.2.1.*

3. Our new space-efficient formats.

3.1. The arithmetical-coding-based (ACB) format. Matrix A can be represented by a bit vector B of size M in which N bits are set to 1 and $M - N$ bits are set to 0. The probability p_0 that a given bit in B is equal to 0 is $\frac{M-N}{M}$. In the arithmetical coding (see [21]), we can encode this information using $-\log_2 p_0$ bits. The probability p_1 that a given bit in B is equal to 1 is $\frac{N}{M}$. In the arithmetical coding, we can encode this information using $-\log_2 p_1$ bits. Since we assume a random distribution of nonzero elements, the vector B is considered to be an order-0 source (each bit is selected independently on other bits). The total number of bits to encode vector B is equal to the value of binary entropy of vector B. This value is

$$S(B) = -M \cdot (p_0 \log_2 p_0 + p_1 \log_2 p_1)$$

Since this expression is hard to compare with other formats, the approximation of the binary entropy of vector B follows:

$$\begin{aligned} S_{\mathrm{ACB}}(n, N) &= -M \cdot \left(\frac{M-N}{M} \log_2 \frac{M-N}{M} + \frac{N}{M} \log_2 \frac{N}{M}\right) \\ &= -(M-N) \log_2 \frac{M-N}{M} - N \log_2 \frac{N}{M} \\ &= -(M-N) \log_2 (M-N) + M \log_2 M - N \log_2 N. \end{aligned}$$

Since we assume that $N \ll M$, we can use the following approximation for very small x: $\ln(1+x) \approx x$. This implies that $\ln(M-N) \approx \ln M - N/M$. The final approximation of the space complexity of the ACB format

is then:

$$\begin{aligned} S_{\mathrm{ACB}}(n,N) &\approx \frac{N}{\ln 2} + N\log_2 M - \frac{N^2}{M\cdot\ln 2} - N\log_2 N \\ &\approx N\cdot\left(\frac{1}{\ln 2} + \log_2 M - \log_2 N\right) \\ &\approx N\cdot\left(\frac{1}{\ln 2} + 2\cdot\log_2 n - \log_2 N\right). \end{aligned}$$

The same space complexity (based on other assumptions) was derived in [8], but it serves only for comparison and no practical algorithm to achieve this space complexity was given. As far as we know, the ACB format has not been described in literature.

The idea of transforming of the matrix A's structure to the ACB format is simple: create $n \times n$ bitmap (with N 1's) from matrix A's structure. Then, compress this bitmap as a bitstream using the arithmetical coding. The representation of matrix A's structure in the ACB format is given by the compressed bitstream.

A comparison to common SSFs is done in Sect. 5.2. A drawback of the ACB format is its computational complexity. Since each bit of vector B is encoded in time $\Theta(1)$, the complete vector B (representation of sparse matrix A) is encoded in time $\Theta(n^2)$. This is too much for sparse matrices with a constant number of nonzero elements per row (i.e., $N \in \Theta(n)$).

3.2. The minimal binary tree (MBT) format. The *full binary tree* (FBT) is a widely used data structure in which all inner nodes have exactly two child nodes. Binary trees especially those used for binary space partitioning can also be used for storing sparse matrices. The idea of binary space partitioning is not new, but as far as we know, the use of these formats for efficiently storing sparse matrices was not described in literature. In standard implementations, every node in a FBT is represented by a structure `standard_BT_struct` consisting of the following items:

- two pointers (*left*, *right*) to child nodes,
- (only for leaves) the value of a nonzero element.

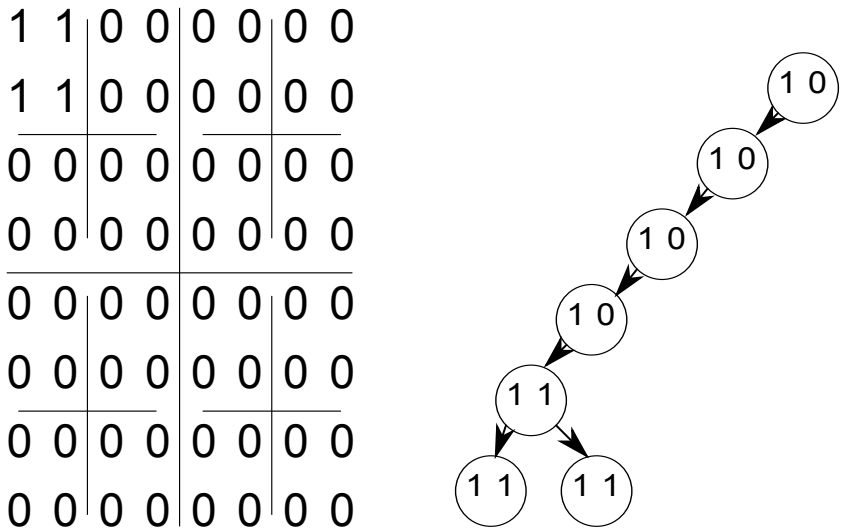
If a FBT is used as a basis for SSF, it describes a partition of the sparse matrix into submatrices and each node in the FBT represents a submatrix. Equally as in k-d trees, see [18], the decomposition is performed in alternating directions: first horizontally, then vertically, and so on. In other words, nodes in an odd height represent a partition of the submatrix into two halves along the the x-axis (left/right), nodes in an even height represent a partition of the submatrix into two halves along the y-axis (upper/lower). From the viewpoint of space efficiency, a drawback of the standard FBT representation is the overhead caused by pointers *left*, *right*. It causes that the standard FBT-based SSF may have worse space complexity than the CSR format.

To eliminate this drawback, we propose a new k-d-tree-based SSF. Each tree node represents again a submatrix, but we modify the standard representation of the FBT and we call this data structure the *minimal binary tree* (MBT) format. The idea is very similar to that in the MQT format.

- All nodes of a MBT are stored in one array (or stream). Since the size of the input matrix is given, we can compute locations of all child nodes, we can omit pointers *left*, *right*.
- All nodes of a MBT contain only two flags (it means only two bits). Each of them is set to 1 if the corresponding half of the submatrix (left/right or upper/lower) contains at least one nonzero element, otherwise it is set to 0.

A comparison of the MBT format with other SSFs is done in Sect. 5.3. Let us describe algorithm 2 that generates an output bitstream representing the matrix in the MBT format from the standard CSR format.

3.2.1. An example of a transformation to the MBT format. Let us give an example of a construction of matrix representation in the MBT format implemented as a bitstream S. The corresponding binary tree is



(a) Original sparse matrix A.

(b) The MBT representation of matrix A.

FIG. 3.2. *The MBT with the minimal number of nodes.*

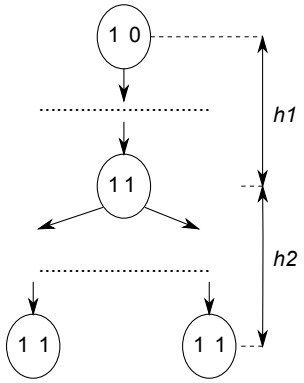
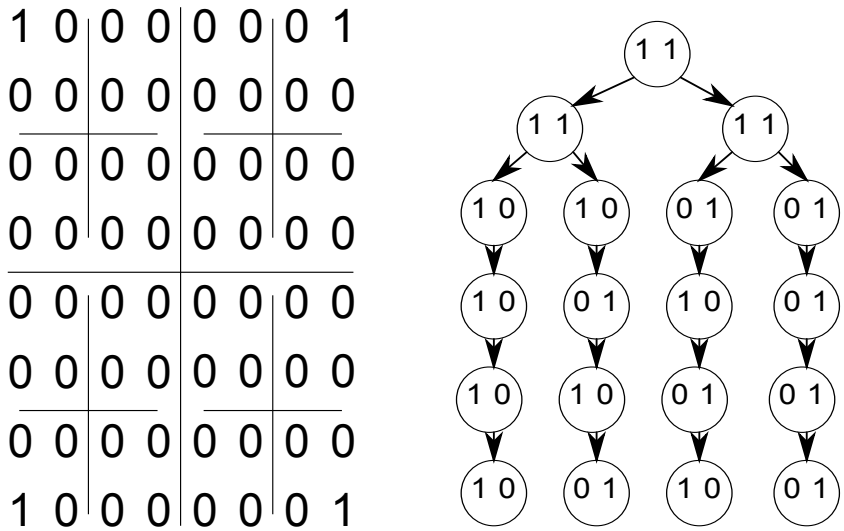


FIG. 3.3. *MBT with the minimal number of nodes (the number of leaves is $N/2$).*

shown in Fig. 3.1.

$$
\begin{aligned}
S = \mathrm{MBT}(A) &= \mathrm{MBT}\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \\
&= \text{"11"} + \mathrm{MBT}\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} + \\
&+ \mathrm{MBT}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \\
&= \text{"11"} + \text{"01"} + \text{"11"} + \mathrm{MBT}\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} + \\
&+ \mathrm{MBT}\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} + \mathrm{MBT}\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} = \\
&= \text{"11"} + \text{"01"} + \text{"11"} + \text{"11"} + \text{"10"} + \text{"01"} + \\
&+ \mathrm{MBT}(\text{"01"}) + \mathrm{MBT}(\text{"10"}) + \mathrm{MBT}(\text{"10"}) + \mathrm{MBT}(\text{"01"}) = \\
&= \text{"11"} + \text{"01"} + \text{"11"} + \text{"11"} + \text{"10"} + \\
&+ \text{"01"} + \text{"10"} + \text{"10"} + \text{"01"}
\end{aligned}
$$

So, if matrix A is stored in the MBT format, 20 bits are needed for representing its structure.



(a) Original sparse matrix A.

(b) The MBT representation of matrix A.

FIG. 3.4. *The MBT with the maximal number of nodes.*

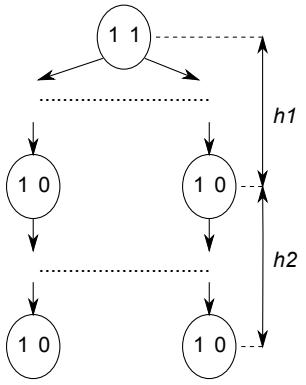


FIG. 3.5. *MBT with the maximal number of nodes (the number of leaves is $N/2$).*

3.2.2. Space complexity. Let us assume a very small example of a sparse matrix with $n = 8$ and $N = 4$. For common storage formats, the space complexity is given by Eq. (2.1) or (2.2), so $S_{\mathrm{COO}}(n, N) = 24[bits]$ and $S_{\mathrm{CSR}}(n, N) = 28[bits]$. For the MQT, the exact size of the output bitstream S (it means the size of the MBT format) cannot be derived from these global parameters, because it depends on the exact locations of nonzero elements. It ranges from 14 to 38 bits (see Figs. 3.2 and 3.4). The derivation of the lower and upper bounds on the size of the MBT format in a general case is the following.

Lower bound. We consider the best case: the MBT with the minimal number of nodes, i.e., the number of leaves is equal to $N/2$ (see Fig. 3.3). It is obviously a generalized idea from Fig. 3.2. This matrix with 4 nonzero elements is represented by 7 MBT nodes = 14 bits. Output bitstream is "10 10 10 10 11 11 11".

- The height of the MBT on Fig. 3.2 is $h = h1 + h2 = 2\log_2 n - 1$, where $h2 = \log_2 N - 1$ and $h1 = 2\log_2 n - \log_2 N$.

- All nodes with height $< h1$ (in upper $h1$ levels) contain exactly one 1 (they have one child node). The number of nodes in these levels is $h1$,

$$h1 = \log_2(n^2/N).$$

- All nodes with height $\geq h1$ (in lower $h2$ levels) are full of 1's (they have two child nodes). The number of nodes in these levels is equal to (this part is a full binary tree)

$$\sum_{i=h1+1}^{2\log_2 n-1} 2^{i-(h1+1)} = N - 1.$$

So, the minimal size of the MBT format is

$$2 \cdot \big(N - 1 + \log_2(n^2/N)\big).$$

Upper bound. We consider the worst case: the quadtree with the maximal number of nodes, i.e., the number of leaves is equal to $N/2$ (see Fig. 3.5). Again, it is a generalized idea from Fig. 3.4. This matrix with 4 nonzero elements is represented by 19 MBT nodes = 38 bits.
The output bitstream is "11 11 11 10 10 01 01 10 01 10 01 10 10 01 01 10 01 10 01".

- The height of this tree is $h = h1 + h2 = 2\log_2 n - 1$, where $h1 = \log_2 N - 1$.
- All nodes with height $< h1$ (in upper $h1$ levels) are full of 1's (they have two child nodes), $h1 = \log_2 N - 1$. The number of nodes in these levels is approximately

$$\sum_{i=0}^{h1-1} 2^i = N - 1.$$

- All nodes with height $\geq h1$ (in lower $h2$ levels) contain exactly one 1 (they have one child node). The number of nodes in these levels is

$$N \cdot h2 = N \cdot (2\log_2 n - \log_2 N) = N \cdot \log_2(n^2/N).$$

So, the maximal size of the MBT format is

$$\approx 2 \cdot N\big(1 + \log_2(n^2/N)\big).$$

3.2.3. Time complexity of the transformation algorithm. Time complexity of the transformation algorithm 2 is relatively high, because for each node in the MBT, it uses algorithm 1 with complexity $O(\log_2 n \cdot (y2 - y1 + 1))$. Fortunately, the average complexity is much lower (it depends on the value of the parameter *avg_per_row*, distribution of nonzero elements, etc.).

We consider the worst case (similar ideas as for derivation of the space complexity in Sect. 3.2.2): the MBT with the maximal number of nodes, the number of leaves is equal to N (see Fig. 3.5). We assume that the time complexity of procedure AppendToBitstream is $\Theta(1)$. Procedure `INES`(A,$x1$,$y1$,$x2$,$y2$) is called for every node in the MBT in the output stream S two times.

- For nodes with height $= h1$: The number of these nodes is N, the expression $(y2 - y1 + 1)$ is equal to $1 + n/\sqrt{N}$. Time complexity of the transformation for all nodes with this depth is $T_{h1} = N \cdot (1 + n/\sqrt{N}) \cdot \log_2 \mathit{avg\_per\_row}$.
- For nodes with height $= h1 - 1$: The number of nodes is $N/2$ and the expression $(y2 - y1 + 1)$ is equal to $1 + 2n/\sqrt{N}$. So, the total time complexity of the transformation for all nodes with depth $\leq h1$ (in upper $h1$ levels) is $T_{upper} \approx \sum_{i=0}^{h1} T_{h1}/2^{(i-h1)} = \Theta(N \cdot (1 + n/\sqrt{N}) \cdot \log_2 \mathit{avg\_per\_row})$.
- For nodes with height $> h1$: The time complexity of the transformation for all these nodes (for the lower $h2$ levels) is $T_{lower} \approx \sum_{i=h1+1}^{h} T_{h1}/2^{(i-h1)} = \Theta(N(1 + n/\sqrt{N}) \cdot \log_2 \mathit{avg\_per\_row})$.

So, the total time complexity of the transformation is

$$\Theta(N(1 + n/\sqrt{N}) \cdot \log_2 \mathit{avg\_per\_row}).$$

A very usual case is $N = \Theta(n)$, it means matrices with constant number of nonzero elements per row. For this case the time complexity is $\Theta(n^{3/2})$.

Algorithm 1 Procedure to test if the given submatrix is nonempty

```
 1: procedure INES(A,x1,y1,x2,y2)
Input: A = an input submatrix in the CSR format
Input: x1,y1,x2,y2 = coordinates of the submatrix
Output: logical value indicating whether the given submatrix is nonempty
 2:     for y ← y1, y2 do
 3:         low ← A.addr[y];   high ← A.addr[y + 1]
 4:         i ← BINARY SEARCH(in array A.ci)
 5:                                          ▷ within indexes from ⟨low ... high⟩
 6:                                          ▷ to find a minimal i such that A.ci[i] ≥ x1
 7:         if C.ci[i] ≤ x2 then
 8:             return true
 9:         end if
10:     end for
11:     return false
12: end procedure
```

3.3. Compression of minimal formats. The MBT and MQT formats have minimal space complexity only if we assume fixed number of bits for each node (2 bits for MBT and 4 bits for MQT). We can relax this assumption to achieve more space efficient formats.

LEMMA 3.1. *Every node in the MBT (or in MQT) format (except for the root node for the zero matrix A) has got at least one bit equal to 1.* The proof of Lemma 3.1 for the MBT format can be done by contradiction: if both bits in a MBT node X are zero, then this submatrix does not contain any nonzero element, so in the parent's node of X the corresponding bit is set to 0 and node X is not included in the output stream and this is a contradiction with the initial assumption.
Similar proof can be done for the MQT format. Q.E.D.

Since we assume only nonempty matrices, the only allowed values in every MBT node are: $01, 10$, and 11 (value 00 is not possible as a result of Lemma 3.1). So, if the first bit is 0, then the second bit must 1. This redundant information can be excluded from the output stream. We call this case the *hidden one*. Based on this idea, we propose a new format, called *compressed binary tree (CBT)*. There are two approaches to transform a LSM to the CBT format:

1. Transform the input matrix to the MBT format (it creates output stream S) and then remove from S all hidden ones (all 4-tuples are read and transformed values according to Table 3.1 are written.
2. Modify Algorithm 2 to Algorithm 3 that directly create the CBT format.

Similarly in the MQT format, the value 0000 is not possible as a result of Lemma 3.1, so if the first three bits are 0, then the fourth bit must 1. Again, this redundant information can be excluded from the output stream, which allows us to construct another new *compressed quadtree (CQT)* format. It is obvious that the probability of hidden one is higher in the MBT format than in the MQT format. In the Table 3.1 is the comparison of code-words in MQT, CQT, MBT, and CBT formats. If we assume the same probabilities for all possible code-words, then the average size of code-word is 4 bits in MQT format, 3.93 bits for CQT format, 5.2 bits for MBT format, and 4.47 bits for CBT format. A comparison of these formats with real matrices is done in Sect. 5.3. A transformation algorithm from the CBT format is described by Algorithm 4. This algorithm transforms the input bitstream from the CBT format into the CSR format.

Algorithm 2 Transformation algorithm to the MBT format

```
1: procedure TR2MBT(A)
Input: A = the matrix for the transformation in CSR format
Output: S = the bitstream representing the input matrix in the MBT format
2:     current ← ()
3:     enqueue {1, 1, A.n, A.n, 0} into current
4:     while current is not empty do
5:         dequeue {x1, y1, x2, y2, h} from current
6:                                          ▷ x1,y1,x2,y2 = coordinates of submatrix
7:                                          ▷ h = current BFS level, divide rows (h is odd) or columns
8:         if h is even then
9:             mx ← x2;   my ← (y1 + y2)/2
10:            lx ← x1;   ly ← (y1 + y2)/2 + 1
11:        else
12:            mx ← (x1 + x2)/2;   my ← y2
13:            lx ← (x1 + x2)/2 + 1;   ly ← y1
14:        end if
15:        l1 ← INES(A, x1, y1, mx, my)
16:        l2 ← INES(A, lx, ly, x2, y2)
17:        APPENDTOBITSTREAM(S, l1)
18:        APPENDTOBITSTREAM(S, l2)
19:        if l1 = true then
20:            enqueue {x1, y1, mx, my, h + 1} into current
21:        end if
22:        if l2 = true then
23:            enqueue {lx, ly, x2, y2, h + 1} into current
24:        end if
25:    end while
26:    return S
27: end procedure
```

3.3.1. An example of a transformation to the CBT format. For an example, we used the same matrix as in the example in Sect. 3.2.1. Hidden ones are denoted by bold numbers.

$$
\begin{aligned}
S = \mathrm{CBT}(A) &= \mathrm{CBT}\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \\
&= \text{"11"} + \mathrm{CBT}\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} + \\
&+ \mathrm{CBT}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \\
&= \text{"11"} + \text{"}\mathbf{0}\text{"} + \text{"11"} + \mathrm{CBT}\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} + \\
&+ \mathrm{CBT}\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} + \mathrm{CBT}\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} = \\
&= \text{"11"} + \text{"}\mathbf{0}\text{"} + \text{"11"} + \text{"11"} + \text{"10"} + \text{"}\mathbf{0}\text{"} + \\
&+ \mathrm{CBT}(\text{"01"}) + \mathrm{CBT}(\text{"10"}) + \mathrm{CBT}(\text{"10"}) + \mathrm{CBT}(\text{"01"}) = \\
&= \text{"11"} + \text{"}\mathbf{0}\text{"} + \text{"11"} + \text{"11"} + \text{"10"} + \\
&+ \text{"}\mathbf{0}\text{"} + \text{"}\mathbf{0}\text{"} + \text{"10"} + \text{"10"} + \text{"}\mathbf{0}\text{"}
\end{aligned}
$$

Algorithm 3 Transformation algorithm to the CBT format

```
1: procedure TR2CBT(A)
Input: A = the matrix for the transformation in CSR format
Output: S = the bitstream representing the input matrix in the CBT format
2:     current ← ()
3:     enqueue {1, 1, A.n, A.n, 0} into current
4:     while current is not empty do
5:         dequeue {x1, y1, x2, y2, h} from current
6:                                          ▷ x1,y1,x2,y2 = coordinates of submatrix
7:                                  ▷ h = current BFS level, divide rows (h is odd) or columns
8:         if h is even then
9:             mx ← x2;   my ← (y1 + y2)/2
10:            lx ← x1;   ly ← (y1 + y2)/2 + 1
11:        else
12:            mx ← (x1 + x2)/2;   my ← y2
13:            lx ← (x1 + x2)/2 + 1;   ly ← y1
14:        end if
15:        l2 ← false
16:        l1 ← INES(A, x1, y1, mx, my)
17:        APPENDTOBITSTREAM(S, l1)
18:        if l1 = true then
19:            l2 ← INES(A, lx, ly, x2, y2)
20:            APPENDTOBITSTREAM(S, l2)
21:        end if
22:        if l1 = true then
23:            enqueue {x1, y1, mx, my, h + 1} into current
24:        end if
25:        if l2 = true then
26:            enqueue {lx, ly, x2, y2, h + 1} into current
27:        end if
28:    end while
29:    return S
30: end procedure
```

For storing matrix A in the CBT format only 16 bits are needed (instead of 20 bits in the MBT format).

MQT	CQT	MBT	CBT
0000	N/A	00	N/A
0001	**000**	01 01	**0 0**
0010	0010	01 10	**0** 10
0011	0011	01 11	**0** 11
0100	0100	10 01	10 **0**
0101	0101	11 01 01	11 **0 0**
0110	0110	11 01 10	11 **0** 10
0111	0111	11 01 11	11 **0** 11
1000	1000	10 10	10 10
1001	1001	11 10 01	11 10 **0**
1010	1010	11 10 10	11 10 10
1011	1011	11 10 11	11 10 11
1100	1100	10 11	10 11
1101	1101	11 11 01	11 11 **0**
1110	1110	11 11 10	11 11 10
1111	1111	11 11 11	11 11 11

TABLE 3.1

Transformation table between the MQT, CQT, MBT, and CBT formats. Hidden ones are marked by bold numbers.

Algorithm 4 Transformation from the CBT format to the CSR format

```
 1: procedure Tr2CSR(S)
Input: S = the input bitstream of the input matrix in the CBT format
Output: A = the output matrix in the CSR format
 2:     A ← new empty matrix
 3:     enqueue {1, 1, A.n, A.n, 0} into current
 4:     while current is not empty do
 5:         dequeue {x1, y1, x2, y2, h} from current
 6:                                          ▷ x1,y1,x2,y2 = coordinates of submatrix
 7:                                          ▷ h = current BFS level, divide rows (h is odd) or columns
 8:         if x1 = x2 & y1 = y2 then
 9:             X ← new nonzero element (x1, y1)
10:             add X to A
11:         else
12:             if h is even then
13:                 mx ← x2;   my ← (y1 + y2)/2
14:                 lx ← x1;   ly ← (y1 + y2)/2 + 1
15:             else
16:                 mx ← (x1 + x2)/2;   my ← y2
17:                 lx ← (x1 + x2)/2 + 1;   ly ← y1
18:             end if
19:             l1 ← ReadOneBit(S)
20:             if l1 = false then
21:                 l2 ← true                                   ▷ hidden one
22:             else
23:                 l2 ← ReadOneBit(S)
24:             end if
25:             if l1 = true then
26:                 enqueue {x1, y1, mx, my, h + 1} into current
27:             end if
28:             if l2 = true then
29:                 enqueue {lx, ly, x2, y2, h + 1} into current
30:             end if
31:         end if
32:     end while
33:     return A
34: end procedure
```

4. Parallel execeution of transformation algorithm.

4.1. Main idea of parallelization. The proposed formats are generic, i.e., they may be applied to sparse matrices of any structure. When processing LSMs on a massively parallel computer system, every processor has its own part of a matrix, which itself can be treated as a stand-alone matrix of a smaller size. Every processor can apply one of the proposed formats to its own matrix independently. Hence, the proposed formats can be utilized on massively parallel computer systems the very same way as in sequential computations. This approach to parallelization is straightforward for the ACB format, but for SSFs based on trees the parallelization is more complicated.

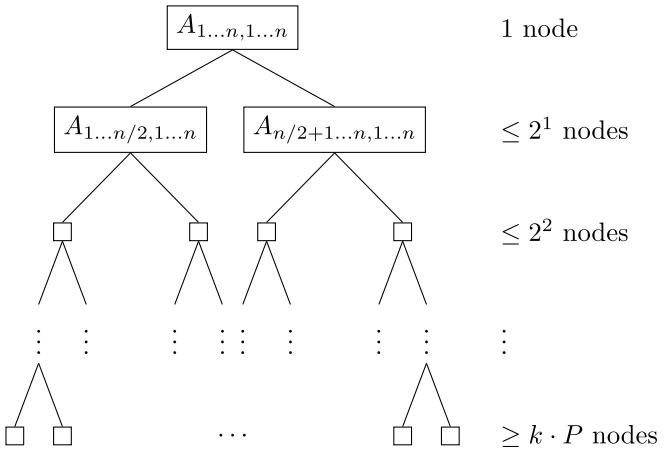


FIG. 4.1. *The main idea of parallelization of MBT or CBT transformation.*

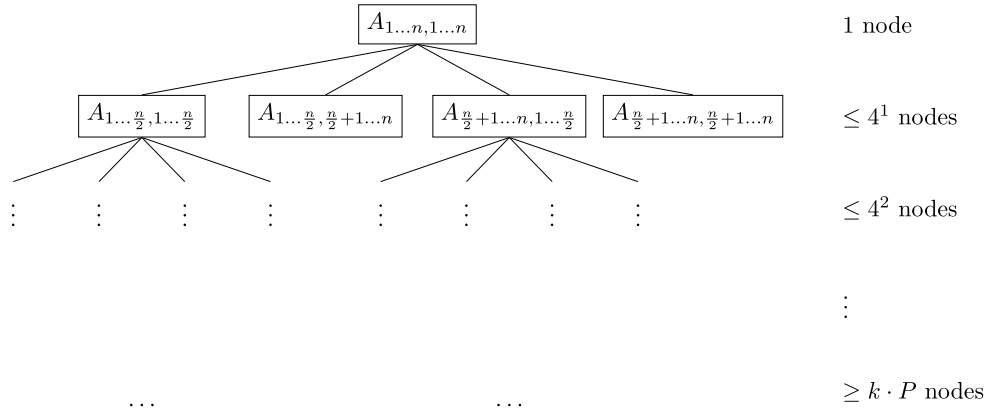


FIG. 4.2. *The main idea of parallelization of MQT or CQT transformation.*

4.2. Parallel transformation of formats based on trees. Consider the MBT format. The proposed Algorithm 2 for transformation is sequential. Let us now describe its master-slave parallelization. We assume two possible mappings how the matrix A is distributed among processors:

- using general mapping.
- using row-wise 1D block mapping (see [17]). Matrix A is divided into P row blocks of variable size (recall P is the number of processors in a MPCS. This mapping uses array *start_row*. In this array, the value *start_row*$[i]$ is the starting row for the row block assigned to processor p_i.

The only difference is that for general mapping we must use a general (unoptimized) procedure PARINES (see Algorithm 5) and for the row-wise 1D block mapping we can use a optimized procedure PARINES2 (see

Algorithm 6).

1. In the first step (Algorithm 8 and codelines 1-10 in Algorithm 7), only processor p_1 (similarly to Algorithm 2) expands nonempty nodes of a binary tree by BFS until the number of nodes (denoted by C) is greater or equal to $k \cdot P$, where k is the chosen constant (see Figs. 4.1 and 4.2, and Algorithm 7). The proper value of parameter k is the trade-off between better work-load balancing (higher values of k) and smaller sequential part of transformation (lower values of k).
   This binary tree defines partitioning of the matrix among processors. This tree (more exactly intervals of coordinates of nodes) is stored into array B and also stored in the MBT format in a special master file.
2. Initial communication (codeline 11 in Algorithm 7): Processor p_1 sends to all other processor blocks of array B using one-to-all-scatter operation. The block for p_i starts at index $1 + (i-1)\lceil C/P \rceil$ and ends at $\min(i\lceil C/P \rceil, C)$. Each block contains intervals of coordinates of submatrices that are assigned to the given processor (see Algorithm 6).
3. Redistribution of nonzero elements (codeline 12 in Algorithm 7): nonzero elements of the matrix A are redistributed between processors according to the resulting partitioning (array B).
4. Local transformation (codelines 13-20 in Algorithm 7): Every processor transforms assigned submatrices to the required MBT format independently and stores them into a separate file.

Algorithm 5 Distributed procedure to test if the given submatrix is nonempty

```
 1: procedure ParINES(A,x1,y1,x2,y2)
Input: A = the input matrix in the distributed CSR format
Input: x1,y1,x2,y2 = coordinates of submatrix
Output: logical value indicating whether the given submatrix is nonempty
 2:     one-to-all broadcast {x1, y1, x2, y2}
 3:     output = false
 4:     for j ← y1, y2 do
 5:         for i ← A.Addr[j], A.Addr[j + 1] − 1 do
 6:             if (x1 ≥ A.Ci[i] ≤ x2) then
 7:                 output = true
 8:                 break
 9:             end if
10:         end for
11:     end for
12:     send predicate output to parallel reduction
13:     po ← parallel reduction of output using logical OR
14:     return po
15: end procedure
```

Matrix	n	N	*avg_per_row*
circuitM5	$5.56 \cdot 10^6$	$5.95 \cdot 10^7$	$1.93 \cdot 10^{-6}$
nlpkkt120	$3.54 \cdot 10^6$	$5.02 \cdot 10^7$	$4.01 \cdot 10^{-6}$
ldoor	$9.52 \cdot 10^5$	$2.37 \cdot 10^7$	$2.60 \cdot 10^{-5}$
TSOPF_RS_b2383	$3.81 \cdot 10^4$	$1.62 \cdot 10^7$	$1.10 \cdot 10^{-2}$
mouse_gene	$4.51 \cdot 10^4$	$1.45 \cdot 10^7$	$7.10 \cdot 10^{-3}$
t2em	$9.25 \cdot 10^5$	$4.59 \cdot 10^6$	$5.36 \cdot 10^{-6}$
bmw7st_1	$1.41 \cdot 10^5$	$3.74 \cdot 10^6$	$1.88 \cdot 10^{-4}$
amazon0312	$4.01 \cdot 10^5$	$3.20 \cdot 10^6$	$2.00 \cdot 10^{-6}$
thread	$2.97 \cdot 10^4$	$2.25 \cdot 10^6$	$2.55 \cdot 10^{-3}$
gupta2	$6.21 \cdot 10^4$	$2.16 \cdot 10^6$	$5.60 \cdot 10^{-4}$
c-29	$5.03 \cdot 10^3$	$2.44 \cdot 10^4$	$9.64 \cdot 10^{-4}$

Table 4.1
Characteristics of the testing matrices.

Algorithm 6 Distributed procedure to test if the given submatrix is nonempty

```
 1: procedure ParINES2(A,x1,y1,x2,y2)
Input: A = the input matrix in the distributed CSR format
Input: x1,y1,x2,y2 = coordinates of submatrix
Output: logical value indicating whether the given submatrix is nonempty
 2:     construct G'
 3:     multicast {x1, y1, x2, y2} in G'
 4:     i ← index of the current processor
 5:     si ← start_row[i]
 6:     si1 ← start_row[i + 1]
 7:     if (si > y2) OR (si1 ≤ y1) then
 8:         output = false
 9:     else
10:         for y ← max(y1, si), min(y2, si1 − 1) do
11:             low ← A.addr[y];   high ← A.addr[y + 1]
12:             i ← BINARY SEARCH(in array A.ci)
13:                                          ▷ within indexes from ⟨low ... high⟩
14:                                          ▷ to find a minimal i such that A.ci[i] ≥ x1
15:             if C.ci[i] ≤ x2 then
16:                 output = true
17:                 break
18:             end if
19:         end for
20:         output = false
21:     end if
22:     send predicate output to parallel reduction
23:     po ← parallel reduction of output using logical OR in G'
24:     return po
25: end procedure
```

5. Results of space efficient formats.

5.1. Testing matrices. We have used 11 testing matrices from various application domains from the University of Florida Sparse Matrix Collection [5]. Table 4.1 shows the characteristics of the testing matrices. For quantification of the reduction in data-representation size produced by a data format, we have used ratio of space complexities. For comparison of the results, we have used these common formats:

- the COO format,
- the CSR format,
- the text-based Matrix Market format [4],
- the zipped Matrix Market format (we have used the PKZIP program with the option for maximal compression).

For our purposes, we have excluded all temporary informations from the source Matrix Market files (like comments and values of nonzero values).

5.2. Comparison of space complexities of common and ACB SSFs. Table 5.1 illustrates the fact that the space complexity of storing the structure of these sparse matrices using common storage formats (COO and CSR) are significantly greater than in the ACB format (independently on the padding). We can conclude that the common SSFs (COO, CSR) are not suitable for our purposes.

5.3. Results for the tree-based formats. The ACB format is space optimal, but only if the distribution of nonzero elements is random (i.e., without any locality in the matrix). Due to this fact, we use this format as the reference format. Table 5.2 compares the ratios of the matrix space complexities in the MBT format w.r.t.

Algorithm 7 Parallel transformation algorithm to the MBT format

1: **procedure** ParTr2MBT(A)
Input: A = the input matrix in the distributed CSR format
Output: S = the local bitstream of the resulting MBT format
2: $i \leftarrow$ index of the current processor
3: **if** $i = 1$ **then** ▷ master section
4: $S \leftarrow ()$
5: enqueue $\{A, 1, 1, A.n, A.n, 0\}$ into *current*
6: **while** $|current| < k \cdot P$ **do**
7: $current \leftarrow$ ExpandLevel(S, *current*)
8: **end while**
9: store S in master file
10: convert *current* to array B
11: **end if**
12: barrier
13: one-to-all scatter of B
14: all-to-all scatter of matrix structure
15: $S \leftarrow ()$
16: $C \leftarrow |current|$
17: **for** $j \leftarrow 1 + (i-1)\lceil C/P \rceil, \min(i\lceil C/P \rceil, C)$ **do**
18: $\{x1, y1, x2, y2, h\} \leftarrow B[j]$
19: $D \leftarrow A[y1 \ldots y2][x1 \ldots x2]$
20: Tr2MBT($D, x1, y1, x2, y2, h$)
21: **end for**
22: store S in separate file dedicated to p_i
23: **return**
24: **end procedure**

Matrix	COO S^{MIN}	COO S^{POW}	CSR S^{MIN}	CSR S^{POW}
circuitM5	2.25	3.13	1.24	1.71
nlpkkt120	2.27	3.30	1.23	1.77
ldoor	2.40	3.84	1.26	2.00
TSOPF_RS_b2383	4.04	4.04	2.03	2.03
mouse_gene	3.73	3.73	1.87	1.88
t2em	2.11	3.38	1.30	2.03
bmw7st_1	2.61	4.63	1.36	2.40
amazon0312	2.23	3.75	1.28	2.11
thread	2.98	3.18	1.52	1.63
gupta2	2.61	2.61	1.36	1.38
c-29	2.27	2.79	1.40	1.68

Table 5.1
The ratio of the space complexities of matrices in the COO or CSR formats using different paddings and in the ACB format.

other storage schemes. CR stands for *compression rate*. CR1 denotes the ratio of the MBT to the (CSR, S^{POW}) format space complexities. CR2 denotes the ratio of the MBT to the ACB format space complexities. CR3 denotes the ratio of the MBT format space complexity to space complexity of the text based Matrix Market format. CR4 denotes the ratio of the MBT format space complexities to the zipped Matrix Market format space complexity.

Table 5.3 shows ratios of space complexities of the four tree-based formats studied in this paper to the ACB format. From this table, we can observe that the CBT format:

- has usually smaller space complexity than the ACB format. There was only one exception among the 11 testing matrices: (mouse_gene).

Algorithm 8 Master section of parallel transformation algorithm to the MBT format

```
1: procedure ExpandLevel(S, current)
2:     create empty queue new
3:     while current is nonempty do
4:         dequeue {x1, y1, x2, y2, h} from current
5:         if h is even then
6:             mx ← x2;   my ← (y1 + y2)/2
7:             lx ← x1;   ly ← (y1 + y2)/2 + 1
8:         else
9:             mx ← (x1 + x2)/2;   my ← y2
10:            lx ← (x1 + x2)/2 + 1;   ly ← y1
11:        end if
12:        l1 ← ParINES(A, x1, y1, mx, my)
13:        l2 ← ParINES(A, lx, ly, x2, y2)
14:        AppendToBitstream(S, l1)
15:        AppendToBitstream(S, l2)
16:        if l1 = true then
17:            enqueue {A, x1, y1, mx, my, h + 1} into next
18:        end if
19:        if l2 = true then
20:            enqueue {A, lx, ly, x2, y2, h + 1} into next
21:        end if
22:    end while
23:    return next
24: end procedure
```

Matrix	CR1 [%]	CR2 [%]	CR3 [%]	CR4 [%]
circuitM5	16.5	28.3	4.9	28.8
nlpkkt120	12.8	22.6	3.5	25.6
ldoor	8	16.1	2.4	15.3
TSOPF_RS_b2383	15.6	31.5	2.7	14.0
mouse_gene	73.0	137.0	12.8	53.9
t2em	16.3	33.0	5.7	26.2
bmw7st_1	8.5	20.3	2.8	15.7
amazon0312	53.1	112.1	18.1	67.7
thread	16.4	26.8	3.0	14.4
gupta2	28.7	39.7	5.3	23.3
c-29	31.7	53.4	8.6	27.2

Table 5.2
Comparison of the space complexity of the MBT format with that of other storage schemes.

- has similar space complexity as the MQT or CQT formats.

We can conclude that the CBT format is very space efficient.

5.4. Results for parallelization of the tree-based formats. The most time-consuming part of master section of ParTr2MBT procedure (parallel implementation of conversion to the MBT format) is the blocking call of ParINES (or ParINES2) procedure. To achieve good scalability of this code, the overhead of these calls should be small in comparison to the global parameters (n and N), because these parameters influence the time complexity of parallel section (local transformation) of ParTr2MBT procedure (see Sect. 3.2.3).

Table 5.4 shows the number of calls of ParINES in comparison to global parameters. From this table, we can observe that these numbers of calls are relatively close to the parameter kP, so the parallel conversion to the MBT format is reasonable (and the value of parameter k is suitable) if $kP \ll n$.

Matrix	MBT [%]	CBT [%]	MQT [%]	CQT [%]
circuitM5	27,4	24,4	26,6	26,5
nlpkkt120	26,1	23,5	19,8	19,8
ldoor	22,5	21,2	14,4	14,3
TSOPF_RS_b2383	36,1	35,9	33,6	33,5
mouse_gene	130,8	111,0	130,4	128,0
t2em	36,6	31,7	29,9	29,4
bmw7st_1	26,3	25,0	20,3	20,3
amazon0312	110,8	89,1	107,1	103,1
thread	37,3	35,3	23,8	23,8
gupta2	48,0	42,3	36,3	36,0
c-29	55,7	48,8	51,3	50,9

TABLE 5.3
Comparison of the space complexity of the tree-based SSFs with that of the ACB format.

Matrix	n	N	#calls ($kP = 10^1$)	#calls ($kP = 10^2$)	#calls ($kP = 10^3$)
circuitM5	$5.56 \cdot 10^6$	$5.95 \cdot 10^7$	20	278	4248
nlpkkt120	$3.54 \cdot 10^6$	$5.02 \cdot 10^7$	26	452	3852
ldoor	$9.52 \cdot 10^5$	$2.37 \cdot 10^7$	26	240	2204
TSOPF_RS_b2383	$3.81 \cdot 10^4$	$1.62 \cdot 10^7$	20	378	3210
mouse_gene	$4.51 \cdot 10^4$	$1.45 \cdot 10^7$	22	220	2060
t2em	$9.25 \cdot 10^5$	$4.59 \cdot 10^6$	26	426	4906
bmw7st_1	$1.41 \cdot 10^5$	$3.74 \cdot 10^6$	22	232	3368
amazon0312	$4.01 \cdot 10^5$	$3.20 \cdot 10^6$	18	200	2072
thread	$2.97 \cdot 10^4$	$2.25 \cdot 10^6$	26	260	3486
gupta2	$6.21 \cdot 10^4$	$2.16 \cdot 10^6$	28	380	3888
c-29	$5.03 \cdot 10^3$	$2.44 \cdot 10^4$	26	348	3900

TABLE 5.4
The efficiency of parallel algorithm (the number of calls of ParINES*).*

6. Conclusions. This paper deals with the design of four new SSFs called arithmetical coding based format, minimal binary tree format, compressed binary tree format, and compressed quadtree format. These formats have been designed in order to minimize the space complexity. We performed experiments with these formats and compared them with other common SSFs (COO or CSR) and other schemes used for LSMs in a file. These experiments proved that our new formats can significantly reduce the amount of data needed for storing LSMs. We have also presented a parallel algorithm for transformation of a LSM in the CSR format to one of these newly proposed formats.

REFERENCES

[1] T. Dytrych, K. D. Launey, J. P. Draayer, P. Maris, J. P. Vary, E. Saule, U. Catalyurek, M. Sosonkina, D. Langr, and M. A. Caprio, *Collective Modes in Light Nuclei from First Principles*, PHYSICAL REVIEW LETTERS, 111 (2013).

[2] K. D. Launey, S. Sarbadhicary, T. Dytrych, and J. P. Draayer, *Program in C for studying characteristic properties of two-body interactions in the framework of spectral distribution theory*, COMPUTER PHYSICS COMMUNICATIONS, 185 (2014), pp. 254–267.

[3] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. V. der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, PA, 2nd ed., 1994.

[4] R. F. Boisvert, R. Pozo, and K. Remington, *The Matrix Market Exchange Formats: Initial Design*, Tech. Report NISTIR 5935, National Institute of Standards and Technology, Dec. 1996.

[5] T. A. Davis, *The university of florida sparse matrix collection*, NA DIGEST, 92 (1994).

[6] E. Im, *Optimizing the Performance of Sparse Matrix-Vector Multiplication - dissertation thesis*, Dissertation thesis, University of Carolina at Berkeley, 2001.

[7] I. Šimeček, D. Langr, and P. Tvrdik, *Minimal quadtree format for compression of sparse matrices storage*, in 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'2012), SYNASC'2012, Timisoara, Romania, sept. 2012, pp. 359–364.

[8] I. Šimeček, D. Langr, and P. Tvrdík, *Space-efficient sparse matrix storage formats for massively parallel systems*, in High Performance Computing and Communication and 2012 IEEE 9th International Conference on Embedded Software and

Systems (HPCC-ICESS), HPCC'12, Liverpool, Great Britain, june 2012, pp. 54–60.

[9] I. Šimeček, D. Langr, and P. Tvrdík, *Space efficient formats for structure of sparse matrices based on tree structures*, in Proceedings of 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2013), SYNASC '13, 2013. to be published.

[10] I. Šimeček and P. Tvrdík, *Sparse matrix-vector multiplication - final solution?*, in Parallel Processing and Applied Mathematics, vol. 4967 of PPAM'07, Berlin, Heidelberg, 2008, Springer-Verlag, pp. 156–165.

[11] D. Langr, I. Šimeček, P. Tvrdík, T. Dytrych, and J. P. Draayer, *Adaptive-blocking hierarchical storage format for sparse matrices*, in Federated Conference on Computer Science and Information Systems (FedCSIS), 345 E 47TH ST, NEW YORK, NY 10017 USA, September 2012, IEEE Xplore Digital Library, pp. 545–551.

[12] D. Langr, I. Šimeček, and P. Tvrdík, *Storing Sparse Matrices in the Adaptive-Blocking Hierarchical Storage Format*, in Federated Conference on Computer Science and Information Systems (FedCSIS), September 2013, IEEE Xplore Digital Library, pp. 479–486.

[13] M. Martone, S. Filippone, M. Paprzycki, and S. Tucci, *On the usage of 16 bit indices in recursively stored sparse matrices*, in Proceedings of the 2010 12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC '10, Washington, DC, USA, 2010, IEEE Computer Society, pp. 57–64.

[14] M. Martone, S. Filippone, S. Tucci, P. Gepner, and M. Paprzycki, *Use of hybrid recursive csr/coo data structures in sparse matrix-vector multiplication*, in Computer Science and Information Technology (IMCSIT), Proceedings of the 2010 International Multiconference on, Oct 2010, pp. 327–335.

[15] M. Martone, S. Filippone, S. Tucci, and M. Paprzycki, *Assembling recursively stored sparse matrices*, in Computer Science and Information Technology (IMCSIT), Proceedings of the 2010 International Multiconference on, Oct 2010, pp. 317–325.

[16] M. Martone, M. Paprzycki, and S. Filippone, *An improved sparse matrix-vector multiply based on recursive sparse blocks layout*, in Large-Scale Scientific Computing, vol. 7116 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2012, pp. 606–613.

[17] A. Pinar and C. Aykanat, *Sparse matrix decomposition with optimal load balancing*, in Proceedings of the Fourth International Conference on High-Performance Computing, HIPC '97, Washington, DC, USA, 1997, IEEE Computer Society, pp. 224–.

[18] L. Romero and E. Zapata, *Data distributions for sparse matrix vector multiplication*, Parallel Computing, 21 (1995), pp. 583 – 605.

[19] Y. Saad, *Iterative Methods for Sparse Linear Systems*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd ed., 2003.

[20] P. Tvrdík and I. Šimeček, *A new diagonal blocking format and model of cache behavior for sparse matrices*, in Proceedings of the 6th International Conference on Parallel Processing and Applied Mathematics, vol. 12 of PPAM'05, Poznan, Poland, 2005, Springer-Verlag, pp. 164–171.

[21] I. H. Witten, R. M. Neal, and J. G. Cleary, *Arithmetic coding for data compression*, Commun. ACM, 30 (1987), pp. 520–540.

[22] M. Tuma, *Overview of direct methods*, I. Winter School of SEMINAR ON NUMERICAL ANALYSIS, January 2004, Ostrava, Czech Republic.

Edited by: Teodor Florin Fortiş
Received: Mar 3, 2014
Accepted: Apr 4, 2014

DOI 10.12694/scpe.v15i1.963
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 15, Number 1, pp. 21–31. http://www.scpe.org

LARGE-SCALE VISUALIZATION OF SPARSE MATRICES*

D. LANGR† I. ŠIMEČEK† P. TVRDÍK† AND T. DYTRYCH‡

Abstract.
An efficient algorithm for parallel acquisition of visualization data for large sparse matrices is presented and evaluated both analytically and empirically. The algorithm was designed to be application-independent, i.e., it works with any matrix-processors mapping and with any sparse storage format/scheme. The empirical scalability study of the algorithm was carried on using multiple modern HPC systems. In our largest experiment, we utilized 262144 processors for 73 seconds to gather and store to a file the visualization data for a matrix with $1.17 \cdot 10^{13}$ nonzero elements. Using the proposed algorithm, one can thus visualize large sparse matrices with a minimal runtime overhead imposed on executed HPC codes.

Key words: visualization, sparse matrices, parallel system, distributed algorithm, data acquisition

AMS subject classifications. 65F30, 65F50, 68W15

1. Introduction. Within our previous work, we have addressed weaknesses of common solutions for the problem of visualization of large sparse matrices emerging in HPC applications [9]. There are several reasons that make such a problem difficult to solve. First, matrices exist in memory only for a short and unknown period of time determined by the scheduler of a given HPC system. Second, it is generally impossible to integrate the visualization process directly into the HPC code so that it will automatically produce a final matrix image of a desired quality. Third, very large matrices, due to their sizes, cannot be processed locally on personal computers. Moreover, their storage to a file on an HPC system and their transfer via network would take high amount of time.

We therefore proposed a solution where a matrix is first partitioned into blocks. Then, for each block, visualization information is calculated and stored into a file. Finally, this file is downloaded from the HPC system into a personal computer and processed there, possibly interactively, into a final matrix image. We introduced an algorithm that accomplishes a part of this procedure performed on the side of an HPC system, i.e., the acquisition of visualization data for a given sparse matrix. We also evaluated this algorithm experimentally using up to 1024 processors of a small-scale HPC system.

This paper is an extended version of our previous results [9]. We present an updated version of the algorithm, which uses more efficient approach to the calculation of visualization data. Namely, due to the utilization of an *ordered* associative array, frequent lookup operations with logarithmic runtime complexity were substituted by a single iteration over the ordered records. Each required record is thus available with constant runtime complexity. We also provide more detailed analysis of the computational, space, and communication complexities of the algorithm. The results of large-scale experiments using several modern HPC systems are presented for up to 266144 utilized processors.

Modern HPC systems are typically hybrid shared-distributed memory machines. They consist of shared-memory multicore computational nodes connected by network subsystems. Parallel programming models and corresponding runtime environments allow to use these machines as are, virtualize them as pure shared-memory, or virtualize them as pure distributed-memory. In the context of this paper, we consider the latter case.

*This work was supported by the Czech Science Foundation under Grant No. P202/12/2011, and by the Czech Technical University in Prague under Grant No. SGS14/106/OHK3/1T/18. This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (award number OCI 07-25070) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications. This research used resources of the National Energy Research Scientific Computing Center (NERSC), which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. This work was supported by the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070), funded by the European Regional Development Fund and the national budget of the Czech Republic via the Research and Development for Innovations Operational Programme, as well as Czech Ministry of Education, Youth and Sports via the project Large Research, Development and Innovations Infrastructures (LM2011033).

†Department of Computer Systems, Faculty of Information Technology, Czech Technical University in Prague, Thákurova 9, 160 00, Praha, Czech Republic. E-mail: mailto:langrd@fit.cvut.czlangrd@fit.cvut.cz.

‡Department of Physics and Astronomy, Louisiana State University, Baton Rouge, LA 70803, USA.

Particularly, we assume the utilization of the MPI parallel programming library and its runtime environment [5, 11], which is motivated by its widespread adoption in the HPC community.

We further call MPI processes involved in algorithm runs simply *processors*. However, we always assume that each MPI process is mapped at runtime to a single CPU core. Therefore, a processor may also refer to a CPU core on which the algorithm runs.

2. Terminology and Notation. Let $A = (a_{i,j})$ be an $m_A \times n_A$ real or complex matrix. Let $B = (b_{i,j})$ be an $m_B \times n_B$ real matrix, where $m_B \leq m_A$ and

$$n_B = \left\lfloor m_B \cdot \frac{n_A}{m_A} + \frac{1}{2} \right\rfloor. \tag{2.1}$$

We call B the *visualization matrix*.

Let us partition A into $m_B \times n_B$ blocks (submatrices) of same or nearly same sizes as follows:

$$A = \begin{bmatrix} A_{1,1} & \cdots & A_{1,n_B} \\ \vdots & \ddots & \vdots \\ A_{m_B,1} & \cdots & A_{m_B,n_B} \end{bmatrix}, \quad A_{k,l} = \begin{bmatrix} a_{r_k,c_l} & \cdots & a_{r_k,c'_l} \\ \vdots & \ddots & \vdots \\ a_{r'_k,c_l} & \cdots & a_{r'_k,c'_l} \end{bmatrix}$$

where

$$r_k = \left\lceil (k-1) \cdot \frac{m_A}{m_B} + \frac{1}{2} \right\rceil, \qquad r'_k = \left\lfloor k \cdot \frac{m_A}{m_B} - \frac{1}{2} \right\rfloor, \tag{2.2a}$$

$$c_l = \left\lceil (l-1) \cdot \frac{n_A}{n_B} + \frac{1}{2} \right\rceil, \qquad c'_l = \left\lfloor l \cdot \frac{n_A}{n_B} - \frac{1}{2} \right\rfloor. \tag{2.2b}$$

We will further write simply $a_{i,j} \in A_{k,l}$ to indicate that the element $a_{i,j}$ belongs to the block $A_{k,l}$. From Eq. (2.2) it follows that $a_{i,j} \in A_{k,l}$ if

$$k = \left\lfloor \left(i - \frac{1}{2}\right) \cdot \frac{m_B}{m_A} \right\rfloor + 1 \quad \text{and} \quad l = \left\lfloor \left(j - \frac{1}{2}\right) \cdot \frac{n_B}{n_A} \right\rfloor + 1. \tag{2.3}$$

Let $\mathbb{D} = \mathbb{R}$ or $\mathbb{C}$ if A is real or complex, respectively. Let $\mathcal{V} : \mathbb{D} \to \mathbb{R}$ be a function that satisfies

$$\mathcal{V}(0) = 0. \tag{2.4}$$

We call $\mathcal{V}$ the *visualization function*. Then, we define the elements of the visualization matrix B as follows:

$$b_{k,l} = \frac{1}{S_{k,l}} \sum_{a_{i,j} \in A_{k,l}} \mathcal{V}(a_{i,j}) = \frac{1}{S_{k,l}} \sum_{\substack{r_k \leq i \leq r'_k \\ c_l \leq j \leq c'_l}} \mathcal{V}(a_{i,j}), \tag{2.5}$$

where

$$S_{k,l} = (r'_k - r_k + 1) \times (c'_l - c_l + 1) \tag{2.6}$$

($S_{k,l}$ equals the number of elements of $A_{k,l}$).

In case of sparse A, condition Eq. (2.4) allows to calculate $b_{k,l}$ by performing the summation only over nonzero elements of $A_{k,l}$. We can thus rewrite Eq. (2.5) as

$$b_{k,l} = \frac{1}{S_{k,l}} \sum_{\substack{a_{i,j} \in A_{k,l} \\ a_{i,j} \neq 0}} \mathcal{V}(a_{i,j}). \tag{2.7}$$

Suppose now that we have an algorithm that calculates B for a sparse matrix A on a given HPC system. Let P denote the number of processors involved in an algorithm run and let $p_1, \ldots, p_P$ denote these processors. Then, we can consider A as

$$A = A^{(1)} + \cdots + A^{(P)},$$

where $A^{(q)}$ contains nonzero elements of A stored in the local memory of processor p_q. (We use the (q) superscript frequently in this text to indicate that some entity belongs to processor p_q.) We will further write simply $a_{i,j} \in A^{(q)}$ to indicate that the element $a_{i,j}$ is contained in $A^{(q)}$, therefore in the local memory of processor p_q. Let $|A^{(q)}|$ denote the number of nonzero elements of $A^{(q)}$.

The calculation of $b_{k,l}$ now becomes trickier, since the nonzero elements of $A_{k,l}$ can be distributed among multiple processors. Let $\mathcal{P}_{k,l}$ denote a set of processors each of which contains at least one nonzero element of $A_{k,l}$. Thus,

$$\mathcal{P}_{k,l} = \big\{\, p_q \;\big|\; \text{there exists } a_{i,j} \text{ such that } a_{i,j} \in A^{(q)} \text{ and } a_{i,j} \in A_{k,l} \big\}. \tag{2.8}$$

Conversely, let $\mathcal{A}_q$ denote a set of blocks from each of which processor p_q contains in its memory at least one nonzero element. Thus

$$\mathcal{A}_q = \big\{ A_{k,l} \;\big|\; \text{there exists } a_{i,j} \text{ such that } a_{i,j} \in A^{(q)} \text{ and } a_{i,j} \in A_{k,l} \big\}.$$

We can now rewrite Eq. (2.7) as

$$b_{k,l} = \sum_{q \in \mathcal{P}_{k,l}} b_{k,l}^{(q)}, \quad b_{k,l}^{(q)} = \frac{1}{S_{k,l}} \sum_{\substack{a_{i,j} \in A_{k,l} \\ a_{i,j} \in A^{(q)} \\ a_{i,j} \neq 0}} \mathcal{V}(a_{i,j}). \tag{2.9}$$

We further say that any processor from $\mathcal{P}_{k,l}$ *contributes* to the calculation of $b_{k,l}$.

Let $b_{k,*}$ denote the k-th row of B. Let $\mathcal{P}_{k,*}$ denote a set of processors that contribute to at least one element of this row, thus:

$$\mathcal{P}_{k,*} = \mathcal{P}_{k,1} \cup \cdots \cup \mathcal{P}_{k,n_B}.$$

We can now define the problem of acquisition of visualization data for a sparse matrix A as follows:

Problem 1. *We are looking for an efficient parallel algorithm that calculates the visualization matrix B, for a given sparse matrix A and a visualization function $\mathcal{V}$, and saves it to a file* `F`. *The additional requirements are:*

Req. 1: The algorithm should not depend on the type of matrix-processors mapping (the type of distribution of the nonzero elements of A among processors).

Req. 2: The algorithm should not depend either on the computer representation of A or on the order in which its nonzero elements are accessible.

Sparse matrices are stored in computer memory in special data structures called *sparse matrix storage formats/schemes* (see, for instance [10, 1, 12]). All these formats have one common feature—they allow to iterate over the nonzero elements. Due to Req. 2, we thus further regard $A^{(q)}, \ldots, A^{(P)}$ as a sequence of their nonzero elements with unspecified order. This allows us to work with A independently of the storage format used within an actual HPC code, where the matrix appears. Moreover, this allows us to work with matrices that are even not stored in memory at all (their elements are computed on-thy-fly).

The information we want to visualize is determined by $\mathcal{V}$. For instance, if we want to visualize the structure (pattern) of the nonzero elements of A, the visualization function might be defined simply as follows:

$$\mathcal{V}_1(a_{i,j}) = \begin{cases} 1 & \text{if } a_{i,j} \neq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{2.10}$$

Then, B would contain the *density* of the nonzero elements of blocks of A. Other visualization functions might be defined, e.g., as follows:

$$\mathcal{V}_2(a_{i,j}) = \begin{cases} |a_{i,j}| & \text{if } a_{i,j} \neq 0, \\ 0 & \text{otherwise,} \end{cases}$$

$$\mathcal{V}_3(a_{i,j}) = \begin{cases} \left|\mathrm{Re}(a_{i,j})\right| & \text{if } a_{i,j} \neq 0, \\ 0 & \text{otherwise,} \end{cases}$$

$$\mathcal{V}_4(a_{i,j}) = \begin{cases} \left|\mathrm{Im}(a_{i,j})\right| & \text{if } a_{i,j} \neq 0, \\ 0 & \text{otherwise.} \end{cases}$$

Thus, the usage of $\mathcal{V}_2$, $\mathcal{V}_3$, and $\mathcal{V}_4$ results in the visualization of the average magnitude of the elements of each block, the average magnitude of their real parts, and the average magnitude of their imaginary parts, respectively.

The visualization matrix B represents the intermediate visualization data that will be saved into `F` and from which the final matrix image will be constructed. We call `F` the *visualization file.* Recall that `F` is supposed to be downloaded from a parallel system to a user's personal computer. This, in effect, limits the size of `F` to some extent (presumably, to hundreds of megabytes or gigabytes nowadays), which consequently limits the size of B.

Suppose that `F` is a binary file and that we save the elements of B into `F` using a b-byte floating-point data type `T` (e.g., $b = 4$ for `T` being the IEEE 754 single-precision floating point data type [6]). The memory requirements in bytes for saving B in `F` are then

$$S(B) = m_B \cdot n_B \cdot b. \tag{2.12}$$

Let $S(\mathtt{F})$ denote the file size of F. Actually, it might be slightly higher than $S(B)$, since there might be some space overhead given by the used file format as well as by any supplemental saved information. In practice, such overhead should not exceed several kilobytes, therefore we consider it as negligible for further analysis and set

$$S(\mathtt{F}) = S(B). \tag{2.13}$$

Since our goal is to acquire as much visualization data as possible, we are looking for the answer to the following question: What is the maximum size of the visualization matrix B to be stored in `F` of size at most $S(\mathtt{F})$ bytes? Combining Eqs (2.1), (2.12), and (2.13), we can find the solution as

$$m_B = \left\lfloor \left(\frac{1}{b} \cdot \frac{m_A}{n_A} \cdot S(\mathtt{F}) \right)^{1/2} + \frac{1}{2} \right\rfloor \tag{2.14}$$

together with Eq. (2.1).

2.1. Example. Let A be a square matrix. Let us limit the size of the visualization file to 4 GB, which implies $S(B) = 2^{32}$. Let us use the IEEE 754 single-precision floating-point data type for storing elements of B into `F`, thus $b = 4$. Then, $m_B = n_B = 32768$. We can thus partition A into $32768 \times 32768 \approx 10^9$ blocks and save the visualization data in a form of a single floating-point number for each block into a file of approximate size 4 GB. From this file, we can then generate a matrix image up to the size of 1 gigapixel.

In the text below, the capitalized word "Algorithm" followed by a number always refers to a pseudocode presented by a corresponding floating text environment (as are figures and tables). Within pseudocode, we write references to corresponding equations in the end-of-line comments.

3. Methodology and Algorithm. To solve Problem 1, we need to develop an algorithm for the calculation of the visualization matrix B according to Eq. (2.9), and its storage into the visualization file `F`. First, note that we can translate Eq. (2.9) into pseudocode as follows:

Algorithm 9 Visualization data acquisition: computationally expensive solution

1: **for** $k \leftarrow 1$ **to** m_B **do**
2: **for** $l \leftarrow 1$ **to** n_B **do**
3: construct $\mathcal{P}_{k,l}$ ▷ (2.8)
4: **for all** processors in $\mathcal{P}_{k,l}$ **do in parallel**
5: $b_{k,l}^{(q)} \leftarrow 0$
6: **for all** local nonzero elements $a_{i,j}$ **do**
7: **if** $a_{i,j} \in A_{k,l}$ **then** $b_{k,l}^{(q)} \leftarrow b_{k,l}^{(q)} + \mathcal{V}(a_{i,j})$ ▷ (2.9)
8: **end for**
9: $b_{k,l}^{(q)} \leftarrow b_{k,l}^{(q)} / S_{k,l}$ ▷ (2.9), (2.6), (2.2)
10: perform parallel reduction of $b_{k,l}^{(q)}$ to $b_{k,l}$ using + operator ▷ (2.9)
11: write $b_{k,l}$ into `F`
12: **end for**
13: **end for**
14: **end for**

for all processors p_q in $\mathcal{P}_{k,l}$ **do in parallel**
$b_{k,l}^{(q)} \leftarrow 0$
for all local nonzero elements $a_{i,j} \in A_{k,l}$ **do** $b_{k,l}^{(q)} \leftarrow b_{k,l}^{(q)} + \mathcal{V}(a_{i,j})$
$b_{k,l}^{(q)} \leftarrow b_{k,l}^{(q)} / S_{k,l}$
perform parallel reduction of $b_{k,l}^{(q)}$ to $b_{k,l}$ using + operator
end for

Since this process needs to be performed for all possible combinations of k and l, a pseudocode that would solve Problem 1 might look like Algorithm 9.

The drawback of this approach is its computational complexity $O\big(|A^{(q)}| \cdot m_B \cdot n_B\big)$ for processor p_q, where $|A^{(q)}|$ might be very high in HPC applications. However, by rearranging Algorithm 9 as described below, we can reduce the computational complexity to $O\big(|A^{(q)}| \cdot \log_2 |\mathcal{A}_q| + m_B \cdot n_B\big)$. Since $|\mathcal{A}_q| \leq m_B \cdot n_B$,

$$|A^{(q)}| \cdot \log_2 |\mathcal{A}_q| + m_B \cdot n_B \leq |A^{(q)}| \cdot \log_2(m_B \cdot n_B) + m_B \cdot n_B \ll |A^{(q)}| \cdot m_B \cdot n_B,$$

when $|A^{(q)}|$, m_B, n_B are not all extremely small numbers.

The idea of this rearrangement is to split the solution of Problem 1 into two phases:

1. Within Phase1, processor p_q iterates over all its local nonzero elements $a_{i,j} \in A^{(q)}$. In each iteration, the contribution of $a_{i,j}$ to $b_{k,l}^{(q)}$ is calculated, where k and l are given by Eq. (2.3). This process can be performed by all processors in parallel with no communication costs.

2. Within Phase2, the local contributions $b_{k,l}^{(q)}$ are reduced in parallel to $b_{k,l}$, which is then written to the visualization file `F`.

The price for such a solution is the additional need to temporarily store the local contributions $b_{k,l}^{(q)}$ in memory of processor p_q. Usage of a *plain two-dimensional array* would imply the algorithm space complexity $O(m_B \cdot n_B)$ for each processor. Consequently, it would limit the size of B by the minimum of the available amounts of memory of all processors. An alternative is to store the contributions in an *associative array* (commonly also known as a *map* or a *dictionary*) of type $(k, l) \to \mathbb{R}$, where k and l are integers. The algorithm space complexity then changes to $O\big(|\mathcal{A}_q|\big)$ for processor p_q. In the *worst case*, processor p_q contributes to all elements of B, which turns this space complexity into $O(m_B \cdot n_B)$ as well (moreover, the hidden constants will be higher here, since the memory overhead of an associative array is higher than of a plain array). However, matrices are in practice mapped to processors typically according to some one- or two-dimensional partitioning scheme. In the *best case*, nonzero elements are distributed in matrices evenly, which implies $|\mathcal{A}_q| \approx m_B \cdot n_B / P$. The space complexity of an associative array-based algorithm then becomes $O(m_B \cdot n_B / P)$. (We observed such

Algorithm 10 Visualization data acquisition

Input: $A = A^{(q)}, \dots, A^{(P)}$: $A^{(q)}$ is located on p_q ▷ input sparse matrix
Input: m_A, n_A ▷ input matrix size
Input: P ▷ number of processors
Input: q ▷ actual processor number
Input: $\mathcal{V}$ ▷ visualization function
Input: $\mathtt{T}$ ▷ data type used for visualization data (elements of B)
Input: $S(\mathtt{F})$ ▷ required size of the visualization file in bytes
Output: $\mathtt{F}$ ▷ visualization file
Data: $\bumpeq\frown\ulcorner$ ▷ ordered associative array
Data: m_B, n_B, b ▷ auxiliary variables
for all processors $p_1, \dots, p_P$ **do in parallel**
  $b \leftarrow$ byte size of the element of type $\mathtt{T}$
  $m_B = \left\lfloor \left(1/b \cdot m_A/n_A \cdot S(\mathtt{F})\right)^{1/2} + 1/2 \right\rfloor$ ▷ (2.14)
  $n_B = \lfloor m_B \cdot n_A/m_A + 1/2 \rfloor$ ▷ (2.1)
  execute Phase1 ▷ calculate local contributions $b_{k,l}^{(q)}$
  execute Phase2 ▷ reduce them to $b_{k,l}$ and store into $\mathtt{F}$
end for

a behaviour in experiments [9].)

In HPC applications, computational problems are often solved as big as the available resources allow. Therefore, the minimum of available amounts of memory of processors can be a relatively small value. Since we do not want it to limit the size of B, we further consider, for the algorithm design, the storage of local contributions in associative arrays.

Moreover, we assume this associative array to be *ordered* lexicographically by the (k, l) key. This allows to iterate over the contributions $b_{k,l}^{(q)}$ of processor p_q in Phase2 only once, which is much more efficient than calling the lookup operation for each needed combination of k and l. We further denote an instance of the defined ordered associative array by $\bumpeq\frown\ulcorner$ and its record with the (k, l) key by $\bumpeq\frown\ulcorner[k, l]$.

3.1. Algorithm Pseudocode. We present here an efficient algorithm that solves Problem 1. Its outline is presented by a pseudocode as Algorithm 10. Let all variables introduced by Algorithm 10 have a global scope, i.e., they are available within the pseudocode of phases as well. Moreover, let all auxiliary variables/arrays defined in the "Data" section of each pseudocode be local to processors.

The pseudocode of Phase1 is shown as Algorithm 11. Within, each processor iterates over its nonzero elements and for each one, its contribution to the corresponding element of B is calculated. These contributions are stored in the $\bumpeq\frown\ulcorner$ data structure such that at the end of Phase1, $\bumpeq\frown\ulcorner[k, l]$ equals $b_{k,l}^{(q)}$ on processor p_q.

In Phase2, local contributions to the elements of B are reduced in parallel to their final values. These are then written to the visualization file $\mathtt{F}$. Performing the parallel reduction $m_B \times n_B$ times, each one for a single element of B, would be inefficient due to the overhead of communication operations. We therefore designed Phase2 such that the reductions are performed for whole rows of B at once, resulting in m_B reductions, each one for n_B elements at once.

To calculate $b_{k,*}$, all the contributions $b_{k,l}^{(q)}$ need to be reduced in parallel from processors $p_q \in \mathcal{P}_{k,*}$. Due to Req. 1 of Problem 1, each processor can generally contribute to any element of B. Consequently, the sets $\mathcal{P}_{k,l}$ and $\mathcal{P}_{k,*}$ can consist of all processors $p_1, \dots, p_P$. To calculate $b_{k,*}$, all the contributions $b_{k,l}^{(q)}$ need to be reduced in parallel from processors $p_q \in \mathcal{P}_{k,*}$. There are thus two options how to perform such a reduction:

1. from all processors $p_1, \dots, p_P$, while setting $b_{k,l}^{(q)} = 0$ for $p_q \notin \mathcal{P}_{k,*}$;
2. from $\mathcal{P}_{k,*}$ only, while this set need to be constructed first.

The second option provides no advantage over the first, since the construction of $\mathcal{P}_{k,*}$ would require collective communication between all processors as well. (In terms of MPI, it would require to form a processors group and a corresponding communicator. There has been, in fact, developed even a noncollective communicator

Algorithm 11 Visualization data acquisition: PHASE1

Data: k, l, r, r', c, c', S ▷ auxiliary variables
1: **for all** local nonzero elements $a_{i,j}$ **do**
2: $k \leftarrow \lfloor (i-1/2) \cdot m_B/m_A \rfloor + 1$ ▷ (2.3)
3: $l \leftarrow \lfloor (j-1/2) \cdot n_B/n_A \rfloor + 1$ ▷ (2.3)
4: **if** record (k,l) does not exist in $\Finv\frown^{\ulcorner}$ **then**
5: insert a record into $\Finv\frown^{\ulcorner}$ with (k,l) key and value $\mathcal{V}(a_{i,j})$ ▷ (2.9)
6: **else**
7: $\Finv\frown^{\ulcorner}[k,l] \leftarrow \Finv\frown^{\ulcorner}[k,l] + \mathcal{V}(a_{i,j})$ ▷ (2.9)
8: **end if**
9: **end for**
10: **for all** records (k,l) in $\Finv\frown^{\ulcorner}$ **do**
11: $r \leftarrow \lceil (k-1) \cdot m_A/m_B + 1/2 \rceil$ ▷ (2.2a)
12: $r' \leftarrow \lceil k \cdot m_A/m_B - 1/2 \rceil$ ▷ (2.2a)
13: $c \leftarrow \lceil (l-1) \cdot n_A/n_B + 1/2 \rceil$ ▷ (2.2b)
14: $c' \leftarrow \lceil l \cdot n_A/n_B - 1/2 \rceil$ ▷ (2.2b)
15: $S \leftarrow (r'-r+1) \times (c'-c+1)$ ▷ (2.6)
16: $\Finv\frown^{\ulcorner}[k,l] \leftarrow \Finv\frown^{\ulcorner}[k,l]/S$ ▷ (2.9)
17: **end for**

Algorithm 12 Visualization data acquisition: PHASE2

Data: $\circledR\ggg\backsim[]$ ▷ array of size n_B
Data: j, k, l, l' ▷ auxiliary variables
1: $j \leftarrow 1$
2: **for** $k \leftarrow 1$ **to** m_B **do** ▷ for all rows of B
▷ gather contributions $b_{k,l}^{(q)}$ for kth row:
3: **for** $l \leftarrow 1$ **to** n_B **do** $\circledR\ggg\backsim[l] \leftarrow 0$ ▷ ensure that $b_{k,l}^{(q)} = 0$ if $p_q \notin \mathcal{P}_{k,l}$
4: **while** $j \leq$ number of $\Finv\frown^{\ulcorner}$ records **and** $k' = k$, where (k',l') is the key of the jth $\Finv\frown^{\ulcorner}$ record **do**
5: $\circledR\ggg\backsim[l'] \leftarrow$ value of the jth $\Finv\frown^{\ulcorner}$ record ▷ $b_{k,l}^{(q)} \leftarrow \Finv\frown^{\ulcorner}[k,l]$
6: $j \leftarrow j+1$
7: **end while**
▷ reduce to $b_{k,l}$ on p_1 and write to F:
8: perform parallel reduction of all values of the $\circledR\ggg\backsim$ array using + operator to processor p_1 ▷ from all processors
9: **if** $q = 1$ **then** ▷ if run on processor p_1
10: **for** $l \leftarrow 1$ **to** n_B **do** append $\circledR\ggg\backsim[l]$ into file F ▷ write $b_{k,*}$ into F
11: **end if**
12: **end for**

creation technique, however, it does not perform well for our purposes [4]). We therefore further consider only the first option for algorithm design.

The pseudocode of PHASE2 is presented as Algorithm 12. Note that the order of processing the local contribution matches the order of records in the $\Finv\frown^{\ulcorner}$ data structure. This allowed us to design the pseudocode such that no $\Finv\frown^{\ulcorner}$ lookup operation is needed, which considerably reduced the overall algorithm complexity. The auxiliary array $\circledR\ggg\backsim$ serves as a buffer for storing and reducing local contributions for a single row of B. Before reduction of the kth row contributions, $\circledR\ggg\backsim[l] = b_{k,l}^{(q)}$ on processor p_q. After this reduction, $\circledR\ggg\backsim[l] = b_{k,l}$ on processors p_1. The reduction is performed by all processors (see line 1 in Algorithm 10).

3.2. Complexities. Let us analyze complexities of the presented algorithm. Generally, the complexities highly depend on the matrix-processors mapping. We therefore restrict our analysis to the following two extreme

cases: In the *worst case*, there is at least one processor that contributes to all $m_B \times n_B$ elements of B. On the other hand, in the *best case*, all processors contribute to at most $\lceil m_B \times n_B / P \rceil$ elements of B.

We define the complexity of the algorithm as its complexity for the most loaded processor (this value determines both the algorithm running time as well as its per-processor memory requirements).

3.2.1. Computational Complexity. For processor p_q, the *computational complexity* of Phase1 is given by two iterative processes defined at lines 1–9 and 10–17 of Algorithm 11. The computational complexity of the first process is given by iterating over $|A^{(q)}|$ nonzero elements, while in each iteration a record is either found or inserted into $\circlearrowleft\!\!\frown^{\ulcorner}$. Recall, that $\circlearrowleft\!\!\frown^{\ulcorner}$ is an ordered associative array. These are typically implemented as binary search trees with logarithmic complexity of both search and insert operations (see, for instance [2, Chap. 12]). The computational complexity of the second process is given by iterating over all $|\mathcal{A}_q|$ records of $\circlearrowleft\!\!\frown^{\ulcorner}$. The computational complexity of Phase1 on processor p_q is thus

$$T^{(q)}_{\text{Phase1}} = O\big(|A^{(q)}| \cdot \log_2 |\mathcal{A}_q| + |\mathcal{A}_q|\big).$$

In the worst case, $|\mathcal{A}_{q'}| = m_B \cdot n_B$ for some processor $p_{q'}$. Therefore, the computational complexity of Phase1 becomes

$$T^{(\text{worst})}_{\text{Phase1}} = O\big(\max_{1 \leq q \leq P} |A^{(q)}| \cdot \log_2 (m_B \cdot n_B) + m_B \cdot n_B\big).$$

In the best case, $|\mathcal{A}_q| \leq \lceil m_B \cdot n_B / P \rceil$ for all processors. The computational complexity of Phase1 then equals

$$T^{(\text{best})}_{\text{Phase1}} = O\Big(\max_{1 \leq q \leq P} |A^{(q)}| \cdot \log_2 \big(\lceil m_B \cdot n_B / P \rceil\big) + \lceil m_B \cdot n_B / P \rceil\Big).$$

The computational complexity of Phase2 is, for processor p_1 and thus for the whole algorithm,

$$T_{\text{Phase2}} = O(m_B \cdot n_B)$$

in all cases.

3.2.2. Space Complexity. The space complexity of Phase1 is given by the number of records of the associative array $\circlearrowleft\!\!\frown^{\ulcorner}$, which equals $|\mathcal{A}_q|$ on processor p_q. The space complexity of Phase2 is given by the auxiliary array $\circledR\!\!\ggg\!\!\backsimeq$ of size n_B. The overall space complexity of the algorithm is thus

$$S^{(\text{worst})} = O(m_B \cdot n_B) \quad \text{and} \quad S^{(\text{best})} = O\big(\lceil m_B \cdot n_B / P \rceil + n_B\big)$$

in the worst case and best case, respectively.

3.2.3. Communication Complexity. The communication complexity depends on the network topology of a given HPC system as well as on the implementation of communication operations by the utilized version of the MPI library. We can therefore only discuss the number of communication operations, instead of analyzing their asymptotic behaviour in terms of running time. In Phase1, there is no communication at all. In Phase2, m_B parallel all-to-one reductions are performed by all processors, each over n_B elements.

4. Experiments. We have performed a weak-scalability study (constant problem size per processor), since it matches the real-world situations, where HPC problems are usually solved as large as available resources allow. On modern hybrid shared-distributed memory HPC systems, the available amount of memory, which limits the size of A, is thus proportional to the number of utilized processors P. In experiments, we always set the size of A such that its number of nonzero elements was approximately $P \cdot 4.6 \cdot 10^7$ (this corresponded to the per-processor memory requirements of A being around 1 GB using the coordinate storage scheme and the single-precision floating point data type for matrix elements).

As a source of large sparse matrices, we used the parallel scalable generator of benchmark sparse matrices [8]. This generator produces matrices by scalable enlargement of a small fixed *seed* matrix. As the seed matrix, we used the cage12 real square matrix obtained from the University of Florida Sparse Matrix Collection [3].

TABLE 4.1
Configurations of HPC systems and their run-time environments used for experiments.

System:	Anselm	Edison	Blue Waters
Provider:	IT4Innovations	NERSC	NCSA
Type:	Linux cluster	Cray XC30	Cray XE6
Total CPU cores:	3344	124608	362240
Total memory [TB]:	15	332	1380
Interconnect:	Infiniband	Aries	Gemini
Topology:	Fat tree (nb)	Dragonfly	3D torus
I/O bandwidth [GB/s]:	6	48	1000+
C++ compiler:	Intel icpc 13.1	Intel icpc 14.0	GNU g++ 4.8.1
MPI library:	Intel MPI 4.1	cray-mpich 6.2.0	cray-mpich 6.0.1
HDF5 library version:	1.8.11	Cray 1.8.11	Cray 1.8.11

We set the input algorithm parameters so to generate a matrix image according to Section 2.1. The size of B was thus always set to $m_B = n_B = 32768$, which resulted in possibility of generation of a 1 gigapixel matrix image. As a visualization function, we used $\mathcal{V}_1$ defined by Eq. (2.10).

We have implemented the presented algorithm with C++ and MPI to evaluate its performance and scalability. As an ordered associative array, we used the `std::map` container from the C++ Standard Library [7, Sect. 7.7], which is typically implemented as a red-black tree [2, Chap. 13]. We exploited the HDF5 library [13] for creation of visualization files `F`.

We utilized several modern parallel systems that are listed in Table 4.1, where we show their parameters together with the C++ compilers and the MPI libraries used for compilations of test programs and their runtime execution. For the Blue Waters system, we present parameters of the XE cabinets only (the XK cabinets are intended primarily for GPU-based computations).

In all experiments, we primarily measured algorithm running times. These are presented by Figure 4.1. The results indicate that the majority of the running time is spent in PHASE2, which is caused by the execution of communication and I/O operations. However, the algorithm is generally fast and the overall algorithm running time grows only slightly with the growing number of processors.

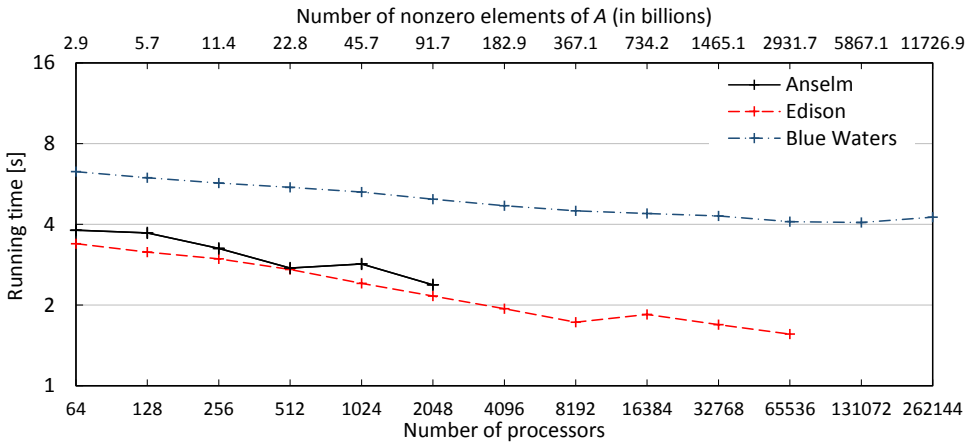
We also measured the estimated memory requirements of the $\widetilde{\frown}^{\ulcorner}$ data structure for all processors. Their maximal and average values are shown in Figure 4.2. Up to some small constant memory overhead, these memory requirements clearly decrease inversely proportionally to the number of processors P. For higher P, the overall memory footprint of the algorithm is thus negligible in practice.

5. Conclusions. This paper extends the work of Langr et al. [9]. It presents an updated version of the algorithm for the parallel acquisition of visualization data for large sparse matrices, along with its more detailed analytical and empirical evaluation. The main characteristic of the algorithm, from a user's point of view, is its application independence. It can be used with any mapping of matrices to processors and any sparse storage formats/schemes. Moreover, it can be used even when matrix nonzero elements are computed on-the-fly. It can be also easily implemented using the MPI parallel programming library, from which it needs to execute only the all-to-one parallel reduction operations.

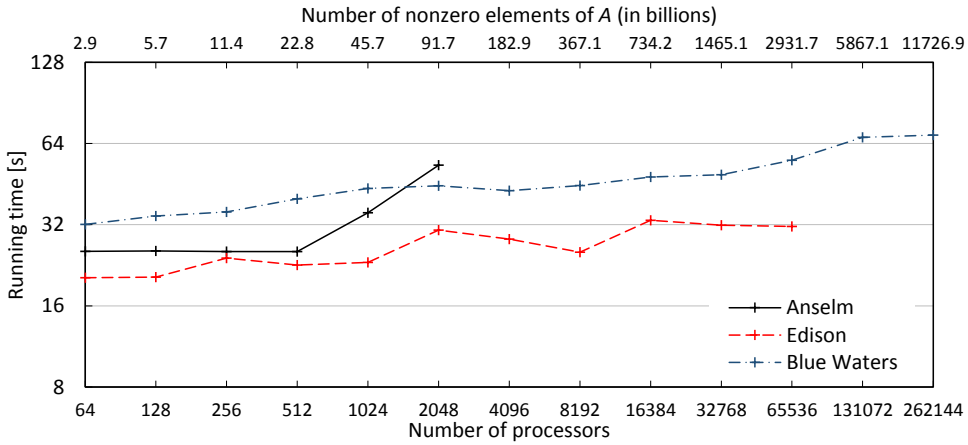
Using this algorithm, one can visualize sparse matrices emerging in an HPC code with minimal runtime and memory overhead. Namely, within the largest presented experiment, 266144 processors gathered and stored visualization data for a sparse matrix with $1.17 \cdot 10^{13}$ nonzero elements with a measured runtime of 73 seconds.

The majority of the algorithm running time takes the collective communication between all processors together with serial I/O operations. Resulting visualization data are appended to output files, which allows, among others, visualization files to be ASCII-based. Due to the application independence of the algorithm, parallel I/O cannot be efficiently employed to increase the I/O performance. This does not seem to be of much importance at the petascale performance level of today's prime HPC systems, as demonstrated by the presented experiments.

However, for emerging exascale and higher-level computing, even faster algorithms might appear to be necessary. In our future work, we want to focus on two promising options how to achieve them. The first one is



(a) Phase1



(b) Phase2

Fig. 4.1. *Running times of both algorithm phases measured on different parallel systems.*

to adapt the presented algorithm for the hybrid MPI+OpenMP parallel programming model, which naturally represents the shared-distributed memory architecture of modern HPC systems. The second option is to give up the application independence and to adapt the algorithm for certain types of matrix-processor mappings, which should allow to reduce the burden of collective communication and to efficiently utilize parallel I/O.

REFERENCES

[1] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. V. der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, PA, 2nd ed., 1994.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, The MIT Press, Cambridge, Massachusetts, USA, 3rd ed., 2009.

[3] T. A. Davis and Y. F. Hu, *The University of Florida Sparse Matrix Collection*, ACM Transactions on Mathematical Software, 38 (2011).

[4] J. Dinan, S. Krishnamoorthy, P. Balaji, J. R. Hammond, M. Krishnan, V. Tipparaju, and A. Vishnu, *Noncollective*

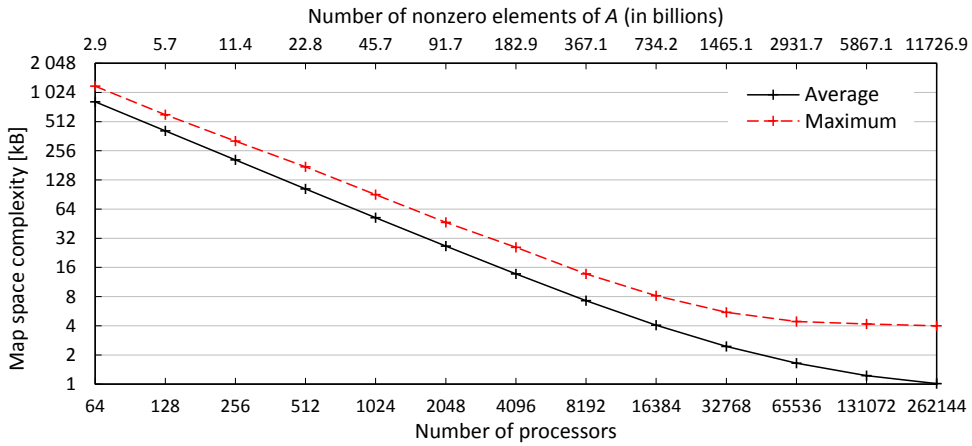


FIG. 4.2. *Statistics of the estimated per-processor memory requirements of the ⩒⌒⌜ data structure.*

communicator creation in mpi, in Proceedings of the 18th European MPI Users' Group Conference on Recent Advances in the Message Passing Interface, Y. Cotronis, A. Danalis, D. Nikolopoulos, and J. Dongarra, eds., vol. 6960 of Lecture Notes in Computer Science, Berlin, Heidelberg, 2011, Springer-Verlag, pp. 282–291.

[5] W. GROPP, S. HUSS-LEDERMAN, A. LUMSDAINE, E. LUSK, B. NITZBERG, W. SAPHIR, AND M. SNIR, *MPI—The Complete Reference, Volume 2: The MPI-2 Extensions*, MIT Press, Cambridge, MA, USA, 1998.

[6] *IEEE Standard for Floating-Point Arithmetic*, IEEE Std 754-2008, (2008), pp. 1–58.

[7] N. M. JOSUTTIS, *The C++ Standard Library—A Tutorial and Reference*, Addison Wesley Longman, Boston, MA, USA, 2nd ed., 2012.

[8] D. LANGR, I. ŠIMEČEK, P. TVRDÍK, AND T. DYTRYCH, *Scalable parallel generation of very large sparse matrices*, in 10th International Confernce on Parallel Processing and Applied Mathematics (PPAM 2013), Lecture Notes in Computer Science, Springer Berlin Heidelberg. Accepted for publication.

[9] ———, *Parallel data acquisition for visualization of very large sparse matrices*, in Proceedings of the 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2013), Los Alamitos, CA, USA, September 2013, IEEE Computer Society, pp. 338–345.

[10] Y. SAAD, *Iterative Methods for Sparse Linear Systems*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd ed., 2003.

[11] M. SNIR, S. OTTO, S. HUSS-LEDERMAN, D. WALKER, AND J. DONGARRA, *MPI—The Complete Reference, Volume 1: The MPI Core*, MIT Press, Cambridge, MA, USA, 2nd (revised) ed., 1998.

[12] P. T. STATHIS, *Sparse Matrix Vector Processing Formats*, PhD thesis, Technische Universiteit Delft, 2004.

[13] *The HDF Group. Hierarchical data format version 5, 2000-2013.* `http://www.hdfgroup.org/HDF5/` (accessed June 3, 2013).

Edited by: Teodor Florin Fortiş
Received: Feb 28, 2014
Accepted: Apr 2014

DOI 10.12694/scpe.v15i1.964
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 15, Number 1, pp. 33–48. http://www.scpe.org

ADAPTIVE MULTI-GRID METHODS FOR PARALLEL CFD APPLICATIONS*

JÉRÔME FRISCH, RALF-PETER MUNDANI, AND ERNST RANK

Abstract. Computational fluid dynamic (CFD) computations are memory and time intensive and need to be executed in parallel for larger computational domains. In order to produce physical accurate solutions, adaptive grid setups have to be chosen as the memory and computing time would otherwise be too high, and results would not be obtained in a reasonable amount of time. This paper describes the usage of a multi-grid based approach for solving the pressure Poisson equation, arising during every time step of the Navier-Stokes equations. It will then highlight an analysis of errors introduced due to an adaptive setup of the domain, and show performance measurements for uniform and adaptive grid setups. Last but not least, a CFD benchmark example based on the von Kármán vortex street will be presented, and the results will be discussed.

Key words: parallel computation, adaptive data structure, multi-grid like solver, computational fluid dynamics, Poisson equation, message passing paradigm

AMS subject classifications. 65Y05, 65G99, 65M55, 35J05, 76D05

48

1. Introduction. Computational fluid dynamics (CDF) simulations are time intensive and require a lot of computational resources. Hence, every optimisation or algorithmic speed-up has huge benefits for expensive long running scenarios in terms of energy efficiency and CPU costs. Moreover, design parameters regarding domain setup or solver settings also prove to have a high impact on run times.

Uniform grids exhibit huge advantages due to very easy computations via stencil-based operations, they require no additional grid management, and are very easy to implement (cf. [5]). Thus, it is very cost efficient using structured uniform grids for computations. One disadvantage of such an approach is the required equidistant discretisation of the entire domain, which is not always necessary depending on the geometry settings. For an engineering based air-flow simulation of an indoor airspace for example, regions have to be computed very fine around obstacles but do not necessarily need to have the same resolution at regions far away from these obstacles. Thus, a geometry based, adaptively refined grid has huge advantages in terms of reducing unnecessary refinements.

Adaptive grids are already quite well studied in literature. There exist several possibilities of generating adaptive grid refinements. One possibility is the stepwise approximation with regular grids, such as the approach described in this paper or from Manhart and Wengle [13]. Another possibility which will not be regarded for this code would be the overlapping of two different grids of different resolution, so called Chimera grids, as studied by Hadžić [11]. Furthermore, boundary adapted non-orthogonal grids could be used. All methods have advantages and disadvantages listed in detail in [5]. The authors chose to use the approximation with hierarchical block-structured grids, as they fit quite well to the used data structure.

The main CFD implementation is based on an incompressible Newtonian fluid flow simulation governed by the Navier-Stokes equations. The spatial discretisation is realised by a finite-volume scheme and for the temporal discretisation currently an explicit forward Euler method enhanced by the Adams-Bashforth method of 2^{nd} order is applied. Further details are given in section 6.

A Poisson equation for the pressure term has to be solved in every step of the fluid solver, hence this step needs to be analysed in detail in order to speed up the solution process. This paper will describe a multi-grid based approach for solving the pressure Poisson equation which is deeply integrated in the data structure itself, and takes advantage of the internal exchange functions and similar concepts of the two approaches. Further details can be found in section 4.

The paper quickly outlines the applied hierarchical data structure for block-structured, orthogonal Cartesian grids and presents additions to compute with an adaptively refined grid structure. For a fair parallel efficiency, a load distribution has to be performed in order to balance the computational effort to the various machines. A detailed analysis concerning errors generated by an adaptive refinement is presented and a performance overview is given. Last but not least, an adaptively refined von Kármán vortex street is computed as test example, which is according to Hirsch [12] a simple geometry which generates a quite complex flow.

*Chair for Computation in Engineering, Technische Universität München, 80290 Munich, Germany (frisch@tum.de).

This paper is based on the conference publication [8], and was extended by including a section on the multi-grid based solver concept, as well as further performance measurements and an enhanced error evaluation.

2. Data Structure Layout. The data structure is based on block-structured, non-overlapping, regular, orthogonal Cartesian grids organised in a hierarchical management grid as depicted in figure 2.1. Two main components form up the system: the management grids on the one hand and the data grids on the other hand. The management grids are ordered according to a hierarchical organisation structure in a tree-like fashion starting from a root grid (also called top grid, cf. figure 2.1 on the right hand side). They contain logical information such as parentage or children links. One management grid can have up to 7^3 management grid children, although the refinement in each dimension can be selected separately.

A special case is the so called octree data structure, if the child refinement is set to 2 in every direction, giving 2^3 management grid children per parent grid. Furthermore, the refinement of the root (r_x^t, r_y^t, r_z^t) can be chosen differently than that of any subsequently refined grid (r_x^s, r_y^s, r_z^s), in order to allow for a useful discretisation of e. g. a channel. A refinement of $r_i = 1$ can be chosen in order to impose a non-refinement in a given axis i, which can be useful for a pseudo-2D computation. The refinement levels impose certain limits on the accuracy of the solution in an adaptive area, and thus, studies about accuracy will be carried out in section 5 of this paper in order to quantify this approximation.

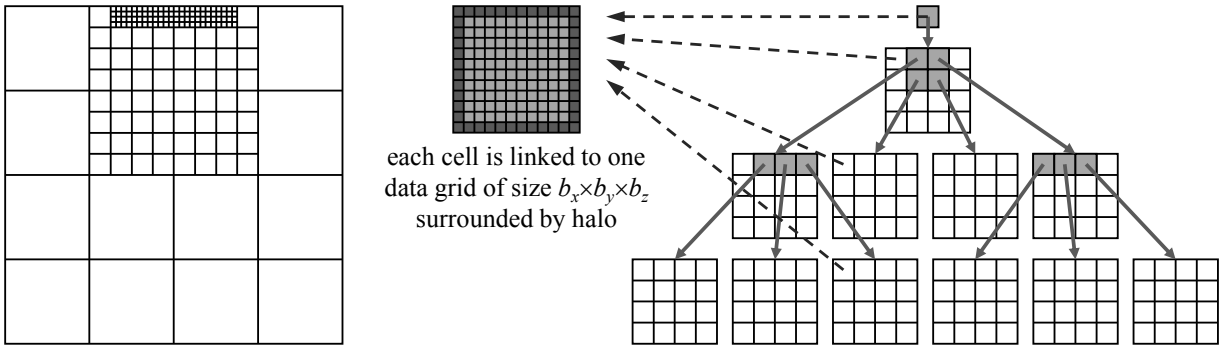


FIG. 2.1. *Logical data structure layout in tree form (on right-hand side) and in overlayed grid form (on left-hand side). Each logical grid holds a pointer to a data grid containing the actual data surrounded by a halo of ghost cells (mid of picture). (Picture based on [9])*

The second main part of the data layout is formed by the data grids depicted in the middle of figure 2.1. Each management grid is linked to exactly one data grid, which will hold the actual computational data of all internal fields, such as velocities, pressure, temperature, etc. The size of a data grid is fixed to (b_x, b_y, b_z) for all grids, with the restriction, that b_i has to be dividable without remainder by r_i^t and r_i^s for all i in order to avoid non-conforming grid boundaries.

This data structure setup enables a strict separation of computation and communication phases. From the user's point of view, a new kernel can be implemented on a regular Cartesian grid. Everything else which has to do with adaptivity, interpolation, or data exchange is outsourced to communication functions which are called by one single call from within the kernel. Thus, an expert in his area of research can easily implement a new computational kernel without being an expert in parallel computing or computer science.

On the other hand, the data exchange has to be treated with extreme care, as race conditions during the update procedure could occur and generate undefined data states. Hence, the communication has to be separated into three different steps which have to be properly synchronised: in a first step, data is aggregated to the parent cells by interpolating child values. The parent grid can only proceed, if updates from all its children have arrived. As the data is send upwards in the scheme depicted in figure 2.1, this stage is called bottom-up communication stage. In a next step, neighbour cells on the same level can exchange values without any synchronisation needs, as no race condition can occur here. This step is called horizontal communication.

As last step, all the ghost cells of the grids have to be filled which have not been touched in the previous horizontal communication phase. Therefore, data stored in the current ghost cells of the parent grids are used to interpolate values in the ghost cells of the child grids. Further information about the communication procedures or the grid setup can be found in [6, 9].

It should be mentioned that unlike in similar concepts, the intermediate grids are not discarded during the refinement process. Hence, all grids are kept in memory, even if refined children are available. The actual computation however, will be carried out only on the 'leaf' (i. e. non-refined) grids. This concept has some considerable advantages. For one, the applied solver for the Poisson problem introduced in section 4 is using the parent grids for its internal solving procedures.

A second advantage is that if a very fast computational response is necessary, such as in the case of a computational steering approach (cf. [19]), e. g., a maximal computation depth can be imposed, where higher refinements will be disregarded, and a coarser but faster computation can be performed on the exact same data structure.

As third advantage, a visualising technique for reducing the amount of transferred data can be applied easily, such as described in [16], called sliding-window approach. Here, a certain window of interest as well as a maximal bandwidth can be fixed. A special server-sided collector decides which data on which level will be sent to a client-sided visualisation application. Thus, interactive data visualisation can be managed at run-time with constant bandwidth requirements for data exchange.

3. Parallel Data Distribution and Load Balancing. For a parallel computation, a good data distribution is inevitable in order to minimise communication over the network. The following section will describe how such a distribution can be achieved by using space-filling curves.

3.1. Neighbourhood Server Concept. For a load distribution or redistribution, two different strategies can be applied: a global view or a local view strategy. In the local view strategy, a diffusion process as proposed by Cybenko [4] can be applied. No information about the global system is necessary, and work is only distributed to the direct neighbours. This process has to be repeated a certain amount of steps, until an overall balanced distribution has been reached. On the other hand, a global view can do the distribution in one single step by having global knowledge of the problem. Unfortunately, the process having the global view typically becomes a bottle neck with an increasing amount of processes in the system.

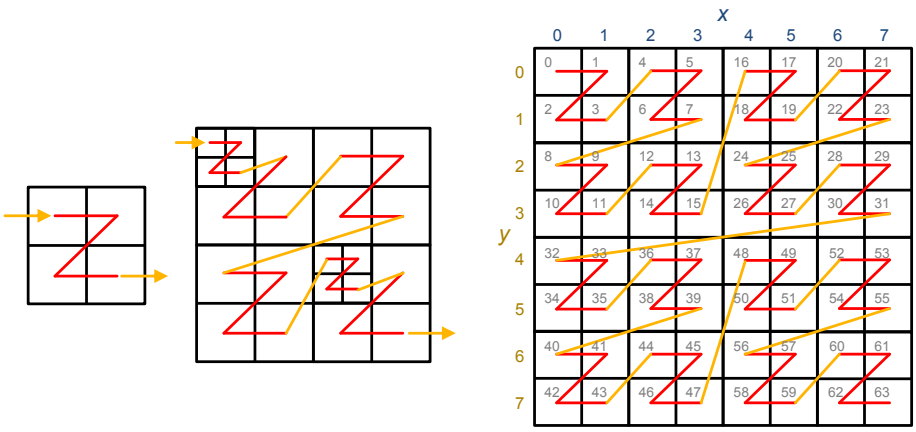
Nevertheless, due to its better characteristics the authors chose to implement a global view concept. A dedicated server, the so called 'neighbourhood server', keeps track of all management grids without having any actual data of the grids themselves. Using this concept, rebalancing and grid migrations are easily possible without a lot of overhead. Possible bottle-neck simulations are avoided by having multiple servers running concurrently and serving different processes. Details of the synchronisation procedures can be found in [7].

3.2. Load Balancing using Space-Filling Curves. The neighbourhood server contains all management grids and can compute the best possible data distribution according to a certain given strategy. Possible distribution strategies should ensure a good data locality, i. e. keeping neighbouring grids as close as possible, and thus, minimising communication efforts. Space-filling curves deliver a good distribution for such purposes (cf. [1]).

Of all the possible space filling curves (SFC), such as Hilbert, Lebesgue, Sierpiński, or Peano, among many others, the authors chose to use the Lebesgue curve depicted in Figure 3.1. The main reason for this choice is the very simple computation of the position of a given cell in the grid. The geometric construction of the curve can be seen in Figure 3.1(a). On the left-hand side, the geometric primitive is shown. Each part in the grid can then be refined further by applying again the construction primitive (due to the form of the primitive, the curve is also referred to as 'z-order curve').

The hierarchical tree is traversed in a depth first approach, respecting on every level the geometric primitive of the construction, thus delivering the figure on the right-hand-side of figure 3.1(a). Contrary to other work (e. g. [20]), the parent grids are not omitted, but also pushed to the linearised list. Thus, the list of grids on the right-hand-side would have a size of 29, even if only 22 of them are currently visible. This linearised list is split into (nearly) equal parts and distributed by means of grid migration to different computing processes.

Although, the Hilbert curve would conserve the locality better than the Lebesgue curve, it is easier to



(a) geometric construction of a Lebesgue curve on an adaptive grid

(b) uniform grid distribution with enumeration according to Morton's index.

FIG. 3.1. *Load balancing strategy by applying Lebesgue's curve and Morton's indexing.*

compute the position in the grid using Morton's index [15], which can be generated easily by bit-interleaving. As example, an 8×8 grid is depicted in Figure 3.1(b). By writing the desired coordinates in binary form and padding them to the largest multiple of two, one can generate the Morton index by interleaving each bit from y with each bit from x. Conversion from binary into decimal form delivers then Morton's index. As example, $x = 5,\ y = 3$ is written in binary form $x = 101,\ y = 011$. By bit-interleaving, one obtains 011011 which reads in decimal 27 and corresponds to the Morton index of x, y. The same procedure works in 3D by using x, y, and z coordinates.

4. Multi-Grid Like Solver Concept. As mentioned in the previous section, the solver concept is based on a multi-grid like approach for the pressure Poisson equation. The basics, describing the mathematical properties, convergence studies, etc can be found in [2, 18, 10]. Putting it into a nutshell, a multi-grid based approach can be described as follows:

Let's suppose the Poisson equation $\mathbf{A} \cdot u = f$ should be solved on a given grid discretisation Ω^h with a spacing of h. First, a smoothing using a 2/3 damped Jacobi relaxation method is applied α_1 times, obtaining an approximation to the solution v^h on this grid. The residual $r^h = f^h - \mathbf{A}^h \cdot v^h$ is computed and restricted to a coarser grid using a restriction operator $\mathbf{I}_h^{2h}$ to serve as right-hand side f^{2h}. On the coarser grid Ω^{2h}, the residual equation $\mathbf{A}^{2h} \cdot e^{2h} = f^{2h}$ is relaxed to obtain an approximation to the error e^{2h}. Again, the residual $r^{2h} = f^{2h} - \mathbf{A}^{2h} \cdot e^{2h}$ is computed and restricted to an even coarser level using a restriction operator $\mathbf{I}_{2h}^{4h}$.

This procedure is repeated until the coarsest level L has been reached, where the residual equation system $\mathbf{A}^{Lh} \cdot e^{Lh} = f^{Lh}$ is small enough to be solved directly. After e^{Lh} is known, the error correction is prolongated to the finer grids and added to the approximation of the solution v. Assuming the error correction e^{4h} is known, then the approximation on level $2h$ can be obtained by $e^{2h} = e^{2h} + \mathbf{I}_{4h}^{2h} \cdot e^{4h}$, where $\mathbf{I}_{4h}^{2h}$ describes the prolongation operator. The error is then relaxed α_2 times using the 2/3 damped Jacobi smoother, before the prolongation to level h is performed. In the end, the error correction e^h is added to the approximation of the solution v^h. This procedure is called a v-cycle and is the simplest form of a multi-grid approach. This v-cycle has to be repeated until the norm of the residual is below a given tolerance, at which point the system is solved to the given accuracy.

While analysing the standard multi-grid approach, similarities can be seen between prolongation and restriction in the v-cycle, and the data exchange mechanisms introduced in the data structure itself. Therefore, it is obvious to combine the two approaches reusing parts already existing in order to construct a multi-grid like solver which is 'fused' into the structure itself. The method is based on matrix-free cell-centred multi-grids,

meaning that no matrix **A** is explicitly assembled, but a stencil operator is used for iterating over the grid points in memory. As the data grid uses a collocated data arrangement, where all values are stored in the cell centre, the data for the multi-grid also needs to be interpreted as cell-centred. As consequence, coarsened points do not coincide with finer points.

The amount of levels and thus, implicitly the amount of grid coarsening is dictated by the data structure itself and by the approximation of the geometry which should be solved.

Figure 4.1 shows performance measurements on 'Shaheen', a 16-rack Blue Gene/P system having a total of 65536 compute cores at King Abdullah University of Science and Technology (KAUST) in the Kingdom of Saudi-Arabia[1]. Figure 4.1(a) shows the pure data exchange times for exchanging all ghost layers of 9 independent variables for different problem sizes using up to 32768 processes. The curves show, that the communication time is decreasing while using more processes. Figure 4.1(b) shows strong scaling results for two different problem sizes for a 3D Laplace problem discretised uniformly up to a given depth. It can be seen, that a certain saturation is reached for a higher amount of processes. This happens earlier for the 16 million unknowns and starts later for the 134 million unknowns due to increased communication overhead and less work per process.

The standard multi-grid algorithm uses a certain amount of smoothing steps α_1 for the residual equation and α_2 for the error correction. These values are usually in a range of approximately 2–3. The multi-grid like concept needs more smoothing steps in order to converge to the correct solution, especially when obstacles are present in the domain.

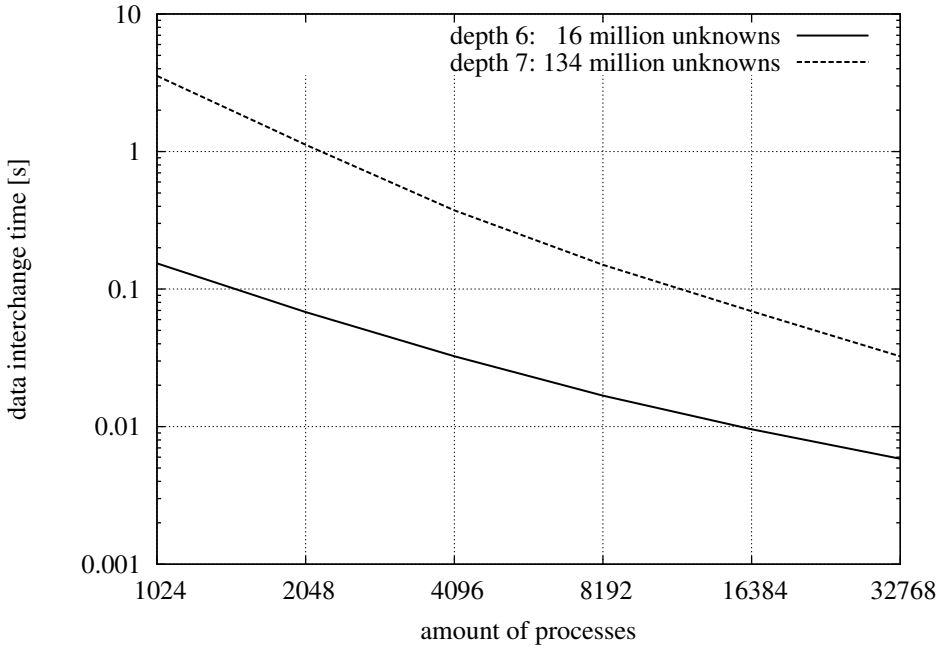
Following the ideas of [14], boundary conditions need to be handled on coarse grids in order to reproduce the effect they exhibit on the fine grids. Therefore, they need to be restricted in some way onto coarser grids. Dirichlet conditions have the highest priority, i. e. if one of the child cells has a Dirichlet boundary condition, the coarse cell will be marked as having a Dirichlet boundary condition. Neumann boundary conditions have a very low priority, and will only be restricted as Neumann conditions on the coarse grid if all child cells had Neumann boundary conditions. Otherwise, the coarse cell will be marked as fluid cell. As consequence, huge Dirichlet regions will appear in the coarsest grid representation, which can only be handled by a higher amount of smoothing towards the boundary regions.

Figure 4.2 shows a plot of the error convergence for different smoothing strategies until a maximal error of 10^{-9} was reached. While applying $\alpha_1 = \alpha_2 = 10$ on every level, 153 cycles were needed to converge (curve 'constant 10'). A next approach is to adaptively change α over different levels. On fine grids, α should be small, as exchanging data is costly. On coarser grids however, α can be increased as less data has to be exchanged, which produces less communication overhead. Hence, the plots named 'adaptive' use such a concept. 'Adaptive 2' starts with 2 smoothing steps on the finest level. In the next coarser level, 4 steps are used, then 8 and so on. This curve shows an even worse convergence than the constant 10. 'Adaptive 4' uses 4 smoothing steps to start on the finest level, then 8 on the next coarser, then 16 and so on. This curve shows exceptionally good convergence and needs only 27 cycles to converge to an error of 10^{-9} whereas 'adaptive 2' needed 477 cycles. This study shows that attention has to be paid to the choice of settings for the solver, as small changes could generate a huge performance impact.

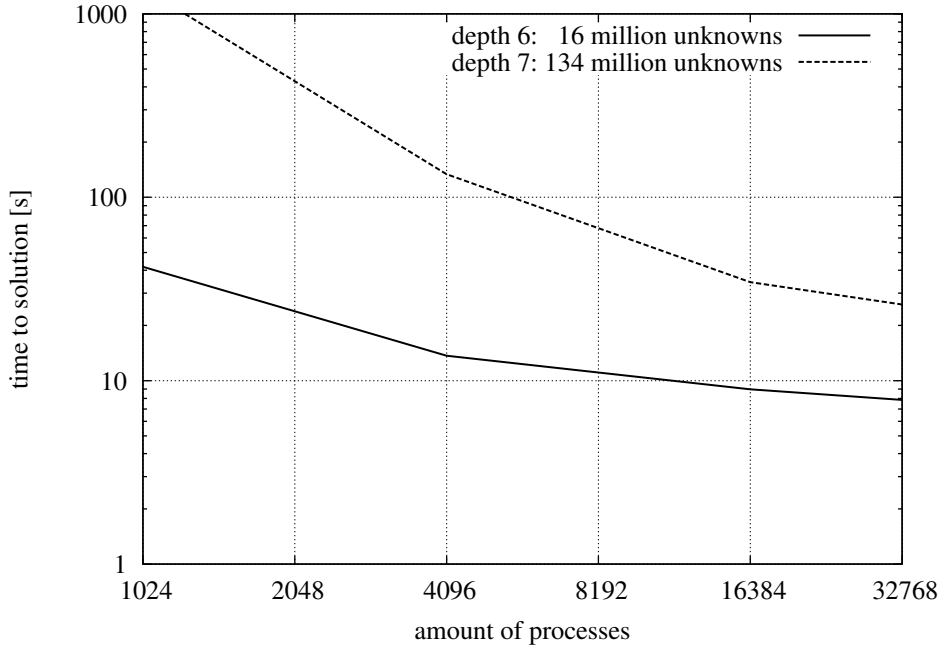
5. Performance and Accuracy Measurements of the Adaptive Data Structure. In order to evaluate the performance as well as the accuracy of the used data structures and its solvers, a Poisson equation will be solved for a simple test geometry. The use of the Poisson equation is realistic, as this equation has to be solved in every time step of a CFD simulation for evaluating the pressure-correction in order to obtain a divergence free velocity field of an incompressible fluid flow after each time step, and thus, ensuring the validity of the continuity equation.

5.1. Test Case Setup. The 3D test case geometry is composed out of a $1 \times 1 \times 1$m block. The top-level as well as the subsequent refinement levels are set to bisection ($r_i^t = r_i^s = 2 \quad \forall i$). The block size of the data grids is set to $b_i = 12$ in every direction. Dirichlet boundary conditions are imposed on all 6 walls. The west and east wall in x-direction are set to 1.0 whereas all other walls are set to 0.0. The right-hand-side f of the Poisson equation $\Delta p = f$ is set to 0, thus describing actually a pure Laplace equation.

[1]KAUST Supercomputing Lab: `https://www.hpc.kaust.edu.sa`



(a) Communication times for a total ghost cell exchange of 9 independent variables in seconds on 'Shaheen' for a 3D domain of fully refined grids with a refinement level (2,2,2) up to depth 6 or 7 and a data grid size of (4,4,4).



(b) 'Shaheen' strong scaling with different depths and block sizes of (4,4,4)

FIG. 4.1. *Performance Measurements for different problem sizes for a 3D Laplace problem on an uniformly refined grid using refinements (2,2,2), block sizes of (4,4,4), and various depths.*

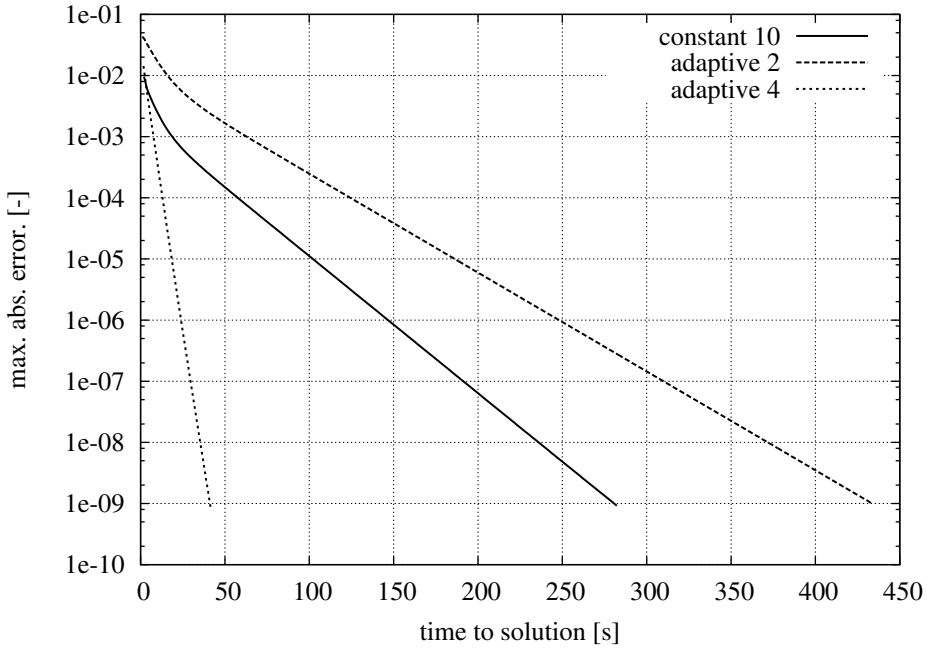


FIG. 4.2. *Time to solution for different choices of $\alpha_{1,2}$ computed on 1024 processes on 'Shaheen' for a 3D Laplace problem on an uniformly refined grid in depth 6 using refinements (2,2,2) and block sizes (4,4,4).*

Figure 5.1 shows a 2D cut through the 3D test problem case (red means $p = 1.0$, blue means $p = 0.0$). In the case of a uniform refinement, all grids were refined up to a certain depth. The amount of grid cells in the complete system can be computed for this example by

$$\sum_{d=0}^{d_{max}} \left(2^d \cdot s\right)^3 = \sum_{d=0}^{d_{max}} \left(2^d \cdot 12\right)^3 \quad .$$

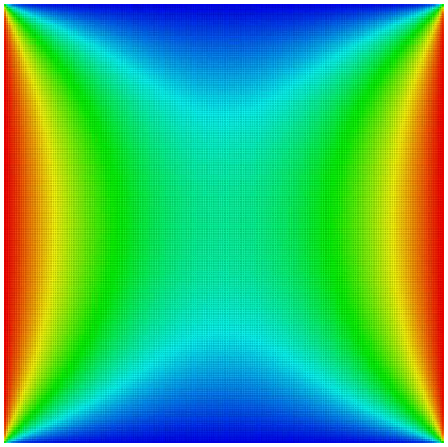
In this case, the total amount of grid cells can be given as follows: depth 4 has approximately 8 million cells, depth 5 has 64.7 million cells, and depth 6 has 517.7 million cells. The cut shown in Figure 5.1(a) is depicted for depth 4 in order to better illustrate the grid refinement. The solution on level 6 is regarded as sufficiently fine and, thus, used as a reference solution.

According to the setting of the boundary conditions, the highest gradients are expected near the walls, thus in the case of an adaptive grid the refinement is increased closer to the boundaries by refining the bisected parts. Figure 5.1(c) shows a grid with a uniform refinement up to level 2 and two consecutive refinements towards the west and east walls. Figure 5.1(b) shows the results computed on that adaptive grid. Hence, such a grid is from now on called adaptive 2+2 or $a2+2$. Close to the west and east walls, the two setups have the same grid size, and are thus comparable.

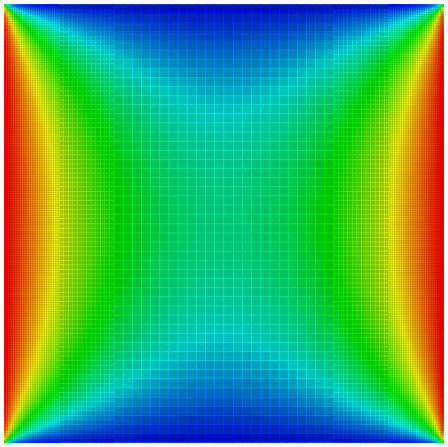
5.2. Distribution Performance. In order to check the distribution for a real 3D problem, the distribution and communication patterns were analysed using the integrated performance monitoring tool IPM[2]. The measurements were performed on 'Shaheen'. Furthermore, some computations were done on the one-rack Blue Gene/P having in total 4096 compute cores at Universitatea de Vest din Timişoara (UVT) in Romania[3].

Figure 5.2 shows two distribution patterns for parallel computations using 512 processes performed on Shaheen. The communication pattern of the $u4$ refinement is depicted in Figure 5.2(a), the $a2+2$ refinement is shown in Figure 5.2(b).

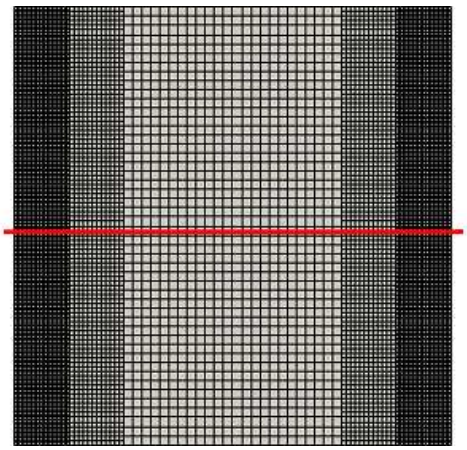
[2]Integrated Performance Monitoring (IPM): `http://ipm-hpc.sourceforge.net`
[3]Universitatea de Vest din Timişoara HPC Center: `http://hpc.uvt.ro`



(a) Uniform Grid Distribution setup at depth 4 ($u4$)

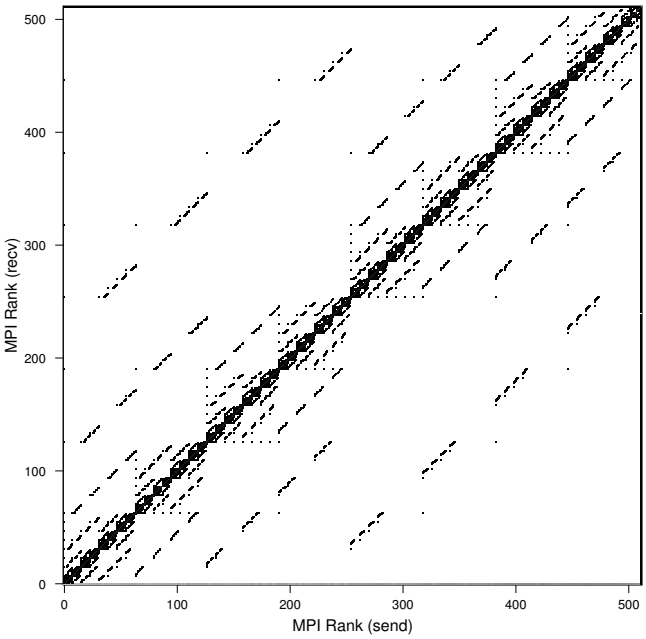


(b) Adaptive Grid Distribution setup at depth 4 ($a2+2$)

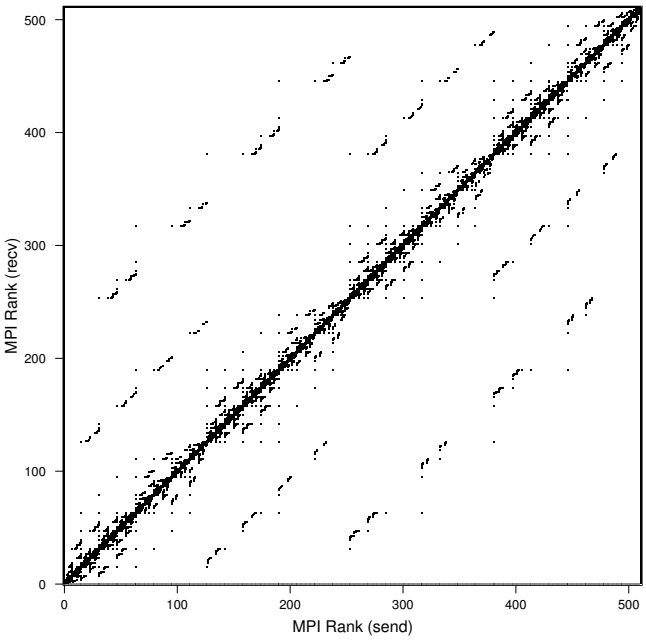


(c) Adaptive Grid Distribution setup at depth 4 ($a2+2$) (grid only) and the cutline (in red) introduced in subsection 5.3

FIG. 5.1. *2D cut through a 3D test problem for testing the adaptive code and distribution.*



(a) Communication Pattern Uniform setup up at depth 4



(b) Communication Pattern Adaptive setup up at depth 4 (2+2)

FIG. 5.2. *MPI communication patterns for 512 processes on Shaheen using IPM.*

Black markers depict a communication between the ranks on the x- and y-axis. In both patterns, the communication with the neighbourhood server residing on rank 511 is obvious and marked completely black, as each process gets its neighbouring information once from the server which will be cached until a change occurs. Otherwise, the main communication is focussed on a diagonal, indicating that only processes close to each other communicate with each other. Independent of the underlying network topology of the respective hardware, this shows that the average number of communication partners per process is small. In case of a non-locality preserving grid distribution, a much higher amount of communication partners could be seen in the communication diagram and would consequently slow down the communication in the system.

It can be observed, that the adaptive grid setup is not differing a lot from the uniform setup, which shows that the data distribution using the selected space filling curve is working as expected by preserving the locality of the data.

5.3. Accuracy Measurements. A first glance at the colour distribution of Figure 5.1 gives already a good hint, that the computations are not that far off and in the same order. To better quantify the error made by the adaptive approach, further treatment is necessary. In Figure 5.3, a quantification of the error is given.

Figure 5.3(a) shows the value for the pressure in the complete domain for different simulation setups. The plot is shown over a cutline going in x-direction through the domain while maintaining y and z at $1/2$. Thus, $x = 0$ represents the west wall and $x = 1$ represents the east wall having each Dirichlet boundary conditions of $p = 1$. On the other walls in the 3D domain, the boundary conditions are set to $p = 0$. As solver, the multi-grid like solver from section 4 is used on the adaptive data structure. To be able to draw 1D results along a cutline, a trilinear interpolation is applied, for interpolating the values along the cutline out of the cell-centred values stored in the model.

In Figure 5.3(a) the general behaviour of the solution is similar. For better quantification, the relative error is plotted in Figure 5.3(b). As reference solution, a linear interpolation of the finest computation of a uniform distribution at depth 6 was used. The relative error is computed by the formula

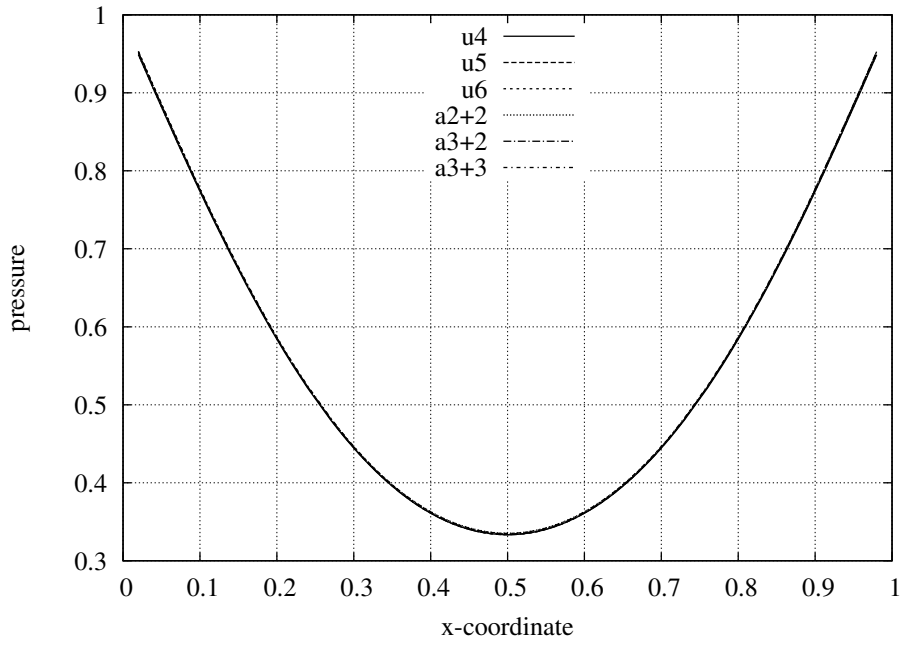
$$e_{rel} = \frac{p - p_{ref,u_6}}{p_{ref,u_6}} \cdot 100 \ [\%] \quad .$$

First of all, uniform grids at different levels were compared, namely level 4 and 5. The results show that the error is larger towards the boundaries than in the middle of the domain. This is due to the fact, that the larger cells cannot capture the higher gradients closer to the wall as good as the refined grids can. In the middle of the domain, where the gradients are small however, the approximation is quite good. Level 4 is worse than level 5, but nevertheless, the computational time and memory consumption are much better. On Sandstorm, the department for Computation in Engineering's own cluster, with four nodes consisting each of two Intel Xeon E5-2690 processors with 8 cores each and a total of 768 GB of main memory, one node using 16 processes needed 6.8 seconds to compute the 8 million cells on level 4 and 98.5 seconds to compute the 64.7 million cells on level 5.

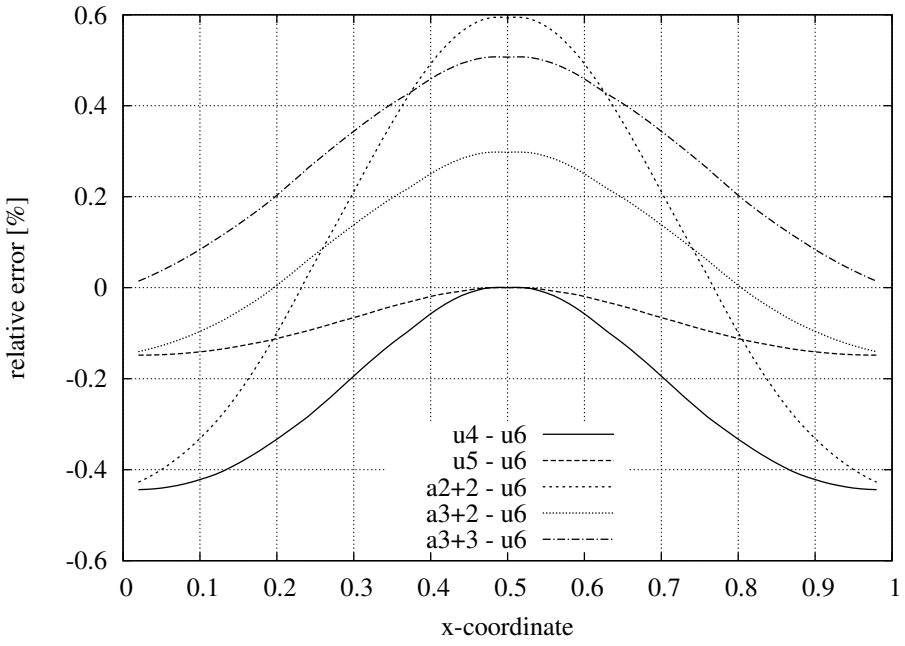
Furthermore, the error from the adaptive refinement with respect to the reference solution can be seen in Figure 5.3(b). Obviously the coarsest grid ($a2 + 2$) has the largest error. Towards the boundary, $a2 + 2$ shows the same error as $u4$ which is not remarkable, as the grids cells have the same size. Towards the middle, the error increases, as the grid is not fine enough to capture effects noticeable on the reference grids. A maximum error of approximately 0.6% can be observed for the $a2 + 2$ curve, which is with respect to all factors such as runtime and memory reduction quite acceptable.

Moreover, it can be noticed that the uniform grids never overestimate the solution of the reference grids, i. e. that the solution never 'crosses' through the reference solution. This is not the case with the adaptive refinement. Here, some solutions cross 'through' the reference solution (i. e. crossing the x-axis).

5.4. Scaling and Performance Measurements. Scaling and performance measurements for uniform grids using up to 32768 processes were already presented in section 4. The following section will provide a few comparative measurements to assess the performance impacts of the adaptive distribution. The measurements were performed on the UVT Blue Gene/P for the same 3D test case described above. The only difference is, that the block size of the data grids was set to $b_i = 4$ instead of 12. Experience suggested that the Blue Gene/P has a much better performance with its PowerPC processors for these chosen settings.

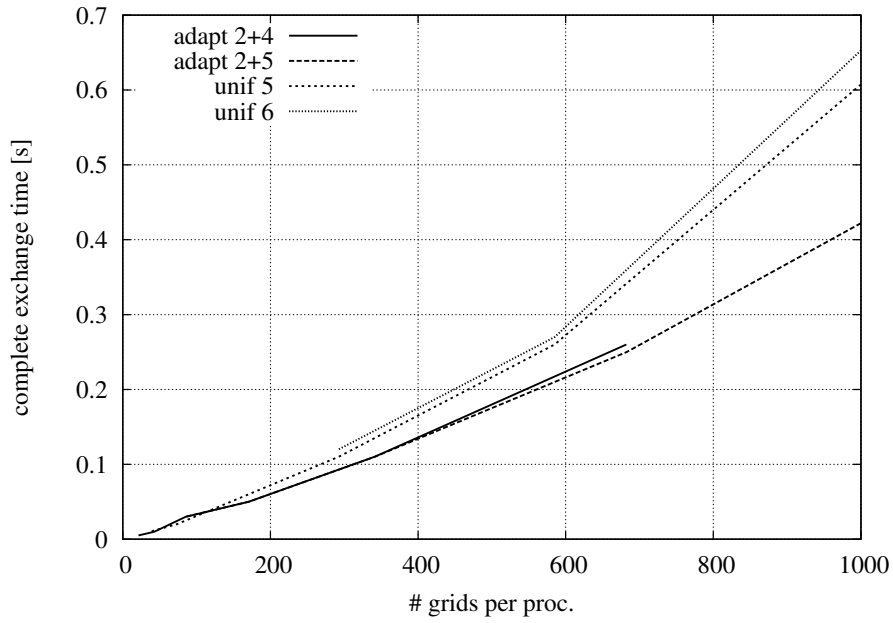


(a) Pressure distribution along the cutline

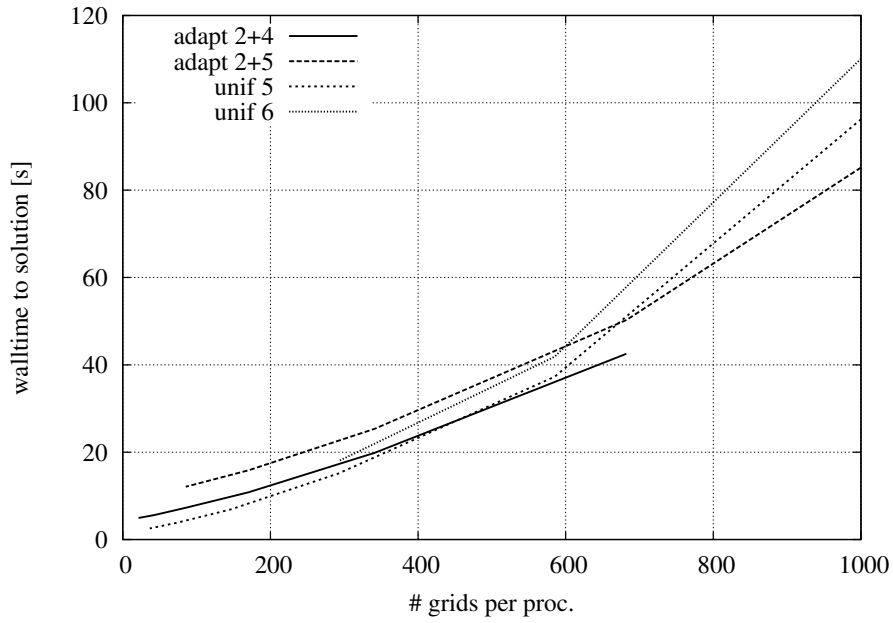


(b) Relative error in % along the cutline

FIG. 5.3. *Data along the cutline in x-direction through the domain at* $y = z = 0.5$. *The geometric position of the cutline is also displayed in Figure 5.1(c).*



(a) Exchange time in seconds plotted over the amounts of grids per process



(b) time to solution plotted over the amounts of grids per process

FIG. 5.4. *Scaling and performance measurements on the UVT Blue Gene/P. A bisection to the given depth was chosen. The 3D test case from section 5 was used with a different data grid size of 4×4×4.*

Figure 5.4(a) shows scaling results for the pure exchange time in seconds of all the necessary variables for the fluid code (3 velocity values, a pressure value, a temperature value, etc). Figure 5.4(b) shows the wall time in seconds until a solution of the Poisson equation converged to an error below 10^{-9}.

The x-axis displays the amount of grids per process, in order to better compare the resulting times of the adaptive and uniform distributions. Computations were performed using up to 1024 processes. Table 5.1 shows the amount of grids in the different configurations.

TABLE 5.1
Amount of grids for different configurations used in the performance measurements

configuration	# grids
adapt 2+4	21833
adapt 2+5	87369
unif 5	37449
unif 6	299593

It can be seen, that the performance of the exchange as well as the solution of the Poisson equation using the multi-grid-like approach is behaving well if less than 600 grids per process are used. On the other hand, a very similar behaviour of the adaptively refined grids can be observed. Hence, it can be concluded, that the adaptive refinements have no negative impact on the overall performance of the code. Contrary, a similar performance can be obtained with less computing power for simulating a domain with nearly the same accuracy but far less grids.

6. Adaptive CFD Example. The implemented CFD code for simulating an incompressible Newtonian fluid flow is currently based on the fractional step method proposed by Chorin [3]. In this approach, an explicit time stepping scheme is used. First, intermediate velocities are computed by omitting the pressure part in the Navier-Stokes equations. Then, using these intermediate velocities, the divergence of the momentum equation is computed leading to a Poisson equation for the pressure. After solving the Poisson equation, the obtained pressure correction is used to correct the intermediate velocity field leading to the velocities at the next time step.

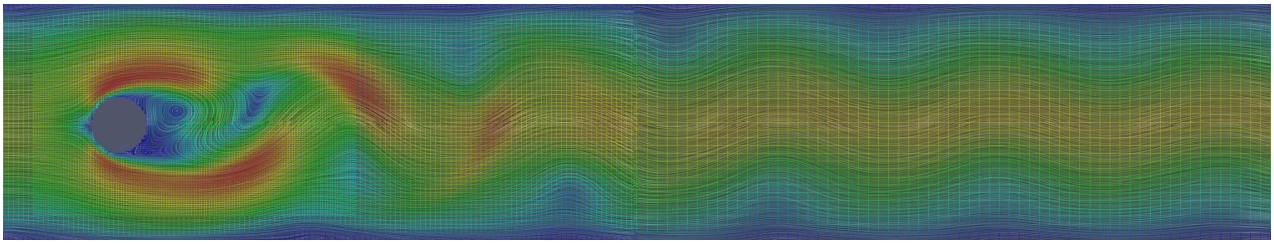
As engineering example of a non-steady-state fluid flow application, the von Kármán vortex street was computed according to the settings of the 2D-2 benchmark of Schäfer and Turek [17]. It is basically a channel flow with a cylinder as obstacle in the channel. After a certain simulation time, vortices start to shed from the disturbances downstream of the cylinder and will be transported out of the region.

Figure 6.1 shows the results of computations for different refinement levels using a Reynolds number of 100. For the $a3+2$ setting, the area consists of 405 adaptively organised grids in a hierarchy going up to depth 5 with an increased refinement around the area where the cylinder is located and the shedding of the vortices starts. $a3+3$ has a more refined resolution around the cylinder using 597 grids. In case the area would be uniformly refined up to level 5, the domain would contain 1365 grids and 5461 grids for level 6 respectively.

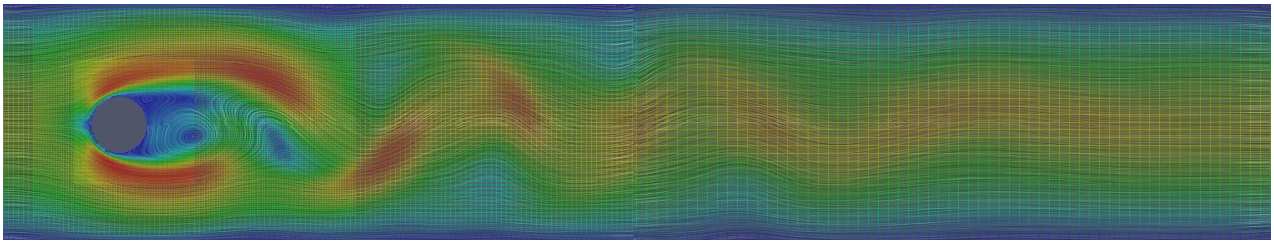
Using a normal desktop computer, a speed up of approximately 7.8 could be achieved by using the $a3+3$ version versus the fully refined $u6$ version (having around 9.1 times more grids), which corresponds also quite nice to the results of the scaling of the solution of the Poisson equation shown in Figure 5.4.

For the 2D-2 benchmark, the adaptive code delivered a Strouhal number ($St = D \cdot f/\bar{U}$, where D is the diameter of the obstacle, f is the frequency of the vortex shedding and $\bar{U}$ is the mean input velocity as defined in [17]) of 0.290 which correlates very good with the experimental results of $St = 0.287 \pm 0.003$. Furthermore, the pressure difference ΔP between the front and the back of the cylinder delivered values between 2.40 and 2.48 depending on the state of the shedding which also corresponds quite well to the benchmark's Table 4 in [17]. These results confirm that the fully adaptive CFD computation as it is currently implemented delivers reasonable physical results.

Although the above presented benchmark is 2D only, the adaptive CFD code is capable of 3D computations. Figure 6.2 shows a fully 3D computation of the 3D-2Z benchmark at Re=20. The adaptive grid refinement has the same settings as the corresponding 2D benchmark.



(a) $a3 + 2$ refinement



(b) $a3 + 3$ refinement

FIG. 6.1. *Adaptively refined von Kármán vortex street according to the Schäfer-Turek benchmark 2D-2 at Re=100 (represented by LIC (line integral convolution) overlaid with the grid refinement. The colour code represents the velocity magnitude. The current time of the two computations is different).*

7. Conclusion and Outlook. In this paper, the authors presented a detailed study about an adaptive layout for a parallel distributed data structure capable of computing fluid flow simulations, as well as the main idea of a multi-grid based solver approach. The accuracy of the adaptive data structure as well as the introduced error was analysed and the influence on distribution and performance was evaluated. It could be shown, that introducing an adaptive layout does not have a negative influence on either performance or distribution, and the error is acceptable if compared to the massive gain in CPU time and memory consumption.

The solution process of the Poisson equation for the pressure correction could be considerably improved by applying a multi-grid-like solver on an adaptive domain. Future work includes further optimisation of the CFD code in order to increase the performance of long time consuming transient computations. Furthermore, the explicit time stepping scheme should be replaced at some point by an implicit scheme which allows larger time steps in the domain. Moreover, run-time adaptivity according to fluid criteria, such as maximal gradients of velocities, can be tackled and incorporated, which will improve the quality of the solution enormously.

Acknowledgments. This publication is partially based on work supported by Award No. UK-c0020, made by King Abdullah University of Science and Technology (KAUST). Furthermore, the authors would like to cordially thank for the support and usage of the Blue Gene/P at Universitatea de Vest din Timişoara (UVT) in Romania.

REFERENCES

[1] M. BADER, *Space-Filling Curves - An Introduction with Applications in Scientific Computing*, vol. 9 of Texts in Computational Science and Engineering, Springer-Verlag, 2013.
[2] W. L. BRIGGS, V. E. HENSON, AND S. F. MCCORMICK, *A Multigrid Tutorial*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd rev. ed., 2000.
[3] A. J. CHORIN, *Numerical solution of the Navier-Stokes equations*, Math. Comp., 22 (1967), pp. 745–762.

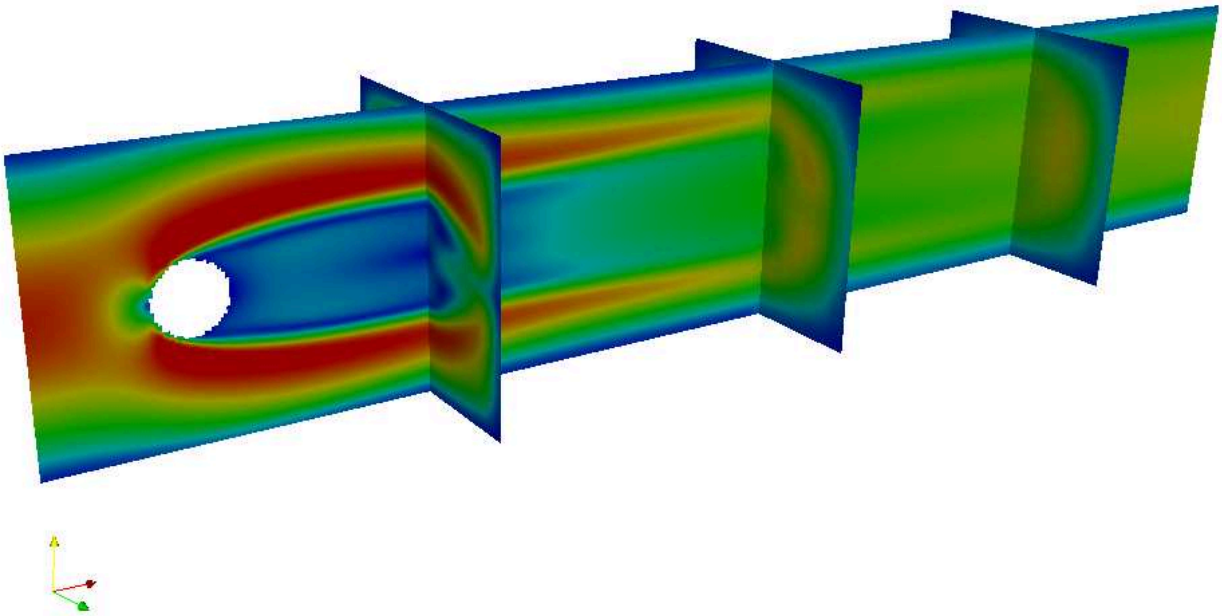


FIG. 6.2. *Adaptively refined von Kármán vortex street according to the Schäfer-Turek benchmark 3D-2Z at Re=20. The colour code represents the velocity magnitude.*

[4] G. CYBENKO, *Dynamic load balancing for distributed memory multiprocessors*, Journal of Parallel and Distributed Computing, 7 (1989), pp. 279–301.

[5] J. H. FERZIGER AND M. PERIC, *Computational Methods for Fluid Dynamics*, Springer, 3rd, rev. ed ed., 2002.

[6] J. FRISCH, R.-P. MUNDANI, AND E. RANK, *Communication schemes of a parallel fluid solver for multi-scale environmental simulations*, in Proc. of the 13th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), Timişoara, Romania, Sept. 26.-29. 2011, pp. 391–397.

[7] ———, *Resolving neighbourhood relations in a parallel fluid dynamic solver*, in Proc. of the 11th International Symposium on Parallel and Distributed Computing, Los Alamitos, CA, USA, 2012, IEEE Computer Society, pp. 267–273.

[8] ———, *Adaptive distributed data structure management for parallel CFD applications*, in Proc. of the 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), Timişoara, Romania, Sept. 23.-26. 2013, pp. 513–520.

[9] ———, *Parallel multi-grid like solver for the pressure Poisson equation in fluid flow applications*, in Proceedings of the IADIS International Conference - Applied Computing, Fort Worth, Texas, USA, 2013, pp. 139–146.

[10] W. HACKBUSCH, *Multi-Grid Methods and Applications*, Volume 4 of Springer Series in Computational Mathematics, Springer, 2010.

[11] H. HADŽIĆ, *Development and Application of Finite Volume Method for the Computation of Flows Around Moving Bodies on Unstructured, Overlapping Grids*, PhD Thesis, Technische Universität Hamburg-Harburg, 2005.

[12] C. HIRSCH, *Numerical Computation of Internal and External Flows, Volume 1*, Butterworth–Heinemann, 2nd ed., 2007.

[13] M. MANHART AND H. WENGLE, *Large-eddy simulation of turbulent boundary layer flow over a hemisphere*, in first ERCOFTAC Workshop on 'Direct and Large-eddy Simulation', Guildford, Mar. 27–30 1994, Kluwer Academic Publisher.

[14] A. MCADAMS, E. SIFAKIS, AND J. TERAN, *A parallel multigrid Poisson solver for fluids simulation on large grids*, in ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA), 2010.

[15] G. M. MORTON, *A computer oriented geodetic data base; and a new technique in file sequencing*, tech. report, IBM Ltd., Ottawa, Canada, 1966.

[16] R.-P. MUNDANI, J. FRISCH, AND E. RANK, *Towards interactive HPC: Sliding Window Data Transfer*, in Proc. of the 3rd International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering (ParEng2013), Pécs, Hungary, 2013.

[17] M. SCHÄFER AND S. TUREK, *Benchmark computations of laminar flow around cylinder*, in Flow Simulation with High-Performance Computers II, E. Hirschel, ed., vol. 52 of Notes on Numerical Fluid Mechanics, Vieweg, Jan. 1996, pp. 547–566.

[18] U. TROTTENBERG, C. W. OOSTERLEE, AND A. SCHÜLLER, *Multigrid*, Academic Press, 2001.

[19] R. VAN LIERE, J. D. MULDER, AND J. J. VAN WIJK, *Computational steering*, Future Generation Computer Systems, 12 (1997), pp. 441 – 450. HPCN96.

[20] T. Weinzierl and M. Mehl, *Peano – a traversal and storage scheme for octree-like adaptive cartesian multiscale grids*, SIAM Journal on Scientific Computing, 33 (2011), pp. 2732–2760.

Edited by: Teodor Florin Fortiş
Received: Feb 28, 2014
Accepted: Apr 16, 2014

DOI 10.12694/scpe.v15i1.965
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 15, Number 1, pp. 49–62. http://www.scpe.org

DEVELOPING SECURE CLOUD APPLICATIONS

MASSIMILIANO RAK, MASSIMO FICCO* AND ERMANNO BATTISTA, VALENTINA CASOLA, NICOLA MAZZOCCA†

Abstract. Today the main limit to Cloud adoption is related to the perception of a security loss the users have. Indeed, the existing solutions to provide security are mainly focused on Cloud service provider prospective in order to securely integrate frameworks and Infrastructures as a Services in a Cloud datacenter. Customer could not monitor and evaluate the security mechanisms enforced by service provider. Service Level Agreements mainly focus on performance related terms and no guarantees are given for security mechanisms. Customers are interested in tools to verify and monitor the implemented security requirements. On the other hand, developers need tools to deploy Cloud applications offering measurable security grants to end users. In this paper, we propose an approach to implement security mechanisms as components in the application design process. We modeled security interactions according to the specific threat, the specific security requirements and user/application capabilities trying to improve security. It enables a Service Provider to offer security guarantees to customers. The approach has been designed to fit with different Cloud platforms, but to demonstrate its applicability, we will present a case study on the mOSAIC Platform.

Key words: Cloud security, security engineering, secure application design

62

1. Introduction. The most desirable Cloud feature is the self-service on demand approach. Cloud customers are able to access and use Cloud services without interacting with system administrators. Resources and services are charged on a pay-per-use base. The clear advantage for customers is the chance to have access to virtually infinite amount of resources, paying for them only when effectively used. As a side effect, end users do not own a real infrastructure: servers and data reside in the Cloud. In such scenario, the main limit to Cloud adoption is related to the perception of a loss of security the users have: is it possible to control where data reside? Are the customers granted about who can access their data? Is the provider reliable? Cloud customers can buy resources from different providers on the basis of the grants they offer, usually expressed in terms of Service Level Agreements (SLAs), i.e., contracts among the customer and the provider that specify what is granted [1, 2]. At state of art, even if clear innovative solutions exist, like the one proposed in the FP7 projects (SLA@SOI [3, 4], Contrail [5], mOSAIC [6]) or with standard languages like WS-Agreement ([7]), SLAs are commonly expressed in natural language and usually focus on performance related terms.

Existing solutions to provide security are mainly focused on Cloud service provider prospective in order to securely integrate frameworks and Infrastructures as a Services (IaaS) in Cloud datacenter. So, they focus on how to enable a Cloud service provider to offer security grants. Up to date very few results are available that consider end customers' perspectives. However, customers are interested in tools to verify and monitor the implemented security requirements. Nevertheless, application developers should be taken in consideration, too. They need tools to deploy Cloud application offering measurable security grants to end users [8, 9].

In this paper we will focus on these latter perspectives. In particular, we propose a methodology that enables the correct usage of a Platform as a Service (PaaS), in order to develop applications which clearly expose security grants to end users. The methodology has been designed to fit with different Cloud platforms, but to demonstrate its applicability we present a case study on the mOSAIC Platform [6].

The remainder of the paper is organized as follows: the next section aims at defining the problem of Cloud application security, outlining actors and roles. Section 3.1 describes the technology we focused on for application development and introduces a simple application we will use as a running example. Some related works are presented in Section 3.2. Section 4 describes the proposed methodology to take into account security requirements in Cloud application design. Section 5 shows how to apply the methodology. Finally, in Section 6 some conclusions and future work are presented.

2. Problem Description. The Cloud Computing paradigm implies to move from a local ownership of resources, services and infrastructures to a different approach in deploying, accessing and managing them. This leads to the multiplication of interactions among providers and Cloud users (actors), involved in the system and, consequently, to the perception of *loss of control* over data and offered services.

*Department of Industrial and Information Engineering, Second University of Naples, Aversa, Italia
†Department of Electrical Engineering and Information Technologies, University of Naples Federico II, Napoli, Italia

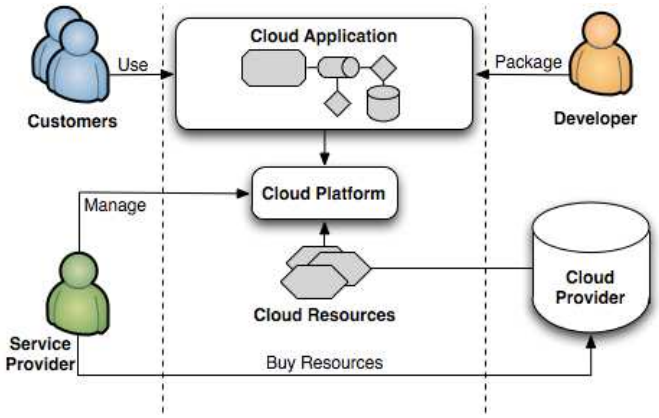


FIG. 2.1. *Actors and interactions*

Figure 2.1 briefly describes the interactions among Cloud actors. In particular, we identified three main actors: the *customer*, which uses Cloud applications and has specific security requirements; the *Cloud service provider* that deploys applications over Cloud resources, and provides application features to customers; *developer* that builds up the applications, and delivers them in terms of software packages offered to the service provider. Moreover, resources may be hosted over independent *Cloud providers*, which may sell them to the service provider under different conditions and with different guarantees. Service providers can then negotiate with their customers the adoption of such resources and services. In such context, a *Cloud application* runs over a *Platform* and it is offered in terms of a *Cloudware* (i.e., a software package that can be executed over Cloud resources in order to offer a platform able to run Cloud applications).

Customers have security requirements that should be granted by the service provider, but many security issues can arise in the described service supply chain. The service provider's task is to grant the overall security for the offered applications, but: (i) it uses resources bought from an independent Cloud provider, and (ii) it offers applications developed by third-party *developers*. Both Cloud providers and developers can enforce proper security mechanisms, but the interaction among them may not be secure at all.

We can summarize the problems to address, as follows: the customers have their security requirements, the developers have to implement them and service providers have to guarantee them in spite of the complex supply chain that can be effectively used. Therefore: (1) developers need to understand the requirements and enforce them into the application; (2) Service providers should be able to control the applications and their security features in order to guarantee them; (3) Customers need to negotiate and monitor their enforcement.

2.1. Contribution. Our first goal, presented in this paper, is to propose a methodology that enables developers to map user security requirements into security mechanisms to enforce; the enforcement is performed by adding security components to existing applications in the design phase and taking into account user requirements and possible application threats. Our long term goal, in future work, will be to enable the *automatic* negotiation, evaluation and enforcement of security features on developed applications.

In order to illustrate the approach in the practice, we focus on the mOSAIC Platform [10, 6], as the enabling technology for development of Cloud applications. The next section briefly introduces the mOSAIC Platform and shows how to develop an application using the mOSAIC APIs. We will use this example to outline the steps of the application development and how to improve security during the design stages.

3. Background and Related Works.

3.1. Developing applications with the mOSAIC Framework. The mOSAIC platform aims at providing a very simple way to develop Cloud applications [6, 10]. The target user for the mOSAIC solution is the application developer (*mOSAIC user*). In mOSAIC, a Cloud application is structured as a set of components running on Cloud resources (i.e., on resources leased by a Cloud provider) and able to communicate with each

other. Cloud applications may also be provided in the form of Software-as-a-Service, and can be accessed/used by users and by the mOSAIC developer (i.e., by *final users*).

Developing a mOSAIC application is very simple as it is built up as a collection of interconnected *mOSAIC components*. Furthermore a wide set of components is available. Components may be (*i*) core components, i.e., predefined tools offered by the mOSAIC platform for performing common tasks, (*ii*) COTS (commercial off-the-shelf) solutions embedded in a mOSAIC component, or (*iii*) *Cloudlets* developed using the mOSAIC APIs and running in a *Cloudlet Container*. mOSAIC Cloudlets are stateless, and developed following an event-driven asynchronous approach [10].

Among ready-to-use components it offers: (1) queuing systems used for component communications (*rabbitmq* and *zeroMQ*), (2) an *HTTP gateway*, which accepts HTTP requests and forwards them to application queues, and (3) NO-SQL storage systems (as Key-Value stores and columnar databases). mOSAIC components run on a dedicated virtual machine, named mOS (mOSAIC Operating System), which is based on a minimal Linux distribution. The mOS is enriched with a special mOSAIC component, the *Platform Manager*, which makes it possible to manage set of virtual machines hosting the mOS as a virtual cluster, on which the mOSAIC components are independently managed. It is possible to increase or to decrease the number of virtual machines dedicated to the mOSAIC application, which scales up and down automatically.

A Cloud application is described as a whole in a file named *Application Descriptor*, it lists all the required components and Cloud resources. A developer has both to develop new components, and to write the application descriptor to connect them together.

3.1.1. mOSAIC application example. To illustrate our work, in this paper, we refer to a simple Cloud application: an XML document manager (XDM) that analyzes the incoming XML files. For each received file, it counts the number of occurrences of XML tags inside the file and stores the results as a pair $<filename, tagcounts>$ (where *tagcounts* is a collection of $<tag, count>$ pairs) into the Key-Value store (KV).

The application consists of several components, which interact according to the sequence diagram shown in Fig. 3.1: an $HTTPgw$ Cloudlet manages the HTTP messages, and pushes the XML document (XML_Doc) into a queue; the XML *Analyzer* extracts and parses the queued XML documents, and stores the results (Res) in a KV store.

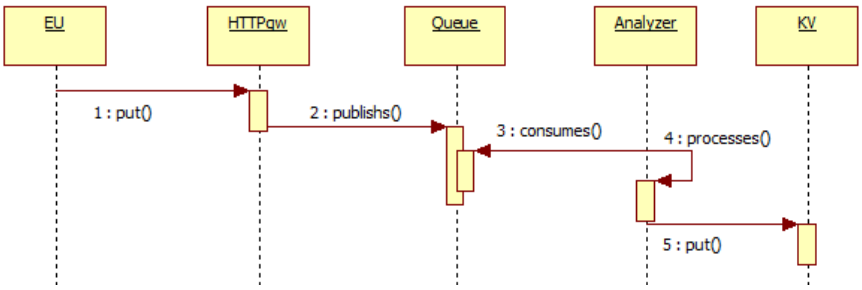


FIG. 3.1. *XDM component interactions*

Fig. 3.2 shows the architecture of such mOSAIC application, outlining the role of the different actors identified at the beginning of this section and different application components (Httpgw, Queue, Analyser and KV store).

Such application does not provide any security features and it is subjected to a number of attacks and threats. As an example, it may be subjected to XML Denial of Services attacks and it cannot guarantee mutual authentication among users and the application. The application developer well knows the components and how to connect them thanks to the application descriptor. If the developer knows - at this stage - the user requirements, is he able to enforce proper security mechanisms? Is there the possibility to design a methodology that help him in identifying and enforcing correct security solutions? At this aim, we propose an approach to implement security mechanisms as components to add in the design process. We can design

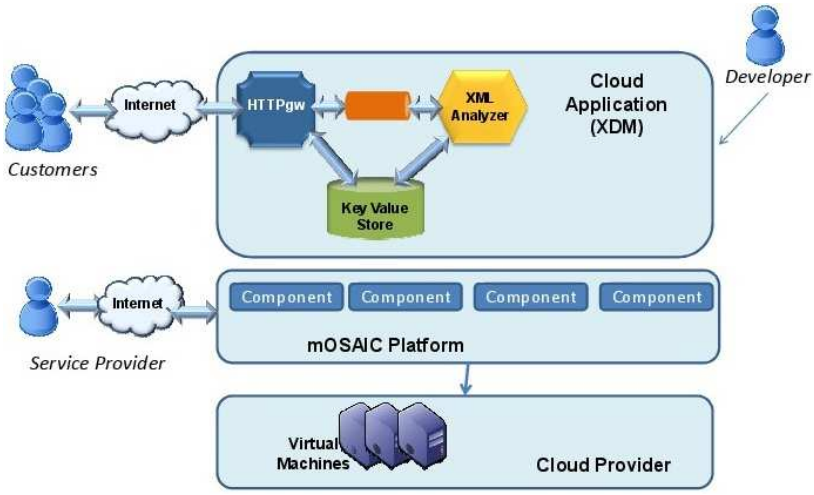


FIG. 3.2. *The XML Analyzer mOSAIC application*

component interactions according to the specific user security requirement, to its security capabilities and to known threats, trying to connect such security components with application/functional specific components.

3.2. Related Work. The protection of cloud resources is still an open problem and different approaches are available in the literature. In particular, several works propose specification languages to specify security attacks and requirements.

Abstract State Machine Language (AsmL) is a software specification language developed by Microsoft, defining abstract state machines through a fairly high-level language that can be compiled into executable code. Raihan et *al*. [11] propose a formal extension of AsmL, which describes how to model attacks, and present the design and implementation of a compiler that generates attack signatures from the attack specifications. Zulkernine et al. [12] present a method for integrating AsmL and Unified Modeling Language (UML) state charts, in order to extend finite state machine based software specification languages, with the Snort open source Intrusion Detection System (IDS). The attack scenarios are specified in AsmL or UML state charts, and then, automatically translated into Snort rules. The Snort's detection engine is modified so that it can understand the rules translated from AsmL or UML state chart scenarios.

Hussein et *al*. [13] present a framework for developing components with intrusion detection capabilities. This framework uses *UMLintr*, an UML profile for intrusion specifications [14]. The profile allows developers to specify intrusion scenarios using UML diagrams. Specifying intrusion scenarios, using the same language that is used for specifying software behavior, eliminates the need for separate languages for describing intrusions. UML diagrams used in UMLintr are called *misuse-case* and *misuse-state-machine* diagrams. Misuse-state-machine diagrams are used to specify the detection of intrusion scenarios. Each misuse-state-machine represents the states which its corresponding attack may go through.

STATL is a language to describe security violations as sequences of actions that an attacker performs to compromise a computer system [15]. The STATL language has been successfully used in describing both network-based and host-based attacks, and it has been tailored to very different environments. By abstracting away from the details of a particular attack, it is possible to detect previously unknown variations of an attack or attacks that exploit similar mechanisms. In particular, the STATL language provides constructs to represent an attack as a composition of states and transitions.

CORAS [16] is a method for conducting security risk analysis, it presents a customized language for threat and risk modeling, and provides detailed guidelines explaining how the language should be used to capture and model relevant information during the various stages of the security analysis.

The main difference between the proposed approach and the above described solutions, is that, our solution follows a *bottom-up* approach, focusing on the application architecture and implementation, having has target

user the developer. In particular, the previous works mainly propose formalisms to specify attack and intrusion models, as well as security rules. Our approach aims at proposing a methodology to describe mechanisms as components in the cloud-based application design process, which can be used to enforce security requirements specified in SLA.

4. Modeling Secure Cloud Interactions. Traditional techniques for protecting systems have almost clear boundaries and rely on a permanent architecture, which is *hardened* through the introduction of ad-hoc security mechanisms. This is not the case for a Cloud application: it runs over an enabling platform, which consumes multiple Cloud resources dynamically acquired. In order to design an application which offers clear grants to *Customers*, grant for which the Provider is responsible, we need a way to state such requirements and a way to evaluate how they are offered.

The approach is based on the following logical steps:

1. developers identify application use cases according to a common software development methodology;
2. developers identify the interactions (i.e., every kind of data exchange among actors in a Cloud system) among components of the Cloud application under study (next sections will detail what we mean with the *interaction* terms) and describe the use cases as a workflow of different *interactions*;
3. any interaction can be implemented and offered with different modalities, taking into account security mechanisms, threats, user requirements, user security attributes and capabilities.

Basing on these assumptions, we propose to develop specific security components that are available to developers, pre-associated to user requirements and pre-evaluated in terms of security guarantees. In this way, a developer can easily add security mechanisms to their applications and, furthermore, he will be able to control the security level that he is enforcing. The pre-evaluated components also enable the Service Provider to evaluate the goodness of a secure application and have a tool to guarantee this to end-users.

In order to demonstrate this, we will introduce the main concepts of *modeling interactions* and *interaction modalities* by using an Infrastructure as a Service example, then we will apply them on a real case study, based on a Platform as a Service (the mOSAIC Platform) to develop the security components.

4.1. The proposed approach. Several Cloud providers, such as Amazon EC2, Google Compute Engine, GoGrid, offer similar services and applications to customers, but they use different technologies and techniques to guarantee security grants to end users. Our approach is to offer to developers and customers a way to identify, independently of the technologies, the features they are interested in terms of components and additional security components to enforce security requirements expressed by end-users. We introduce a model that describes Cloud applications in terms of interactions and the ways in which such interactions take place (modalities) to implement specific Use Cases. Moreover, we took into account the roles' interactions that may occur in a Use Case.

4.1.1. Cloud actors. The actors classification introduced in the first section is specialized according to their capabilities to perform an action inside or outside the Cloud:

- EU: End Users (the customers) that interact with the Cloud by means of proper Web or software interfaces.
- CSP: Cloud Service Provider.
- S: Services that provide access to applications, resources and features.
- R: Resources located in the Cloud.

Each actor can assume different roles being both providers or requester of services. According to the specific interaction with the system, the roles we located are:

- INVOKER: It initiates the interaction and provides all the necessary information to process.
- TARGET: It is the interaction object.
- PROVIDER: It provides and manages the target. It is responsible for the accessing mechanisms both in terms of functionalities (such as interfaces) and security (e.g., access control lists, policy).

4.1.2. Actor roles. Even if actors can assume any role, generally, the EU acts as an INVOKER, the R acts as a TARGET, while CSP and S assume all the three roles. The general interaction schema among actors is

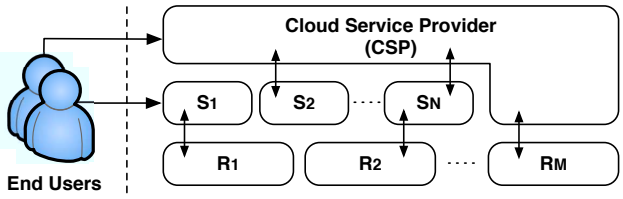


FIG. 4.1. *Interactions among actors*

reported in Fig. 4.1. For instance an EU can access a resource either using an ad-hoc service or through a CSP interface. Moreover, a CSP can offer a resource using direct access or by a service. The EU interacts with the Cloud resources and services in order to use an available *Use Case*. The term Use Case refers to the application features that an Invoker requests to the Provider. An Use Case implementation requires several interactions that can be implemented in different modalities. Therefore, we will model an Use Case as a workflow (sequence) of interactions. Each an *interaction* is characterized by the triple:

<INVOKER, PROVIDER, TARGET>.

Examples of valid interactions in which an EU has the Invoker role are reported in Tab. 4.1. Such examples help in clarifying what we mean with an *interaction Type*.

TABLE 4.1
Examples of valid Interactions Types

n.	Use Case description	Interaction Type
1	First EU's registration to CSP	<EU,CSP,CSP>
2	EU mounts a disk to an instance	<EU, S, R>
3	EU creates a virtual volume	<EU, S, S>
4	EU generates a new set of credentials for SSH	<EU,CSP,S>

The first example focuses on the actions performed by an EU when he registers himself to the Cloud provider. In this case, we outline that the *End user* acts as *Invoker*, while the CSP as *Provider*. Note that, the *Target* is the CSP itself: the interaction has the effect of changing the set of users of the CSP, enabling the EU to accept future requests for different services. From a security point of view, this operation is critical and the security mechanisms adopted to protect such interaction will be very relevant. We will take into account these mechanisms in the *Iteraction Modality*.

4.1.3. Iteraction modalities. CSPs can offer the same Use Case by several modalities. In Tab. 4.2 different examples of Interaction Modalities are associated to the same interaction. The Use Case "First EU's registration to CS" can be implemented with a Web Console interface and can require a user and password authentication mechanisms, or a more secure One Time Password mechanisms; these two are equivalent from the functional point of view, but provide a different security level being the second more secure.

Each modality can be associated to a different security mechanisms, which depends on threat related to the specific interaction modality, on user requirements, on his attributes and capabilities. Finally, different modalities can provide a different security level that is pre-evaluated. The evaluation of the security level is performed by adopting an evaluation methodology as the ones presented in [17, 18, 19]. Therefore, we can characterize an *interaction modality* by <Threat, Security Requirements, Security Mechanisms, Invoker's Attributes>, where:

- threats are specific attacks or systems vulnerabilities that we want to mitigate;

TABLE 4.2
Examples of valid Interactions Modalities and Security Level

Interaction Type	Interaction Modalities	Security Level
<EU,CSP,CSP>	Web Console with usr/pwd	2
-	Web Console with OTP	3
<EU,S,S>	Web Console with usr/pwd	1
-	Web Console with OTP	2
-	HMAC SHA1	3
-	HMAC SHA2-256	4

- security requirements are user desiderata that usually are requested in the Service Level Agreement with the provider;
- security mechanisms can be enforced with both additional components or with a specific implementation of an already existing component; for example, to improve security in the communication channel, one can use components that already have cryptoghaphic primitives to encrypt the channel or, alternatively, one can add external components to provide this feature;
- invoker attributes are needed to specify invokers security capabilities, if any.

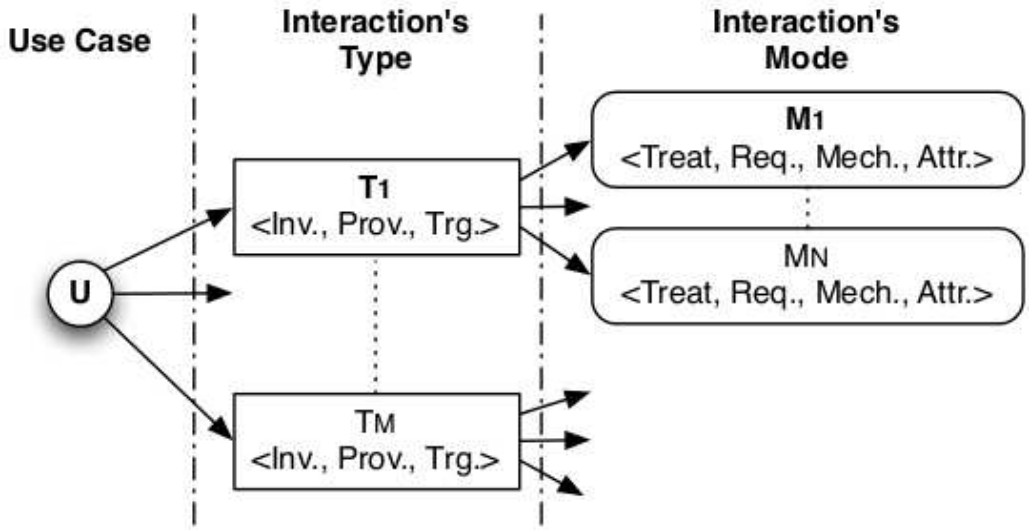


FIG. 4.2. *Modeling a Use Case with interactions*

From the developer point of view, this model helps to see a Cloud application as a composition of n different Use Cases implemented by a number of interactions T_i. In Fig. 4.2 the Use Case representation model is reported. Interactions T_i can be implemented in different modalities M_j that enforce different security mechanisms and are able to guarantee different pre-evaluated security levels. The developer has the possibility to choose one of the available application packages that best fit user requirements and provider guarantees.

Specifically, the approach will enable to add security related terms in the Service Level Agreements among Customers and Service Providers being able to easily evaluate and enforce proper security mechanisms.

4.2. Security mechanism description. Some security mechanism to be adopted can be defined by the security (detection) rules that it has to implement. A rule is specified by three basic elements: Condition, Attributes, and Reaction.

- *Condition* A rule is satisfied if its condition is true. A condition is made of a set of events. An event can be any significant occurrence detected by a probe, such as a particular string contained in a received packet, a predetermined number of messages received from the same IP or a high number of failed login attempts. It is also possible to set up a hierarchy of rules. In this case, a condition can be made of multiple rules too. If all the rules are true and the events are occurred, then the condition is satisfied. Events and rules can be combined with AND, OR, NOT logical operators.
- *Attributes* Every rule may have some attributes too. Designer has the possibility to specify the involved security probes or the monitored resource. For example, it is possible to specify the variable or the log file considered by the rule condition. It is also possible to include the eventual security protocol involved in the rule.

- *Reaction* If the specified condition occurred, the rule is true and the system may be under cyber attack. Thus, when all the events and the rules in a condition are true, an appropriate reaction is performed. The action to take is different basing on the events occurred. Actions depend also on events severity level. For example, an alert e-mail can be sent to the administrator if a low severity event occurs. Instead, in the case of a critical event extreme countermeasures must be taken.

Figure 4.3 shows an example of a rule representation. The stereotype 'Rule' characterizes the object and a numerical ID uniquely identifies the rule. The object is divided in three sections. The first one contains a short rule description and a list of all its optional attributes. In 'Condition' there are events and rules that make true the rule considered. The rule is valid if at least one of the conditions is verified. Each event must be associated to an ID and should be described in a separately table. 'Reaction' contains a short description of what the system has to do. It may contain a list of operations too.

5. A Case Study: Securing mOSAIC Applications. In order to validate the proposed approach we consider the application described in Section 3.1. We assume that the application offers two use cases:

- UC_P: a parsing service of the XML documents (XML_Doc), i.e., it performs the XML analysis (using a DOM approach), and computes the total number of nested XML tags;
- UC_Q: a query service for retrieving results (Res).

In this case study, we consider two kinds of Denial Of Service threats, the HTTP flooding and the Deeply-Nested XML:

- The HTTP flooding aims at creating a plenty of requests to the XDM application in order to exhaust resources and to inflict a denial of service to legitimate EUs.
- The Deeply-Nested XML exploits the XML message format by inserting a large number of nested XML tags in the message body. The goal is to force the XML parser (the Analyzer), to exhaust the computational resources of the host VM, by processing a large number of deeply-nested XML tags [20]. Moreover, this kind of attack is particularly harmful when using DOM based parser. A DOM based parser reads the complete message and builds an in-memory representation (called DOM tree), that is much larger than the message itself. Therefore, the application memory would be exhausted before the validator could even start the validation process.

For this case study, an advantage of using mOSAIC is the ability of automatically scale the application components (the Cloudlets and queues) when the host resources are overloaded (because the attack). But, as a side effect, the application consumes an increased amount of resources, on charge of the application owner (the Service Provider). The Provider needs proper security mechanisms in order to mitigate the considered attacks, as well as to guarantee the signed SLA. In next section we will illustrate how the developer can add such security mechanisms within the already deployed application, by simply adding the set of already available Cloudlets that act as mitigation means.

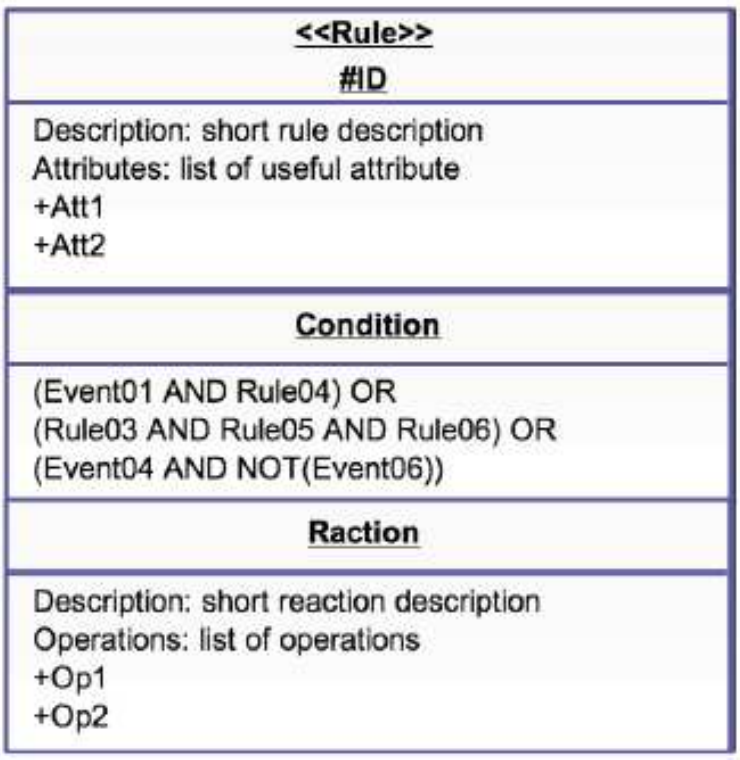


FIG. 4.3. *Example of rule representation*

5.1. Approach implementation. To understand how to apply the proposed methodology, we need to understand how to map user requirements with specific security mechanisms to add in the application. For this reason, to express user requirements, we will adopt the same tuple as described in the previous section, in this way it will be easier to identify additional components for the application developer. Fig. 5.1 expresses the security requirements and attacks to mitigate (they can be reported in a Service Level Agreement with the adoption of the same tuple [21]). In particular, they state that the User wants to protect the interaction T_P from both the attack types, and T_Q by the HTTP flooding. During the specification of requirements, the protection mechanisms and security attributes are not explicitly indicated and the values in the tuple are zero.

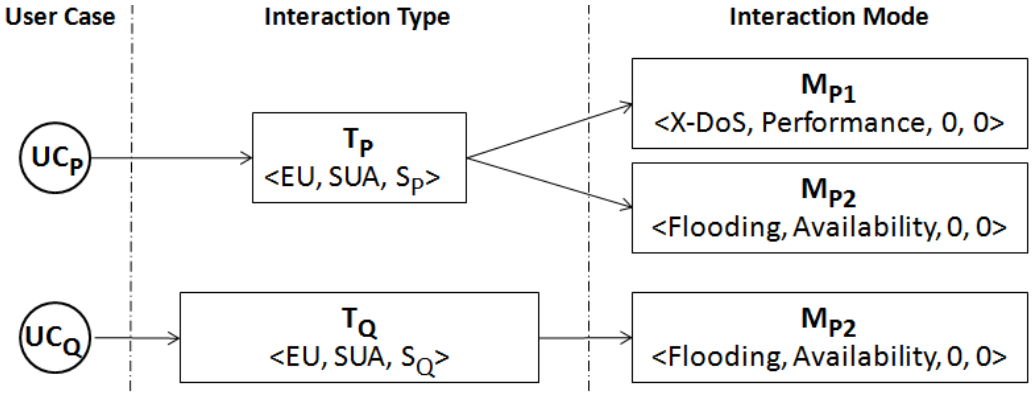


FIG. 5.1. *Security requirements expressed with the same formalism as interaction types*

According to these requirements, in the design phase, in order to protect the application from the considered

attacks, the developer has to identify/choose appropriate mitigation means (i.e., the modalities) for reducing or eliminating the potential intrusions. Specifically, for each interaction between the application components (in terms of INVOKER and PROVIDER), the developer has to identify the vulnerabilities that can be exploited by the specified attacks and, for each attack, he has to define/choose the mitigation means to be implemented, in order to mitigate or eliminate attacks exploiting an interaction vulnerability.

Therefore, the overall approach consists in mapping the high level Interaction Modes M_{p1} and M_{p2}, which express security requirements, with the low-level interactions among application components, as defined in the Service Descriptor of mOSAIC [22]. Moreover, for each interaction type, the developer has to identify vulnerabilities, possible mitigation means, needed user attributes to eventually enforce one mechanism (Can the user manage security tokens? Can the user provide a secret password?).

Looking at the sequence diagram represented in Fig. 3.1, the Use Case UC_P is implemented with four interactions (Fig. 5.2): $\langle$EU, HTTPgw, put$\rangle$, $\langle$HTTPgw, Queue,
publish$\rangle$, $\langle$Analyzer, Queue, consume$\rangle$, $\langle$Analyzer, KV, put$\rangle$ and, for each of them, the possible Interaction Modes with associated vulnerability are reported. For the sake of simplicity, in figure we just reported the threat and the requirement without reporting the mechanisms and user attributes that we will illustrate later together with a description of the located threats.

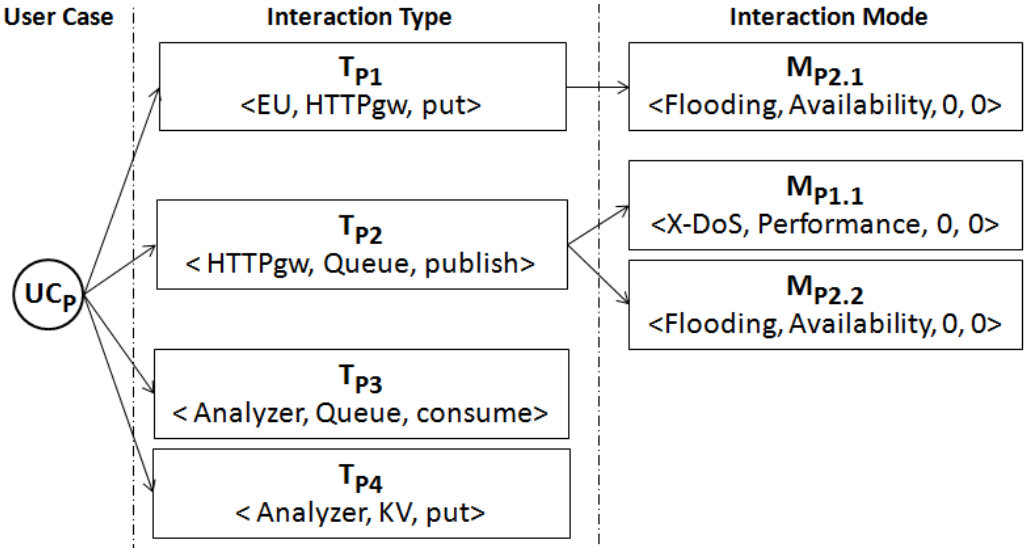


FIG. 5.2. *Use Case P: interaction types and Modes*

5.1.1. Interaction Mode implementation. As for the Interaction Mode $M_{p1.1}$ related to X-DOS threat, the schema validation is a good countermeasure against attacks based on XML messages that are not conform to the Web service description. These schema can be enforced by validating the incoming messages against specific restrictions [23]. Sophisticated solutions can be implemented by restricting the length of each XML element or limiting the list of XML elements contained within each message. However, although efficient XML validation engines (that can operate completely on the fly) have been proposed [24], in the current Web server frameworks, they are not used or not activated by default. This is mainly due to performance reasons, since schema validation is expensive regarding CPU load and memory consumption. This mechanism can be used to verify the presence of an attack, and to identify the malicious flows. If an attack is detected, a reaction can be triggered to filter malicious requests, or to block the attack sources [25].

The detection rule consists in monitoring the overall CPU consumption of the target application deployed over the distributed VMs. We can adopt a window-based state monitoring that triggers state alerts only when the normal state is continuously violated for a time window W [26]. W is essentially the tolerable time in abnormal state, which must be established by the service provider. Given the threshold T_R, the size W of the

monitoring window, and n monitored VMs with CPU values $CPU_i(t)$ at time t, an alert is triggered only when $\sum_{i=1}^{n} CPU_i(t-j) > T$, $\forall j \in [0, W]$ at any t. In order to validate the adopted technique, we assume that the alarm must be triggered when the overall CPU load on the involved VMs exceeds 80% for a time window $W = 10$ minutes. The considered rule is represented in Figure 5.3.

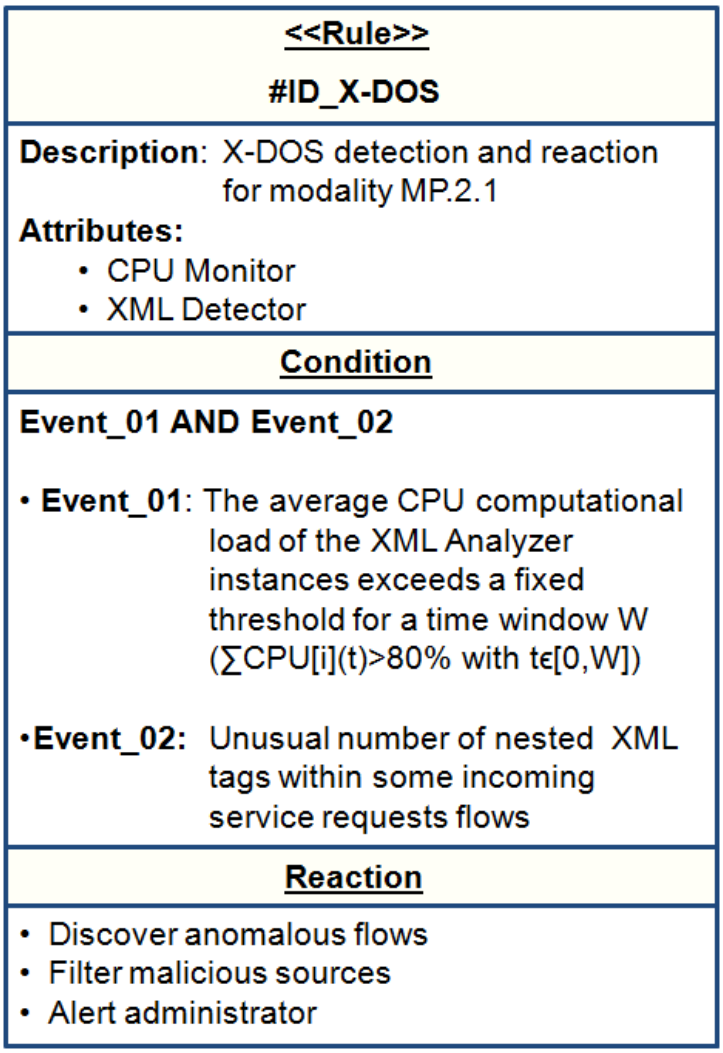


FIG. 5.3. *Detection rule description for modality* $M_{P1.1}$

As for the Interaction Modes $M_{p2.1}$ and $M_{p2.2}$ related to HTTP flooding threat, a different consideration can be done. In the process of visiting a Web site, the user sends requests, which have different properties in idle time, memory computational and computational load. On the contrary, the HTTP flooding sources send out HTTP requests with another pattern. In particular, several known worms' attack pattern use parallel threads to send GET requests in order to exhaust the server's resource. They might use different numbers of threads or different delays, but it's hard to simulate human user's request patterns. Therefore, the fact that the human user and attacker have different request patterns can be used in detecting the attackers, for example, by exploiting the idle time between each request from the same user [27].

5.1.2. Interaction Mode enforcement. In order to implement the identified solutions, we used specific mOSAIC components that are available and can be added by the Service Provider in a transparent way with respect to the application. For example, as for the X-DoS attack the following additional security components

can be used to implement the iteration modality:

- *Monitor*: By using the features offered by the mOSAIC Platform, the Monitor monitors the CPU consumption of the application. In particular, if the average computational load of the XML Analyzer instances exceeds a fixed threshold for a long time period, the Monitor enables the Detector.
- *Detector*: It consists of several sub components: the *Parser*, the *Validator*, and the *Engine* [8, 28]. The first two parse and validate the incoming messages, by using a schema validation approach (e.g., it is verified whether a hunge number of XML tags included in the incoming messages is below a fixed threshold). The Engine uses an anomaly based methods to identify the malicious sources (e.g., for each flow of messages, it counts the number of malformed XML documents received by the application). If a flow of requests is identified as malicious, it sends a message to the HTTPgw, reporting the malicious source of the messages (IP address).
- *HTTPgw*: It is enabled to operate as a filter. It discards the messages flows that are identify as malicious. This action is applied until the system performance returns to a normal state (e.g., the average CPU consumption of the observed Cloudlets is below the fixed threshold).

Fig. 5.4 shows how the architecture of the identified solution changes respect to the original one in Fig. 3.2, with all connected components to enforce the chosen interaction modalities.

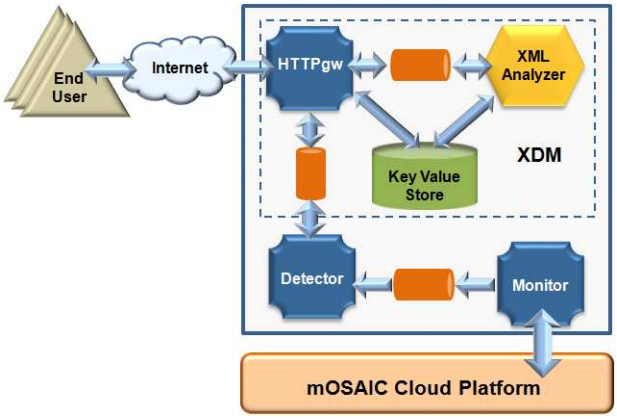


FIG. 5.4. *Application components to enforce chosen Interaction modes*

According to the proposed solution, the validation process of the incoming messages is enabled only when an attack symptom is detected. Moreover, in order to prevent resource exhaustion, the Detector works on an event-based approach, using a SAX parser. For each XML document parsed, an event is sent to the Validator, which directly operates on this event to validate the document. There is no need, neither for the Parser nor the Validator, to reconstruct the whole document in memory. In fact the Validator has constant memory consumption (only depending on the schema size) and linear run-time. The Detector can therefore easily process very large documents. If the Validator finds a schema violation inside a message, the Detector has read the document only up to that particular element, the remaining document is not read and therefore there is not impact on the Detector performance. If the Engine detects a potential attack, the message source is tagged as malicious, and a policy is set on the HTTPgw to discard messages from that source.

As for the HTTP flooding, two interaction modes can be enabled, but they cover different requirements (Fig. 5.2). The first mode can be used to prevent the attack. In particular, secure sockets layer (SSL) and transport layer security (TLS) encryption protocol can be used to secure Web communications via HTTPS. In particular, we can assume that a flooding attack is likely in progress, if the XDM application is overloaded with HTTP requests to be processed, and a huge number of incoming HTTP requests are discarded. In such a case, a detection method must be enabled in order to identify and filter malicious sources. The second mode can be applied for detecting and mitigating the attack. The associated detection rule is represented in Fig. 5.5

The implemented solution is similar to that presented in Fig. 5.4. By using some mOSAIC Platform features, a Monitor can monitor the state the XDM application's queue, as well as the number of incoming

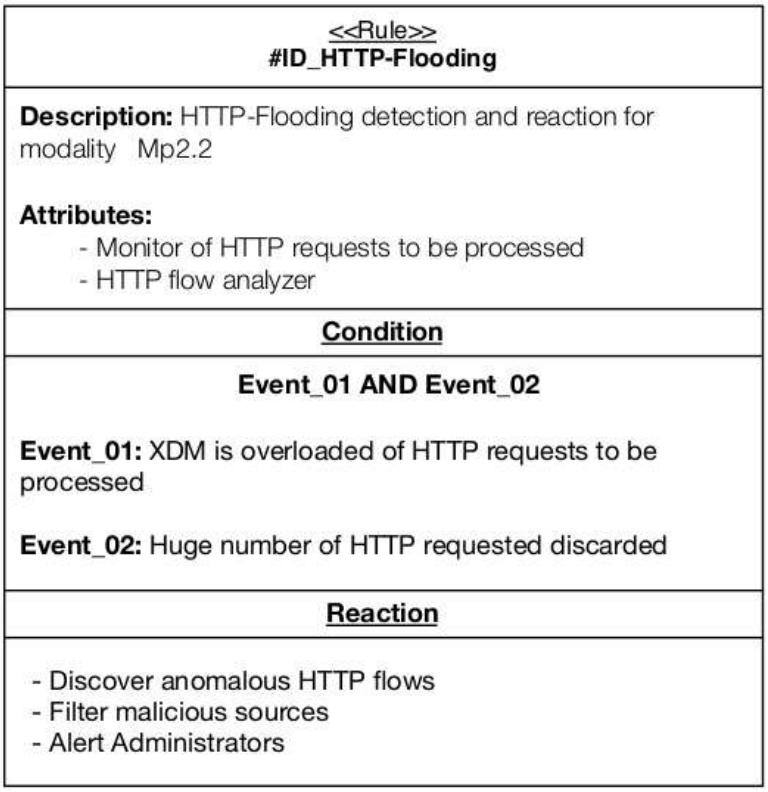


FIG. 5.5. *Detection rule description for modality* $M_{P1.2}$

HTTP requests. If the queue is full for a long time and many HTTP requests are discarded, a specific Detector is enabled. The Detector analyzes each incoming flow of requests. If a flow is identified as anomalous (it does not fit the model estimated during the training phase), the source of the flow is considered malicious and filtered by the HTTPgw. The detection method is based on the assumption that attackers and normal users have different properties in the request patterns. Therefore, a large volume of normal users' request sequences can be used to train a normal model, and then use the model to check incoming users. A Hidden Semi-Markov Model can be used to describe the process of an user visiting a website [27]. If the user's request sequence fits the model, the user will be considered as normal user. Otherwise, it will be considered as a potential attacker source and should be filtered.

6. Conclusions and Future Work. Security is one of the main limits in the adoption of Cloud infrastructures and applications. Users and providers do not have enough tools to evaluate and monitor the real enforcement of implemented security solutions. Indeed, security parameters are not included in Service Level Agreements as they cannot be negotiated and easily monitored. In this paper, we propose an approach to cope with this problem, by giving to cloud application developers the possibility to choose different security mechanisms that can be easily integrated in the application and have been pre-evaluated from the security point of view. We propose to model components' interactions from a security point of view in order to enable a developer to choose from different solutions which best fits the user requirements. The methodology we propose does not aim at proposing a new language to specify security attacks and requirements, indeed at the state of the art many solutions to this problem are based on this kind of approach. We aim at demonstrating the applicability of a simple methodology to implement security mechanisms in a transparent way with respect to the user applications, and to enable a Service Provider to easily offer security guarantees to customer. This is a first work in this direction, and we presented first results through a simple case study that introduces new components in an existing application to improve the overall security and meet user requirements. In future

work, we are going to implement different security packages and truly offer different secure interaction modalities. Our goal is to design a full and easy-to-use methodology to develop secure Cloud applications, such that the provided security is measurable and can be monitored by both end users and providers.

Acknowledgments. This work has been partially supported by the FP7-ICT-2013-10-610795 (SPECS).

REFERENCES

[1] M. Ficco, L. Tasquier, and B. Di Martino, "Interconnection of federated clouds," *Studies in Computational Intelligence*, vol. 511, pp. 243–248, 2014.
[2] C. Esposito, M. Ficco, F. Palmieri, and A. Castiglione, "Interconnecting federated clouds by using publish-subscribe service," *Cluster Computing*, vol. 6, no. 4, pp. 887–903, 2013.
[3] W. Theilmann, R. Yahyapour, and J. Butler, "Multi-level sla management for service-oriented infrastructures," vol. 5377, pp. 324–335, 2008. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-89897-9_28
[4] M. Comuzzi, C. Kotsokalis, C. Rathfelder, W. Theilmann, U. Winkler, and G. Zacco, "A framework for multi-level sla management," vol. 6275, pp. 187–196, 2010. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-16132-2_18
[5] CONTRAIL, "Contrail: Open computing infrastructres for elastic computing," http://contrail-project.eu/, 2010.
[6] mOSAIC Project, "mosaic: Open source api and platform for multiple clouds," http://www.mosaic-cloud.eu, 2010.
[7] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu, "Web services agreement specification (ws-agreement)," in *Global Grid Forum*. The Global Grid Forum (GGF), 2004.
[8] M. Ficco, "Security event correlation approach for cloud computing," *High Performance Computing and Networking*, vol. 7, no. 3, pp. 173–185, 2013.
[9] M. Ficco and L. Romano, "A generic intrusion detection and diagnoser system based on complex event processing," in *Proc. of the IEEE Int. Conf. on Data Compression, Communications and Processing (CCP'11)*. IEEE CPS, 2011, pp. 275–284.
[10] I. Ivanov, M. van Sinderen F. Leymann, B. S. SciTePress, and T. Publications, Eds., *Towards a cross platform Cloud API. Components for Cloud Federation*, 2011.
[11] M. Raihan and M. Zulkernine, "Asmlsec: An extension of abstract state machine language for attack scenario specification," *Proc. of the 2nd Int. Confe. on Availability, Reliability and Security (ARES'07)*, pp. 775–782, 2007.
[12] M. Zulkernine, M. Graves, and M. Khan, "Integrating software specification into intrusion detection," *Journal on Information Security*, vol. 6, pp. 345–357, 2007.
[13] M. Hussein and M. Zulkernine, "Intrusion detection aware component-based system: A specification-based framework," *Journal of System and Software*, vol. 80, no. 5, pp. 700–710, 2007.
[14] Hussein and M. Zulkernine, "Umlintr - a uml profile for specifying intrusions," *Proc. of the 13th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems*, pp. 279–286, 2006.
[15] S. Eckmann, G. Vigna, and R. Kemmerer, "Statl - an attack language for state-based intrusion detection," *Journal of Computer Security*, vol. 10, no. 1-2, pp. 71–104, 2002.
[16] F. den Braber, I. Hogganvik, M. Lund, K. Stølen, and F. Vraalsen, "Model-based security analysis in seven steps - a guided tour to the coras method," *BT Technology Journal*, vol. 25, no. 1, pp. 101–117, 2007.
[17] V. Casola, A. Mazzeo, N. Mazzocca, and V. Vittorini, "A security metric for public key infrastructures," *Journal of Computer Security*, vol. 15, no. 2, 2007.
[18] R. Preziosi, M. Rak, L. Troiano, and V. Casola, "A reference model for security level evaluation: Policy and fuzzy techniques," *Journal of Universal Computer Science*, January 2005.
[19] V. Casola, A. Mazzeo, N. Mazzocca, and M. Rak, "A sla evaluation methodology in service oriented architectures," *Advances in Information Security*, vol. 23, pp. 119–130, 2006.
[20] M. Ficco and M. Rak, "Intrusion tolerant approach for denial of service attacks to web services," *Proc. of the IEEE Int. Conf. on Data Compression, Communications and Processing (CCP'11)*, pp. 285–292, 2011.
[21] M. Ficco, M. Rak, and B. Di Martino, "An intrusion detection framework for supporting sla assessment in cloud computing," *Proc. of the 4th IEEE Int. Conf. on Computational Aspects of Social Networks*, pp. 244–249, 2012.
[22] M. Ficco, S. Venticinque, and B. Di Martino, "mosaic-based intrusion detection framework for cloud computing," *On the Move to Meaningful Internet Systems: OTM 2012*, vol. 7566, pp. 628–644, 2012.
[23] M. Jensen, N. Gruschka, R. Herkenhoner, and N. Luttenberger, "Soa and web services: new technologies, new standards - new attacks," *Proc. of the 5th European Conf. on web services*, pp. 35–44, 2007.
[24] N. Gruschka and N. Luttenberger, "Protecting web services from dos attacks by soap message validation," *IFIP Advances in Information and Communication Technology*, vol. 201, pp. 171–182, 2011.
[25] M. Ficco and M. Rak, "Intrusion tolerance of stealth dos attacks to web services," *IFIP Advances in Information and Communication Technology*, vol. 376 AICT, pp. 579–584, 2012.
[26] S. Meng, T. Wang, and L. Liu, "Monitoring continuous state violation in datacenters: Exploring the time dimension," in *IEEE 26th Int. Conf. on Data Engineering (ICDE)*. IEEE, 2010, pp. 968–979.
[27] W.-Z. Lu and S.-Z. Yu, "An http flooding detection method based on browser behavior," *Proc. of the Int. Conf. on Computational Intelligence and Security*, vol. 2, pp. 1151–1154, 2006.
[28] M. Ficco, L. Tasquier, and R. Aversa, "Intrusion detection in cloud computing," *Proc. of the 8th Int. Conf.on P2P, Parallel, Grid, Cloud and Internet Computing*, pp. 276–283, 2013.

Edited by: Teodor Florin Fortiş
Received: Mar 3, 2014
Accepted: Apr 16, 2014

DOI 10.12694/scpe.v15i1.966
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 15, Number 1, pp. 65–78. http://www.scpe.org

A DISTRIBUTED AGENT-BASED DECISION SUPPORT FOR CLOUD BROKERING

ALBA AMATO* AND SALVATORE VENTICINQUE†

Abstract. In goal oriented problems, decision-making is a crucial aspect aiming at enhancing the ability to make decisions by full autonomous, or supervised software agents. Agents usually resolve a huge number of evaluations and decisions without the user intervention. Just the remaining uncertainty should be left to the user's attention. In this paper we present a complete constrained multi-objectives optimization problem, modeled as an agent based decision support systems, that helps to choose the Cloud proposals that best satisfy the needs of the user, among the ones offered by known vendors. In particular we focus on a distributed solution that exploits the multi-agents programming model and the Cloud elasticity to comply with computational requirements of delivering such brokering at Cloud service level.

Key words: Multi-Agents systems, Cloud provisioning, Brokering, Multi-Criteria Decision

78

1. Introduction. The research of solutions for delegation to intelligent software agents in information systems has implied the need to develop systems able to act effectively: the systems must be able to act independently and follow the choice that best represents their interest during the interaction with a human being or another system.

To successfully interact, is essential for agent the ability to cooperate, coordinate, and negotiate with each other. In particular the research for even more intelligent systems implies that the complexity of tasks delegating to computers has also grown steadily leading to the need to develop systems able to act effectively and independently and to make decisions. In fact delegation and intelligence imply the need to build computer systems that can act effectively in a way that represents the best interests of the user while interacting with other humans or systems [23].

In this kind of goal oriented problem, decision-making is a crucial aspect aiming at enhancing the user's ability to make decisions. The decision has to be made among possible alternatives taking into account all the certainty and uncertainty considerations and using a prescribed set of decision criteria.

The reason for this intensive interest in decision-making [35] is that the metaphor of autonomous problem solving entities cooperating and coordinating in order to achieve their desired objectives is an intuitive and natural way of the problem solving. Moreover, the conceptual apparatus of this technology provides the powerful and useful set of computational structures and processes for designing and building the complex software applications.

The multi-agent techniques combined with decision-making tools can help decision makers to deal with the problems of the information overload. In addition, it is possible to make better and wider use of the existing knowledge by extracting value from large volumes of data. An additional challenge of the research is the possibility for an intelligent agent of being "rational" by having preferences about the goals to be achieved on the basis of interests expressed by the user. In particular the application of agent-based decision support helps users to make the right choice improving their satisfaction.

Enhancing software agents with decision making features and addressing them like rational entities is a topic of keen interest in designing agents and, which has been researched intensively [16] , [32].

The brokering problem, described in [1, 5] is a decision problem that consists of choosing the best proposals among the number of offers, which have been received from different providers, who answered to the same call [4]. The brokering has been addressed as a complete multi-objective constrained problem. The solution of this problem is one of the main objective of mOSAIC project [21].

To solve the brokering problem, we designed and developed Cloud Agency, a Multi-Agent System (MAS) that has the main task of dynamically selecting a set of Cloud resources, from different vendors, that best fits users' requirements. It also allows for vendor agnostic management and monitoring of Cloud infrastructures, where legacy or mOSAIC application are deployed as presented in [11]. It is compliant with the NIST definition

*Department of Industrial and Information Engineering, Second University of Naples, Aversa, Italy (alba.amato@unina2.it).
†Department of Industrial and Information Engineering, Second University of Naples, Aversa, Italy(salvatore.venticinque@unina2.it).

of cloud broker as defined in [20], that is an entity that manages the use, performance and delivery of cloud services, and negotiates relationships between Cloud Providers and Cloud Consumers. Cloud Agency actively performs cloud broker functions at the platform level.

In [3] we demonstrated the feasibility of the specific approach in the case of a single user, considering the number of existing providers in the current Cloud market, in the case that all the providers have at least one proposal. However the centralized approach is not effective in the case of brokering delivered as a service to a huge number of users. Performance bottleneck was observed so we investigated a new solution to support either a greater number of decisions and the dimension of the single decision problem.

To provide the brokering facility at Application as a Service level we have to overcome the performance limitations of the mOSAIC Cloud Agency solution, starting from the identification of new requirements. First of all it needs to identify the issues introduced by the new scenario, but also to take in consideration the available technologies for designing and implementing a new engineered solution.

About the issues, we need to take into account the number of service users who can access contemporary the service by multiple requests. We have to consider that in a service oriented context, not only human users, but also applications and robots will be able to invoke the service producing different kind of workloads. The new workload can vary in dimension and it can also dynamically change during the day, with regular or unforeseeable bursts on special periods. For this reason we have to share the workload over a distributed computing infrastructure, and we need to grant that both the infrastructure and the application will scale dynamically.

In this paper we present a scalable distributed multi-users version of the brokering service provided by Cloud Agency. Such new Broker As A Service solution exploits the capability of a distributed environment and addresses related issues. The idea is to divide the brokering problem into simple tasks, which are distributed to independent and collaborative agents, scaling dynamically in number, together the computing infrastructure, to support unforeseeable workloads produced by the interactions with large groups of users. In addition we evaluate and discuss performance results and effectiveness of a first prototype implementation.

2. Related Work.

2.1. Decision-making. According to [34] the Multi-Criteria Decision Making (MCDM) is divided into Multi-Objective Decision Making (or MODM) and Multi-Attribute Decision Making (or MADM). As described in [28] MODM studies decision problems in which the decision space is continuous. The goal is to find the best alternative among endless alternatives known implicitly.

On the other hand, MADM concentrates on problems with discrete decision spaces. In these problems the set of decision alternatives has been predetermined. Paper [9] claims that, although MADM methods may be widely different, many of them have certain aspects in common, such as the notions of alternatives, and attributes (or criteria, goals). In fact the general approach consists of the utilization of information about the problem (factual elements) together with the preferences express by the decision maker (value elements) to find the best compromise that will help to select the most consistent alternative, according to his preferences.

A decision making process typically has five steps that are summarized as follow [7]: Identify the problem or opportunity, Develop alternative, Evaluate alternative, Choose and implement the best alternative, Evaluate the decision.

In general, the outcome of a decision (action) depends on the context in which the decision is implemented, but a very general decision rule applied to MCDM problems consists in the identification of all efficient alternatives and the selection of the solution using the information of preference that the decision maker makes available. According to [6] the "optimum" solution corresponds to the feasible point for which the objective function achieves an optimum value.

The current role of multi-objective optimization in various sectors is becoming increasingly relevant. As with most problems, the objectives to be considered are many and often contradict each other. In particular, our approach is a MultiCriteria Decision Analysis/Aiding (MCDA) as addresses mainly discrete problems with not very large (combinatorial) sets of alternatives. For combinatorial problems find rules for decision making. Constrains are taken into account implicitly: into set of criteria and/or alternatives.

The multi-criteria decision-support is a valuable methodological tool in the processes of decision-making support. The multi-criteria decision-support is intended to provide to decision-makers some tools, which help

them in resolution of a problem of choice among alternatives, when numerous and often contradictory points of view must be taken simultaneously in consideration.

In general there is no decision (solution, action) that is the best one compared to all involved points of view. Besides there are situations where, as many factors have to be considered, the available alternatives are too much different or even not comparable. In addition, we are interested in those contexts where there are different stakeholders that give a different value to the several factors to be considered, that represent the preferences for criteria. Hence the problem is how to measure the importance of these criteria and how to combine these values.

In this scenario, one of the main implications of multi-criteria decision support has been the loss of optimality. In fact, given the presence of heterogeneous objectives often in conflict with each other, a cost benefit analysis is not feasible. Usually it is not possible to find solutions that simultaneously pursue all the objectives. The decision problem is solved looking for the most satisfactory solution, or "more consistent" with the logic of decision makers. The choice must therefore be performed among those solutions that realize a certain level of achievement of the various objectives, such that it is not possible to improve one of them without degrading some others.

According to [22], multiple criteria modeling methodology can address the following set of issues :

- Issue 1: Object of the decision, including the definition of the set of potential actions A and the determination of a problematic applied on A.
- Issue 2: Modeling of a consistent family of criteria assuming that these criteria are non-decreasing value functions, exhaustive and non-redundant.
- Issue 3: Development of a global preference model, to aggregate the marginal preferences on the criteria.
- Issue 4: Decision-aid or decision support, based on the results of Issue 3 and the Issue 1.

In Issue 1, Roy [22] distinguishes four referential problematics, each of which does not necessarily preclude the others. These problematics can be addressed separately, or in a complementary way in all phases of the decision-making process. They are listed below:

a: Choosing one action from A (*choice*).
b: Sorting the actions in well defined categories which are given in a preference order (*sorting*).
c: Ranking the actions from the best one to the worst one (*ranking*).
d: Describing the actions in terms of their performances on the criteria (*description*).

2.2. The Brokering Problem. There were some efforts aimed on solving the brokering problem. In [8] an architecture is presented for a federated Cloud computing environment named InterCloud to support the scaling of applications across multiple Cloud providers using a Cloud Broker for mediating between service consumers and Cloud coordinators for an allocation of resources that meets QoS needs of users.

Sim [24] proposes an extension of the alternate offers protocol that supports multiple complex negotiation activities in interrelated markets between user agents and broker agents, and between broker agents and provider agents.

In [19] an architectural design of a framework capable of powering the brokerage based Cloud services is presented. It is currently being developed in the scope of OPTIMIS, an EU FP7 project. The cited paper introduces the problem and the architectural design but it does not provide an implementation or algorithms to achieve the brokering.

Our brokering solution, overcomes the problems of the centralized broker so allowing a multi-user utilization. Besides it allows the concurrency, is highly scalable, open and available to enable easy extensions of existing components and addition of new components.

2.3. BDI agents. In BDI architecture [15],Beliefs, Desires, Intentions are the basic components of an agent that should be able to operate in a dynamic, uncertain world. Beliefs represent agent's knowledge of the world, Desires (or goals) represent what the agent wants and Intentions are a set of plans used to describe how an agent achieve his goals. BDI agents are adaptive in the sense that they can quickly reason and react to asynchronous events acting accordingly to them.

According to Wooldridge [32], the process of practical reasoning in a BDI agent is summarized in Figure 3.1 As this figure illustrates, there are seven main components to model a BDI agent:

- a set of current beliefs, representing information the agent has about its current environment;
- belief revision function, (brf), which takes a perceptual input and the agent's current beliefs, and on the basis of these, determines a new set of beliefs;
- an option generation function, (options), which determines the options available to the agent (its desires), on the basis of its current beliefs about its environment and its current intentions;
- a set of current options, representing possible courses of actions available to the agent;
- a filter function (decision), which represents the agent's deliberation process, and which determines the agent's intentions on the basis of its current beliefs, desires, and intentions;
- a set of current intentions, representing the agent's current focus (those states of affairs that it has committed to trying to bring about);
- an action selection function (execute), which determines an action to perform on the basis of current intentions.

BDI agents often have to make decisions, about which plan is used to achieve a goal, and in which order goals are to be achieved. Besides The BDI approach has proved valuable for the design of agents that operate in dynamic environments. It offers a higher level of abstraction by explicitly allowing beliefs to have a direct impact upon the agents behavior. This means the agents can respond flexibly to changing circumstances despite incomplete information about the state of the world and the agents in it [12].

2.4. Mult-agent decision support systems. Intelligent agents, when combined with Decision Support System, provide powerful support in solving difficult applied problems that are often real-time, involve large amounts of distributed data, and benefit from complex reasoning.

In fact it is evident [10] that a methodology for solving multi-criteria decision support can be applied to the agents modeling problems [13]: a consistent family of criteria (Issue 2) can sufficiently model the world with respect to the object of the decision (Issue 1). A global preference model (Issue 3) is able to guide agents' decisions. Finally, agents' actions will be realized based upon the decision support step (Issue 4).

Multiple criteria methodologies could contribute as a methodological background for agents modeling as well as a tool for their real-time implementation [29]. However, in this work we stress their potential for modeling support. In this respect, the potential contributions lie across two levels:

1. Agent design level: The multiple criteria modeling can respond to the representation issue discussed earlier. Existing methods can provide agents with critical decision making elements.
2. Multi agent level: Existing methods can profile decision makers (DMs), thus facilitate coordination – negotiation. In addition, DM's profiles can be exploited to assign roles or tasks to agents.

3. Problem Formulation. Problem formulation represents the first step of the decision process. The way of approaching the problems means defining a set of actions in order to act in an organized and methodical manner in the situations where a problem appears. In that way, the plan defines problem solving procedures according to the typology and the level of complexity of the problem itself. In particular in our approach the problem is modeled as shown in Figure 3.1. Some elements of the agents model will be introduced here and detailed in the next sections.

Using a simple definition, an agent can be seen as an entity that perceives the environment and acts rationally trying to achieve goals after that input perceptions and a knowledge base are given.

3.1. Environment. An agent is anything that can be viewed as perceiving environment through sensors and acting upon that environment through effectors. Intelligence of agents is the ability to adapt to the environment. In other words, as the environment changes an intelligent agent is able to adapt its behaviour to the new environment [25].

Environment represents information source to which react. This environment might be a real physical environment, a virtual one (so only existing in a computer), or a mix of both. Environment can be changed by agents through actions.

Arguably, the capability to adapt to the environment, means that just about any agent is an intelligent agent. Given an environment that can be in two possible states, an agent that is able to adapt its behaviour to both states is intelligent. An intelligent agent is then an agent that is able to adapt to a large number of states of its environment.

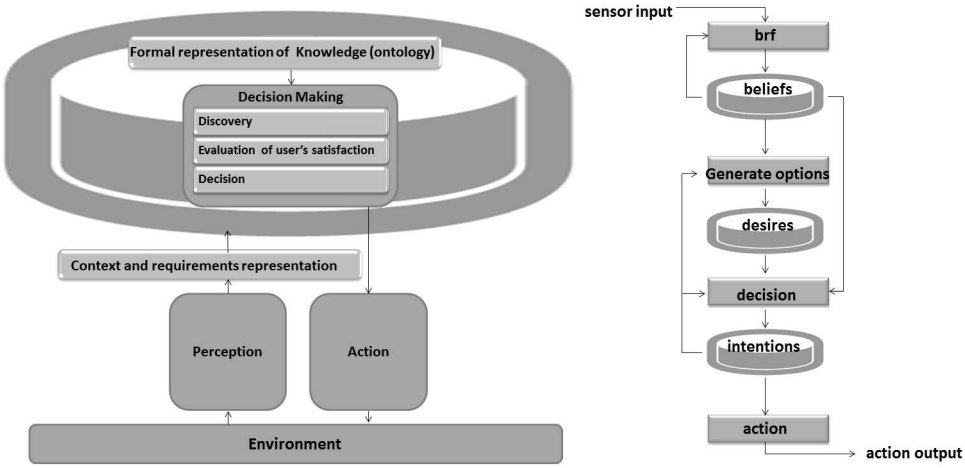


FIG. 3.1. *Decision-making support methodology*

In the brokering problem the environment is represented by the different proposals that agents perceive.

3.2. Knowledge representation. Knowledge is a set of representations of facts of the world and constitutes the base for reasoning. In facts intelligent agents need knowledge about the world for making good decisions.

Knowledge also includes the awareness and control of environment, but obviously an agent will not have complete control over its environment. In fact we always assume an agent has an imperfect view of its environment, and hence it cannot be certain if its actions always have the intended result. It will have at best partial control, in that it can influence it. The agent takes sensory input from the environment, and produces as output actions that will affect it.

Perceptions, state and actions of an agent constitute the agent's knowledge. If an agents is able to choose action sequences that maximize changes of the environment according to its own expected preferences, it can be defined a rational agent. Rationality is an important aspect of intelligence, although not the only one.

A rational agents has preferences about the state of its environment. What is the rational thing to do depends on the preferences of the agent, what at the agent has perceived so far, what the agent knows about its environment, what its sensors perceive and what the agent deduces from perceptions, what actions the agent is able to take.

In the brokering problem the knowledge is represented by the different proposals that agents perceive, together with the Call for Proposals and the set of constraints and objectives.

3.3. Decision Making. Making a rational agent more intelligent means making it better able to adapt to its environment. It implies more knowledge, better sensors, additional sensors, and enhancing its possible actions, but also better deduction skills.

The Decision Making process, described in detail in the next section, can be seen as an e inference mechanism, that uses perceptions and the knowledge base to deduce which actions to take.

4. The decision process. In our approach, the decision process is modeled as a problem of choosing among many contents or proposals to be retrieved according to the users' need. The process of discovery is performed in different ways depending on the problem and the solution is realized as a multi-agent system.

Multi-agent systems are fundamental enabling technology, especially in situations where mutual interdependencies, dynamic environments, uncertainty, and sophisticated control play a role. They can provide, as stated in [14], robust representational theories and very direct modeling technologies to help us understand large, multi-participant, multi-perspective aggregates.

The decision-making support methodology proposed here is shown in Figure 3.1. First of all it needs to define a knowledge model and its formal representation. It is necessary for the following evaluation and

comparison.

Besides it needs to provide a discovery service to collect the relevant information about the available choices. In particular in the following case studies a semantic discovery has been carried out and a negotiation protocol has been implemented to ask for a proposal to available vendors. Different objectives can be defined and both quantitative and qualitative characterization of available information must be provided.

In fact decision problems are governed by a number of stakeholders with their own objectives and priorities. Finally a strategy to evaluate the optimal decision is needed. The following subsections details each phase of the process shown in Figure 3.1.

4.1. Discovery service. The process of discovery consists of the retrieval of a set of resources, which are are relevant to the user's query, expressed by a set of requirements or a personal profile. The process of discovery is performed in different ways depending on the problem. The aim is the retrieval of all feasible alternatives.

4.2. Evaluation. The process of evaluation consists of assessing an overall evaluation measure to the alternatives in order to provide a ranking of alternative assignments. This overall evaluation is assessed taking into account user's constraints and preferences. The aim is to find the expected utility of each alternative, calculated using an utility function U(x) that depends on the problem.

4.3. Decision Strategy. The process of decision consists of filtering the alternatives with the highest expected utility, that are compliant with user's constraints. The aim is to choose the alternative or the set of alternatives that maximize the utility function.

Results of this phase could still be multiple choices, in fact aim of the decision support is to delegate all the evaluations and decisions, which can be resolved without the user intervention, to agents. Only the remaining uncertainty should be left to the user's attention.

5. The decision problem formulation.

5.1. Agents based modeling. In [33] an agent is defined as a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its objectives. We need to formally define objectives, believes and actions in order to formulate the problem in terms of multi-agent system.

The decision making model can be formally defined by the following items:

- A set of *Believes* $B = \{b_1, \cdots, b_{ne}, \}$, to be used for describing the knowledge in the application context, is described by a domain ontology. The ontology has to be defined for the specific use case.
- The knowledge of agent i at the time t, about the environment and about his own preferences, is described by a set of concepts and individuals belonging to the ontology. It is used to build the query for retrieving the available options to be evaluated in the decision process. For each user i we have a representation of his interest by a subset of preferred believes $P_i \subset B$ and a set of constraints $C(P)$.
- A discovery service for retrieving all available alternatives that are relevant to P_i.
- A set of goals: $G = \{g_1, \cdots, g_{nw}\}$, whose achievement improves the user's utility.
- Let us define $A = \{a_1, \cdots, a_{na}\}$ the set of assets which have been discovered. We need to define a multi-objectives metric for the user utility $U : A \rightarrow u_1(A), \ldots, u_{nw}(A)$, that is the measure of the achievement of each goal. U is a set of functions the defines the evaluation criteria based on which the alternative assets are evaluated;
- A decision maker or group of decision makers will choose the best collection of assets $A' \subset A$ which optimize the user's utility, but it is compliant with user's constraints. The optimal set of assets to be proposed will be $U(G(A')) > U(G(A' \cup a_j))\forall j : a_j \subset A - A'$. It means that any additional asset that is not useful to increase the user's utility must be excluded. Of course both an empty solution, or equivalent ones could be found.

In our multi-agents solution:

- perceptions provide information regarding the context of decision,
- the review process of knowledge updates the user profile and then the requirements for the decision,
- in the discovery phase the goal determines the alternative plans or actions to be evaluated for making the decision.

5.2. Believes and Preferences. In this case study, the set of *Believes* $B = \{b_1, \cdots, b_{ne},\}$ is described using the OCCI taxonomy defined in [17], an interface that offers a uniform access to IaaS resources. Here we define Call For Proposal (CFP) the document to be prepared by the customer to specify his requirements.

The CFP includes the list of resources to be acquired and the rules/policies to be used for defining resource brokering strategies, *i.e.*, prices, availability, etc.

As shown in Fig.5.1, the CFP is composed of two documents. The first one is the *SLA Template* described according to the XML SLA@SOI schema [27]. The *SLA Template* expresses the configuration of resources that are necessary for the user and consists of a set of virtual resources that may include compute, network and storage and can be complemented by the user with other information. In particular, as it is described in [2], the SLA template is composed of *Service Properties*, *Guarantee Terms* and *Terms of Service*.

The second document is the *Broker Policy*, containing a set of rules, to be enforced by the brokering algorithm, in order to choose among the different proposals offered by the Cloud market.

SLA brokering is part of the agent based provisioning service of Cloud Agency. The broker collects a number of proposals described in an vendor agnostic way and chooses the best one(s) according to the brokering rules.

Broker service provided by Cloud Agency is targeted to Cloud developers and deployers, who aim at building their own Cloud Infrastructure.

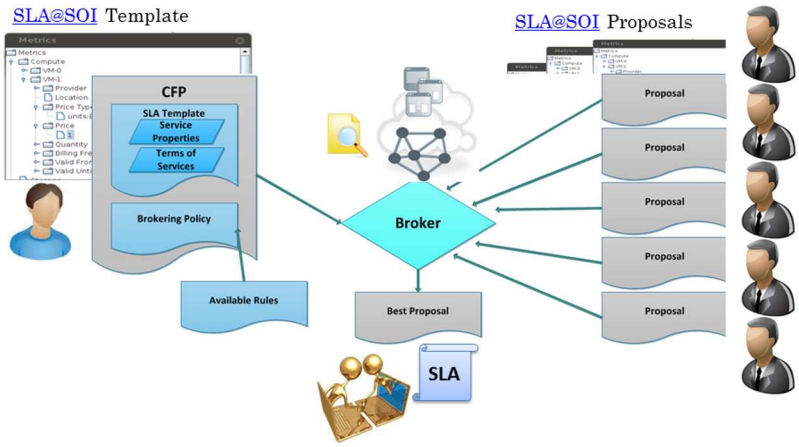


FIG. 5.1. *Broker*

Of course different proposals will come from Cloud Vendors. They will offer different kind of products at different terms. The broker should be able to choose the best one according to the policies specified by the customer such as best price per time unit, maximum amount of memory, service availability and so on.

The set of *Goals* $G = \{g_1, \cdots, g_{nw}\}$ includes the best price, the greatest number of cores, the best accredited provider or the minimum accepted availability. A number of constraints can be also defined to specify the minimal user's requirements.

6. Brokering Model. Our broker finds in a set of SLA proposals $\{P_1, \ldots, P_m\}$ the ones that optimize a multi objectives function and satisfy a number of constraints. Those proposals constitutes the set of *Assets* which have been discovered.

Each proposal is provided by a Cloud vendor complementing the terms of the SLA template $T = \{t_1, \ldots, t_n\}$, received from the customer, with the correspondent offered values $Pj = \{(t_1; v_{j,1}), \ldots, (t_n; v_{j,n})\}$.

In order to evaluate the best proposal the broker uses a set of rules $R = C \cup O$, which can be constraints rules $cr \in C$ or goals rules $or \in O$ rules.

Constraints rules are boolean expressions that represent soft or hard requirements:

$$cr : (t_i, c_i, m_i) \rightarrow [0, 1]$$

t_i is an SLA term, c_i is a boolean expression, and m_i is a float number whose value specifies that the constraints is hard (when 1) or soft (0).

Goals rules assign a score between 0 and 1 to the compliance of the SLA value of the term t_i with the correspondent user's requirements

$$or : (t_i, f_i, o_i) \rightarrow [0, 1]$$

For each goal rule the user has to select a mapping function f_i between the t_i values and the correspondent score, and has to specify if that rule is an explicit ($o_i = true$) or implicit goal ($o_i = false$).

The mapping function change the way used by the user to evaluate that goal. For example logarithmic, linear, and exponential function can be used to define the relevance of that goal according to the value $v_{j,i}$ offered by the provider and the one desired by the customer.

The broker policy will be a subset of rules $R' \subset R$ that is defined for those terms of the SLA template, which are relevant for the user's requirements.

To solve the brokering problem we have to perform the following computation for each received proposal by replacing t_i with the correspondent value $v_{j,i}$:

- $M_j = \prod_{i=1}^{n} \neg(\neg c_i \wedge m_i) \, \forall j = 1, \ldots, m$
  that is used to check if the SLA proposal can be considered as a valid candidate for the SLA , in fact at least a false hard constraint invalidates that offer.
- $Opt_j = \sum_{i=1}^{n} (\neg m_i * cr_i) \, \forall j = 1, \ldots, m$
  evaluates how many soft constraints are met. It can contribute to the evaluation of the proposal.
- $V_j = \sum_{i=1}^{n} (\neg o_i * f_i(v_{j,i})) \, \forall j = 1, \ldots, m$
  represents an overall evaluation for all those terms which have not to be negotiated independently.

All goals rules which have $o_i = true$ will be considered independent goals. In general the best proposals will be the ones which solve the following equations:

$$max_{j=0}^{m}(or_i) : o_i = true \; and \; M_j = 1 \, \forall i = 1, n$$

An additional criteria will be:

$$max_{j=0}^{m}(Opt_j) : M_j = 1$$

All the defined criteria can be grouped if the user set $o_i = false \; \forall i = 1, \ldots, n$, and $m_i = false \, \forall i = 1, \ldots, n$. In this case the result of brokering will be:

$$max_{j=0}^{m}(V_j) : M_j = 1$$

that means the best proposal are the ones with the best overall score.

The policy edited will be translated in a language supported by programs for symbolic computation like Octave, Matlab or Maple and is computed by replacing t_i parameter with actual values of each proposal.

6.1. Goals and Constraints. Goals and constraints are defined by a brokering policy embedded into the CFP. Constraints can be architectural constraints and service level constraints. Hard constraints refer to the fact that the cloud offer must satisfy the required condition, otherwise it is to be excluded. Soft constraints refer to desired requirements that can make a provider preferred with respect to another. The classification of constraints into hard or soft depends on user's need.

Explicit objectives can be the maximization or minimization of some parameters (e.g.: memory and price). Implicit objectives are automatically evaluated by the broker. An example is the amount of soft constraints that have been satisfied.

The rules can be defined selecting the SLA parameters and setting the required options using a friendly graphic interface.

Examples of constraints rules, shown in Table 6.1 are:

- exact matches,
- value in a set,
- greater/less then,
- value in a range.

TABLE 6.1
Rules Types

Rule's Name	**Value Type**	**Boolean Expression**
Exact Match	Numerical & Non Numerical	$t_i = s$
Value in a Set	Numerical & Non Numerical	$t_i \in S$
Greater then	Numerical	$t_i > s$
Less then	Numerical	$t_i < s$
Value in a Range	Numerical	$t_i \in R$

Of course not every constraint can be applied to any SLA parameters.

Some among the same parameters can be considered as input of objectives functions to be optimized [18]. There may be no one single goal for the optimization, and no single optimal solution. For example, given a set of constraints, a goal could be the minimization of the cost, maximizing the memory.

In this case, a multi-objective approach should be adopted and one of the solutions on the Pareto front (that is a set of all those solutions that are considered to be optimal in multi-criteria optimization) should be chosen. To choose one of the solutions on the Pareto front, a posteriori approaches is used that deliver to the user the set of Pareto-optimal solutions among which the user will choose the preferred one.

In order to simplify the usage of the brokering service we allow for grouping multiple objectives according to the kind of SLA parameter: *Service Properties*, *Terms of Services* or *Service Levels*. We also define the *Provider Reputation* as an additional brokering parameter, that is out of the SLA Template, but it is known to the broker.

To compute the overall score we map the domain of each SLA parameter to $[0, 1] \subset \mathcal{R}$ and we allow to assign a percentage relevance to each category.

7. Brokering As A Services.

7.1. Cloud Agency Brokering AAS requirements. Cloud Agency, presented in details in [30], is a Multi-Agent System that complements the common management functionalities which are currently provided by Private and Public Infrastructure as a Service (IAAS) with new advanced services, by implementing a Vendor Agnostic layer.

The Provisioning service of Cloud Agency implements the FIPA Contract-Net protocol described in [26]. It is a minor modification of the original contract net IP pattern in that it adds rejection and confirmation communicative acts.

In the contract net IP, one agent (the Initiator) takes the role of manager, which wishes to have some tasks performed by one or more other agents (the Participants) and further wishes to optimize a function that characterizes the task. This characteristic is commonly expressed as the price per time unit, in some domain specific way, but could also be soonest time to completion, fair distribution of tasks, etc.

For a given task, any number of Participants may respond with a proposal; the rest must refuse. Negotiations then continue with the Participants that returned valid proposals. For each received CFP Cloud Agency creates a broker that searches for vendors that can offer resources with the required QoS (Quality of services).

Cloud Agency implements a single-user service at platform level that can be used to provide autonomic capability to Cloud applications over IaaS infrastructures.

At Application as a Service level we have to overcome the performance limitations of the mOSAIC Cloud Agency solution as it has been developed [31], starting from the identification of new requirements.

First of all it needs to identify the issues introduced by the new scenario, but also to take in consideration the available technologies for designing and implementing a new engineered solution. About the issues we need to take into account the number of service users who can access contemporary the service by multiple requests.

We have to consider that in a service oriented context, not only human users, but also applications and robots will be able to invoke the service producing different kind of workloads. The new workload can vary in

dimension, but also it can dynamically change during the day, with regular or unforeseeable bursts on special periods. For this reason we have to share the workload over a distributed computing infrastructure, and we need to grant that both the infrastructure and the application will scale dynamically.

7.2. Architecture design and implementations. The Cloud technology is a promising solution to build a scalable computing infrastructure, but it needs to re-design the agents in order to let them exploit such an elastic computing model.

In fact while the Cloud allows to scale the computing resources according to the measured workload improving also the utilization, the application must be able to reconfigure itself autonomically to keep the QoS level above the desired threshold. Hence we are going to build a distributed Cloud broker over a Cloud infrastructure.

Not only performance issues should be addressed, but also reliability and availability when the service execute over such kind of distributed platform, due to lack of control both of the network and of the compute utility.

Hence, the main assumption we will take as strict requirement here will be the design of stateless and asynchronous scheduling of agents that will be able to run on every available computing resources.

This solution delegates the evaluation of a single proposal to any idle broker within a pool of agents, who are waiting for the evaluation of a proposal and update an eventual optimal solution. This design choice allows for the easy distribution of the workload using a task parallel programming model. On the other hand the front-end will be implemented by a service interface that accepts synchronous requests. It stores the new pending tasks to be handled by pools of asynchronous working agents, and returns the current status of service elaboration.

The service architecture is shown in Figure 7.1. In Figure 7.1, in the upper left corner, we can see many service instances that receive requests from end users and access in-memory shared information. The in-memory session manager will allow for the exploitation of elastic capability of the Cloud infrastructure. The warm copy of the session manager allows for improving reliability. As it is shown in Figure 7.1 agents will implement the back-end of the new agency, that is completely relieved of handling interactions with clients. Cloud storages (database ad queues) keep persistent information of the distributed applications and implement communication channel between synchronous front-end handlers and back-end workers. This design choice allows for the easy distribution of the workload using a task parallel programming model.

Stateless agents take new problems by a common bag of task, execute wherever there are available computing resources, and update the computing results if they complete successfully. Reliability is addressed by re-scheduling of unsolved problems which remain in the bag because of any failures or delay.

The vendors agent sign up to the CFP_QUEUE to receive the cfps received from users. Each of them submits its proposal to the PROPOSALS_QUEUE. Idle brokers are waiting for proposals. A single proposal is dispatched to one broker that executes its matching with the correspondent CFP for evaluation purpose. The matching results is updated into the SLA_QUEUE if it belongs to the Pareto front of optimal solutions. Brokering results are stored also into an in-memory session, together with information of each related user's request.

Users are provided with the web interface shown in Figure 7.2. It allows for composing the CFP and listing, for each session, the best SLAs according to different brokering objectives. Users log into a web page, the web page makes a request to a REST service and submits the call for proposals.

The Call for Proposal are included, along with information of the session and the user, in CFP queue. The characteristic of the CFP queue is that all consumers that join the queue, receive all the submitted CFPs.

Apache Tomcat has been used as web and application server to run the web service at front-end. It has been specifically configured for working with the Terracotta Framework for the transparent distribution and sharing of web sessions.

Jersey API have been used to implement the RESTFull service that provides methods for authentication, CFP submission and SLA retrieval. Meanwhile their requests are pending, users can wait for the result of brokering or may poll periodically to get the status of their request. When one of the brokers has found a feasible proposal for that request the current results are updated. The completion of brokering is notified when there are no more proposal candidate to optimize the user's query to be evaluated.

The RDBMS Mysql databases is used as persistent storage of CFPs and to SLAs. ActiveMQ has been used as queue services for communication and synchronization.

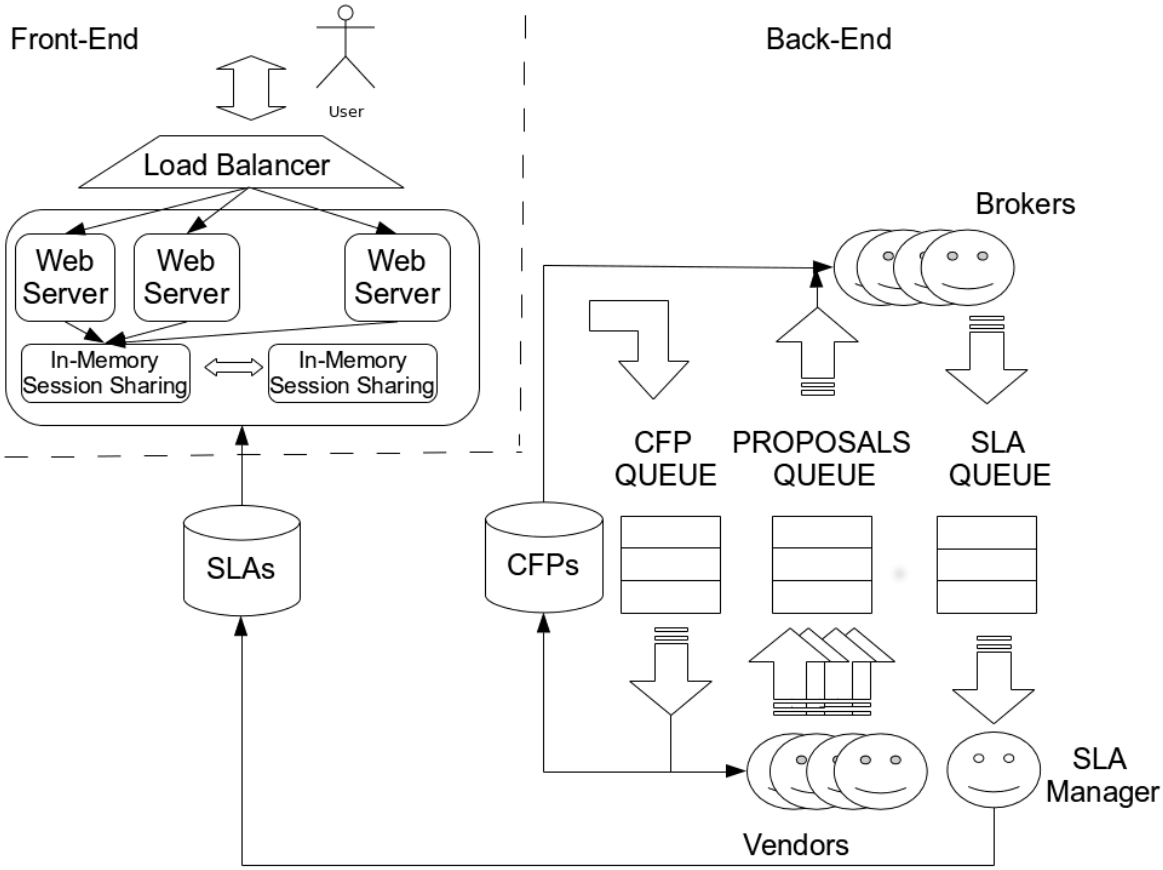


FIG. 7.1. *Distributed architecture*

In this solution, on the arrival of any Call for Proposal worker agents will cooperate to compare all the proposal candidate to satisfy the brokering policy. Each agent continuously will keep from a queues the next proposal to be evaluated, till when there are not more waiting requests. On their own users will be able to query the system about the status of their requests and the intermediate results.

7.3. Experimental results and discussion. In order to evaluate performances of the proposed approach we set up the following testbed. A Linux physical machine hosts the ActiveMQ 5.6 service, with a Topic, named CFP_QUEUE, that receives CFPs from concurrent clients, which run on a different physical machine in the same 1GB Ethernet local network. The server (H1) is 64bit Intel Core T M 2 Quad Processor Q9300 (6M Cache, 2.50 GHz, 1333 MHz FSB) with 4GB RAM. Oracle Java7 is the runtime environment.

Concurrent clients send 10 CFPs, each one, according to a Poisson process with different mean time of arrivals. Vendors run on the server and wait for incoming CFPs. All vendors get the same CFP and generate their proposal, which is sent to the $PROPOSALS\_QUEUE$.

In a first scenario all brokers run on the server itself. They receive a different proposal from the QUEUE and evaluate the compliance with the correspondent CFP. The result is sent to an SLA_QUEUE from which only the best ones are notified to the clients.

In different scenario, in order to improve this parameter we investigated the possibility to offload part of the workload by a Cloud Infrastructure.

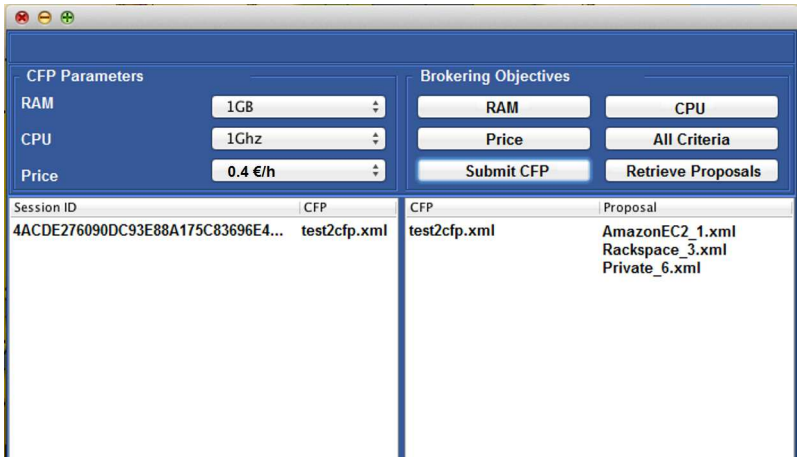


FIG. 7.2. *Web GUI*

We used an OpenStack installation in the same local network. This private cloud provided a Linux virtual machines (H2). The Linux OS sees just one processor with a 64bit virtual Intel Core 2 Duo P9xxx (Penryn Class Core 2) 2.5GHz with 2K L1 cache and 2GB RAM.

We evaluated the performance of such configurations changing the number of clients, the number of vendors and the number of brokers.

We observed a increasing throughput in terms of number proposals per second evaluated by brokers when the mean time of arrivals decreases and the number of brokers increases.

A drop of the throughput has been observed when the number of brokers running on a machine is greater than the number of cores, both on the server and on the working machines, and when the system is overloaded by a huge number of proposals to be evaluated, which depends wither on the number of vendors and on the mean time of CFPs arrivals.

In the first case the time of proposals evaluation increases due to the scheduling overhead. In the second case a noticeable effect is above all the average waiting time of a proposal into the queue, that affects directly the response time of the system and service level perceived by the client.

In Table 7.1 we show the mean enqueued time of a proposal in the case of 8 clients sending 10 CFPs with a mean time of arrivals of 500 ms. In all the case we have 8 vendors, that means 640 proposals to be evaluated.

Due to the unbalance between the computational power of H1 and H2 the queue service is able to dispatch a limited number of proposals to the working machine spawned in the Cloud and we have not a relevant benefit. We expect the with more homogeneous machines and with an greater number of working nodes we will get better improvement.

8. Conclusion. In this paper we presented a methodology that uses Multi Criteria Decision-Aid for designing and developing multi-agents solution of problems, whose complexity is characterized by different criteria to be evaluated and compared independently and according to the user's/agent's preferences.

We applied the proposed methodology to solve brokering problem of Cloud service with multiple objective and constraints set by the users' requirements. The proposed Brokering As A Service application has been built and executed on a distributed computing infrastructure that uses Cloud resources as working machines and asynchronous brokers for evaluating independent proposals from different vendors.

To perform an exhaustive search of the optimal solutions we assumed that the resulting SLA cannot be composed with offers from different vendors to dominate the complexity. We evaluated the performance changing the mean time of arrivals of CFPs from multiple clients and the number of vendors.

Future works will take in consideration larger problem by service compositions, using bigger computing infrastructure and heuristics for the computation of sub-optimal solution within the required time limits.

Furthermore we aim at advancing the current results by extending the proposed methodology to support multiple conflicting decisions in many-to-many negotiation protocols and by Game Theory techniques.

TABLE 7.1
Proposals mean enqueued time

Client	H1	H2	Mean Enqueued Time
8c	8v+1b		428
8c	8v+2b		180
8c	8v+4b		170
8c	8v+8b		171
8c	8v	1b	1430
8c	8v	2b	920
8c	8v	4b	11296
8c	8v+4b	1b	165.578
8c	8v+4b	2b	160
8c	8v+4b	4b	167

REFERENCES

[1] A. AMATO, G. CRETELLA, B. D. MARTINO, AND S. VENTICINQUE, *Semantic and agent technologies for cloud vendor agnostic resource brokering*, in AINA Workshops, 2013, pp. 1253–1258.

[2] A. AMATO, L. LICCARDO, M. RAK, AND S. VENTICINQUE, *Sla negotiation and brokering for sky computing*, in CLOSER, 2012, pp. 611–620.

[3] A. AMATO, B. D. MARTINO, AND V. SALVATORE, *Cloud brokering as a service*, in The 8th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, USA, December 28-30 2013, IEEE Computer Society, pp. 9–16.

[4] A. AMATO, L. TASQUIER, AND A. COPIE, *Vendor agents for iaas cloud interoperability*, in Intelligent Distributed Computing VI, G. Fortino, C. Badica, M. Malgeri, and R. Unland, eds., vol. 446 of Studies in Computational Intelligence, Springer Berlin Heidelberg, 2013, pp. 271–280.

[5] A. AMATO AND S. VENTICINQUE, *Multi-objective decision support for brokering of cloud sla*, in AINA Workshops, 2013, pp. 1241–1246.

[6] F. ANDRÈ, M. CARDENETE, AND C. ROMERO, *Basic aspects of the multiple criteria decision making paradigm*, in Designing Public Policies, vol. 642 of Lecture Notes in Economics and Mathematical Systems, Springer Berlin Heidelberg, 2010, pp. 33–53.

[7] H. ARMESH, *Decision making*, in Proceedings of the 12th International Business Research Conference, Melbourne, Australia, 2010, World Business Institute, pp. 11–21.

[8] R. BUYYA, R. RANJAN, AND R. N. CALHEIROS, *Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services*, in ICA3PP (1), 2010, pp. 13–31.

[9] S. J. CHEN, C. L. HWANG, AND F. P. HWANG, *Fuzzy multiple attribute decision making: Methods and applications in collaboration with dr. frank p. hwang*, (1992).

[10] P. DELIAS AND N. F. MATSATSINIS, *The multiple criteria paradigm as a background for agent methodologies*, in 8th Annual International Workshop 'Engineering Societies in the Agents World', 2007, pp. 227–237.

[11] B. DI MARTINO, D. PETCU, R. COSSU, P. GONCALVES, T. MÁHR, AND M. LOICHATE, *Building a mosaic of clouds*, in Proceedings of the 2010 conference on Parallel processing, Euro-Par 2010, Berlin, Heidelberg, 2011, Springer-Verlag, pp. 571–578.

[12] F. DIGNUM, D. MORLEY, E. SONENBERG, AND L. CAVEDON, *Towards socially sophisticated bdi agents*, in MultiAgent Systems, 2000. Proceedings. Fourth International Conference on, 2000, pp. 111–118.

[13] M. DOUMPOS AND E. GRIGOROUDIS, *Multicriteria Decision Aid and Artificial Intelligence: Links, Theory and Applications*, Wiley, 2013.

[14] L. GASSER, *Mas infrastructure: Definitions, needs and prospects*, in Revised Papers from the International Workshop on Infrastructure for Multi-Agent Systems: Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems, London, UK, UK, 2001, Springer-Verlag, pp. 1–11.

[15] M. P. GEORGEFF, B. PELL, M. E. POLLACK, M. TAMBE, AND M. WOOLDRIDGE, *The belief-desire-intention model of agency*, in Proceedings of the 5th International Workshop on Intelligent Agents V, Agent Theories, Architectures, and Languages, ATAL '98, London, UK, UK, 1999, Springer-Verlag, pp. 1–10.

[16] N. R. JENNINGS, K. SYCARA, AND M. WOOLDRIDGE, *A roadmap of agent research and development*, Autonomous Agents and Multi-Agent Systems, 1 (1998), pp. 7–38.

[17] T. METSCH, A. EDMONDS, AND R. NYREN, *Open cloud computing interface – core and models*, in Standards Track, no. GFD-R in The Open Grid Forum Document Series, Muncie (IN), 2011, The Open Grid Forum Document Series.

[18] F. MOSCATO, R. AVERSA, AND A. AMATO, *Describing cloud use case in metamorp(h)osy*, in CISIS, 2012, pp. 793–798.

[19] S. K. NAIR, S. PORWAL, T. DIMITRAKOS, A. J. FERRER, J. TORDSSON, T. SHARIF, C. SHERIDAN, M. RAJARAJAN, AND A. U.

Khan, *Towards secure cloud bursting, brokerage and aggregation*, in Proceedings of the 2010 Eighth IEEE European Conference on Web Services, ECOWS '10, Washington, DC, USA, 2010, IEEE Computer Society, pp. 189–196.
[20] NIST, *Nist cloud computing reference architecture - special publication 500-292*. Available at `http://www.nist.gov/`.
[21] D. Petcu, C. Craciun, M. Neagul, S. Panica, B. D. Martino, S. Venticinque, M. Rak, and R. Aversa, *Architecturing a sky computing platform*, in ServiceWave Workshops, 2010, pp. 1–13.
[22] B. Roy, *Multicriteria Methodology for Decision Aiding*, vol. 12 of Nonconvex Optimization and Its Applications, Springer, Formerly Kluwer Academic Publishers, Dordrecht, Boston, London, 1996. Translator: Mark R. McCord.
[23] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1995.
[24] K. M. Sim, *Towards complex negotiation for cloud economy*, in 5th International Conference on Advances in Grid and Pervasive Computing (GPC 2010), 2010, pp. 395–406.
[25] R. Slotboom, *Mobile agents as a distributed application architecture*, 2000.
[26] F. I. P. A. Technical report, *Fipa contract net interaction protocol*, 2002. Available at `http://www.fipa.org`.
[27] W. Theilmann, *Slasoi*, 2011. Available at `http://sla-at-soi.eu/`.
[28] E. Triantaphyllou, B. Shu, S. Nieto Sanchez, and T. Ray, *Multi-criteria decision making: An operations research approach*, In: Encyclopedia of Electrical and Electronics Engineering, vol. 15 (1999), pp. 175–186.
[29] P. Tučník, J. Kožaný, and V. Srovnal, *Multicriterial decision-making in multiagent systems*, in Computational Science – ICCS 2006, V. Alexandrov, G. Albada, P. Sloot, and J. Dongarra, eds., vol. 3993 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2006, pp. 711–718.
[30] S. Venticinque, *User-centric infrastructure as a service by cloud agency*, MULTIAGENT AND GRID SYSTEMS, 9 (2013), pp. 157–159.
[31] S. Venticinque, R. Aversa, B. Di Martino, M. Rak, and D. Petcu, *A cloud agency for SLA negotiation and management*, vol. 6586 LNCS of Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Springer, 2011, pp. 587–594.
[32] M. Wooldridge, *Intelligent agents: The key concepts*, in Proceedings of the 9th ECCAI-ACAI/EASSS 2001, AEMAS 2001, HoloMAS 2001 on Multi-Agent-Systems and Applications II-Selected Revised Papers, London, UK, UK, 2002, Springer-Verlag, pp. 3–43.
[33] M. Wooldridge, *An Introduction to MultiAgent Systems*, Wiley Publishing, USA, 2nd ed., 2009.
[34] H.-J. Zimmermann, *Fuzzy set theory and its applications, second edition*, (1991).
[35] R. Šperka and K. Slaninova, *The usability of agent-based simulation in decision support system of e-commerce architecture*, I.J. Information Engineering and Electronic Business, 4 (2012), pp. 10–17.

Edited by: Teodor Florin Fortiş
Received: Mar 3, 2014
Accepted: Apr 16, 2014

DOI 10.12694/scpe.v15i1.967
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 15, Number 1, pp. 79–87. http://www.scpe.org

IMAGE SCRAMBLING ON A "MESH-OF-TORI" ARCHITECTURE*

MARIA GANZHA† MARCIN PAPRZYCKI‡ AND STANISLAV G. SEDUKHIN§

Abstract.
Recently, a novel method for image scrambling (and unscrambling) has been proposed. This method is based on a linear transformation involving the Kroneker-delta function. However, while quite interesting, the way it was introduced, leaves some open issues concerning its actual usability for information hiding. Therefore, in this paper, we extend the original proposal and show how it can be used to securely pass image-like information between the users.

87

1. Introduction. Recently we observe a constantly growing interest in processing *multimedia content*. Here, the multimedia content is understood broadly and encompasses "still images" of all types, as well as video streams. Since a video stream can be viewed (with all understandable caution) as a sequence of still images (video frames), let us focus our attention on images, while keeping in mind that data processing may involve their sequences.

Observe that, there are two main sources of large images. First, sensor arrays of various types. For instance, one of the largest of them is the 2D pixel matrix detector installed in the Large Hadron Collider in CERN [14], which has approximately 10^9 sensor cells. Similar number of sensors would be required in a CT scanner array of size approximately $1m^2$, with about 50K pixels per $1cm^2$. Second, large data streams start to materialize in "consumer electronics." For instance, currently existing digital cameras capture images consisting of 22.3×10^6 pixels (Cannon EOS 5D Mark II [3], as well as its successor Cannon EOS 6) or even 36.3×10^6 pixels (Nikon D800 [4]). What is even more amazing, recently introduced Nokia phones (Nokia 808 PureView [5], and Nokia Lumia 1020 [6]) have cameras capturing 41×10^6 pixels.

What has to be noted, from the point of view of computing (here, image processing), is that most of the existing sensor systems do not consider the natural arrangement of data. Specifically, the input image, which is square or rectangular, is "serialized" to be processed. Specifically, input data from multiple sensors is read out serially, pixel-by-pixel, and send to the CPU for processing [1]. Next, after processing is completed, results are sent (serially) to a "display device," where they are reassembled into a rectangular format. This means that the transfer of pixels destroys the 2D integrity of data (in computer memory, an image or a frame, exists **not** in their natural layout). As a result, the need to transfer large amount of data, from "multiple data input streams" to the processor, may prohibit development of applications, which require (near) real-time response [10].

Observe that, for the (near) real-time image / video processing, as well as a 3D reconstruction, it could be advantageous to process them using a device that has multiple processing units arranged rectangularly, and that allows to load data directly from the sensors to these units (for immediate processing). This would be possible, for instance, when a focal-plane I/O, which maps the pixels of an image (or a video frame) directly into the stacked "array of processors," was to be used. Here, the computational elements could store the sensor generated information (e.g. a single pixel, or a block of pixels of a specific size) directly in their registers (or their local memory). Such an architecture would have three potential advantages. First, cost could be reduced, because there would be no need for the memory buses or a complicated layout of the communication network. Second, speed could be improved as the integrity of the input data would not destroyed by the serial communication. Third, speed could be considerably improved by skipping the step of serialization of the rectangular image and, later, its reassembling in order to be displayed. As a result, data processing could start as soon as the data is available (i.e. in the registers / memory). Note that proposals for similar hardware architectures have been outlined, among others, in [9, 16, 26]. However, such approaches have not been tried in real-life. Furthermore, previously proposed focal-plane array processors were envisioned with a mesh-based interconnect between the

*Work of Marcin Paprzycki was completed while visiting the University of Aizu.
†Systems Research Institute Polish Academy of Sciences, Warsaw, Poland, email: maria.ganzha@ibspan.waw.pl
‡Systems Research Institute Polish Academy of Sciences, Warsaw, Poland, email: marcin.paprzycki@ibspan.waw.pl
§University of Aizu, Aizu Wakamatsu, Japan, email: sedukhin@u-aizu.ac.jp
[1]Here, the term CPU is used broadly and encompasses standard processors, GPU's, Digital Signal Processing units, etc.

processing elements. This arrangement is good for the local data reuse (convolution-like simple algorithms), but is not the best to support global data reuse (matrix-multiplication-based complex algorithms).

2. Mesh-of-tori interconnection topology. This discussion leads us to the following question. Assuming that the focal plane I/O like approach is used to provide an effective data transfer to a rectangularly arranged group of processing units, what network topology should be used to connect them. Here, the experiences from development of parallel supercomputers could be useful. Historically, a number of topologies have been proposed, and the more interesting of them were: (1) hypercube – scaled up to 64000+ processor in the Connection Machine CM-1, (2) mesh – scaled up to 4000 processors in the Intel Paragon, (3) processor array – scaled up to 16000+ processor in the MassPar computer, (4) rings of rings – scaled up to 1000+ processors in the Kendall Square KSR-1 machines, and (5) torus – scaled up to 2048 units in the Cray T3D.

However, all of these topologies suffered from the fact that at least some of the elements were reachable with a different latency than the others. This means, that algorithms implemented on such machines would have to be asynchronous, which works well, for instance, for ising-model algorithms similar to these discussed in [8], but is not acceptable for most computational problems, that require synchronous data access. Therefore, an extra latency had to be introduced, for the processing units to wait for the information to be propagated across the system. Obviously, this effect became more visible with the increase of the number of processing units. At the same time, miniaturization counteracted this process only to some extent. Interestingly, supercomputers with some of the most efficient network connectivity, the IBM Blue Gene machines, combine the toroidal network with an extra networking provided for operations involving global communication (primarily, reduction and broadcast). To overcome this problem, recently, a new (*mesh-of-tori*; *MoTor*) parallel computer architecture (and topology) has been proposed. Let us now summarize this proposal. What follows is based on [20] and this source should be consulted for further information.

The fundamental (*indivisible*) unit of the *MoTor* system is a μ-Cell. The μ-Cell consists four computational units connected into a 2×2 doubly-folded torus (see, Figure 2.1). Logically, an individual μ-Cell is surrounded by so-called membranes that allow it to be combined into larger elements through the process of cell-fusion. Obviously, collections of μ-Cells can be split into smaller structures through cell division. For instance, in Figure 2.1, we see a total of 9 μ-Cells logically fused into a single macro-μ-Cell consisting of 4 μ-Cells (combined into a 4×4 doubly folded torus), and 5 individual (separate) μ-Cells. Furthermore, in Figure 2.2 we observe all nine μ-Cells combined into a single system (a 6×6 doubly folded torus; with a total of 36 processing units). Observe that, when the 2×2 (or 3×3) μ-Cells are logically fused (or divided), the newly formed structure *remains* a doubly folded torus. In this way, it can be postulated that the single μ-Cell represents a "holographic image" of the whole system.

Let us note that the proposed *MoTor* topology has similar restriction as the array processors from the early 1990's. Specifically, the *MoTor* system must be square. While this was considered an important negative factor in the past, this is no longer the case. When the first array processors were built and used, arithmetical operations and computer memory were "expensive." Therefore, it was necessary to avoid performing "unnecessary" operations (and maximally reduce the memory usage). Today (December 2013), when GFlops costs about 16 cents (see, [25]) and its price is systematically dropping; and when laptops come with 8 Gbytes of RAM (while some cell phones include as much as 64 Gbytes of flash memory on a card), it is the data movement / access / copying that is "expensive" (see, also [13]). Therefore, when images (matrices) are rectangular (rather than square), it is reasonable to assume that one could just pad them up, and treat them as square. Obviously, since the μ-Cell is a single indivisible element of the *MoTor* system, if the matrix is of size $N \times N$ then N has to be even (or padded to be such).

Note that in earlier publications (e.g. [20, 19, 18, 21] the computational units, as well as the whole concept of the *MoTor* system, were purely "theoretical entities." However, in [11] we have considered such system more closely, from the perspective of its potential realization. This line of reasoning resulted in the following proposal of the *computational unit* to be realized in the *MoTor* system. (1) It accepts input from the sensor(s) and transfers it directly to the operational registers / local memory. (2) Is capable of generalized fused multiply-add (*gfma*) operations, originating from various algebraic semirings (see, also [22, 1] for more details). The latter requirement means that, (3) the *gfma* unit considered here should store (in its registers) all constants needed to efficiently perform *fma* operations originating from various semirings. Finally, analysis of cell connectivity

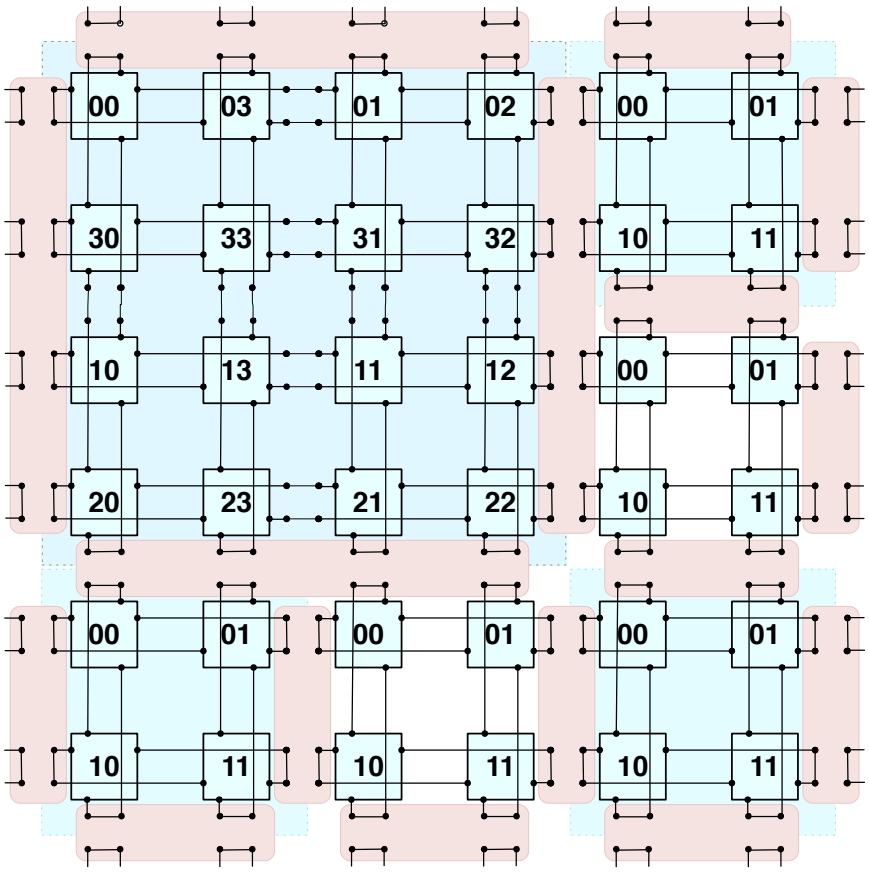


FIG. 2.1. *9 μ-Cells fused into a single 2×2 "system," and 5 separate μ-Cells*

in Figure 2.1 shows that (4) the *gfma* unit should include four interconnects that allow assembly of the *MoTor* system. Let us name such computational unit the *extended generalized fma: egfma*. Furthermore, let us keep in mind that the *MoTor* architecture is build from *indivisible* μ-Cells, each consisting of four, interconnected into a doubly folded torus *egfma* units.

Let us now observe that there are two sources of inspiration for the *MoTor* system: (i) processing data from, broadly understood, sensor arrays (e.g. images), and (ii) matrix computations. Furthermore, we have stated that the *egfma* should contain data registers to store (a) the needed scalar elements originating from various semirings, (b) data that the *fma* is to operate on and, as stipulated in [11], (c) elements constituting special matrices needed for matrix operations / transformations. Here, we have to take into account that each *egfma* should have a "local memory" to allow it to process "blocks of data." This idea is based on the following insights. First, if we define a pixel as "the smallest single component of a digital image" (see, [7]), then the data related to a single RGBX-pixel is very likely to be not larger than a single 32 bit number. Second, in early 2013 the (Tianhe-2) has 23,040,000 `fma` units [2] (see, [2]). This means that, if there was a one-to-one correspondence between the number of *egfma* units and the number of "pixels streams," then the system could process stream of data from 23 Megapixel input devices (or could process a matrix / image of size $N \simeq 4800$). This is clearly not enough. Finally, let us note that to keep an *fma* / *gefma* unit operating at 100% it is necessary to form a pipeline that is (minimally) 4-6 elements deep. Therefore, from here on, we will assume (and in this follow [19])

[2]Here, we count only the Intel Xeon Phi nodes. There are 48000 such nodes, each with 60, 8-way, cores, giving us a total of 23,040,000 double precision `fma` units

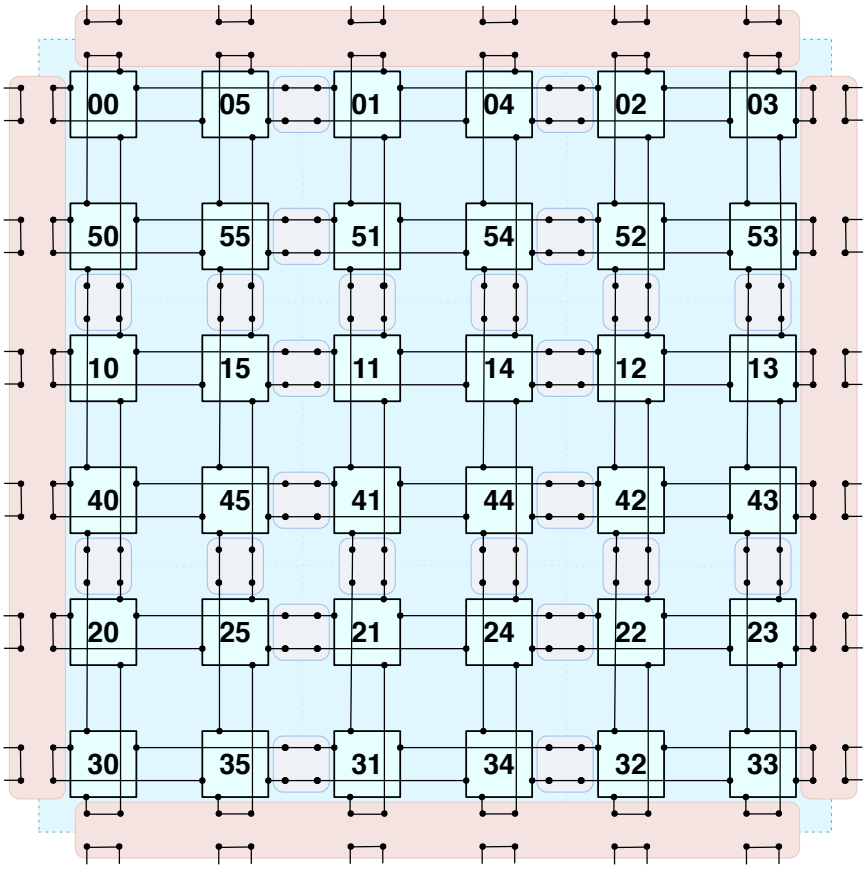


FIG. 2.2. *9 μ-Cells fused into a single 6×6 EG FMA system*

that each computational unit in the *MoTor* system has local memory and thus be capable of processing blocks of data. For instance, a natural way of augmenting the *egfma* to achieve this goal would be to use 3D stacked memory [15].

Let us now consider the development of the *MoTor*-based system. In the initial works, e.g. in [20], links between cells have been conceptualized as purely abstract links (μ-Cells were surrounded by logical membranes that could be fused or divided as needed, to match the size of the problem). Obviously, in an actual system, the abstract links and membranes could be realized logically, while the whole system would have to be hard-wired to form an actual *MoTor* system (of a specific size). Therefore to build a large system with M^2 μ-Cells (recall the assumption that the *MoTor* system will have the form of a square array), it can be expected that such system will consist of silicon etched groups of μ-Cells residing on separate chips (μ-processors), combined into the *MoTor* system of a given size (similarly to multicore / multi-FMA processors combined into supercomputers).

As what concerns cell fusion and division, it will be possible to assemble sub-system(s) of a needed size, by logically splitting and/or fusing an appropriate number of cells within the *MoTor* system. However, it should be stressed that while the *theoretical* communication latency across the *MoTor* system is uniform, this is not likely going to be the case when the system will be assembled from μ-processors constituting physical macro-μ-Cells. In this case it may be possible that the communication within the μ-processor (*physical* macro-μ-Cell) will be relatively faster than between the μ-processors. Therefore, the most natural μ-Cell split should involve complete μ-processors. However, let us stress that the design of the mesh-of-tori topology *does not* distinguish between the connections that are "within the μ-processor" and "between μ-processors." Therefore, the communication model used in the algorithms described in [20, 19, 11], and applied in subsequent sections, is *independent* of the

hardware configuration.

3. Fundamental operation. To proceed, let us note that, in what follows, we use the generalized matrix multiply-and-update operation (MMU) in the form elaborated in [23, 12, 11]:

$$\mathtt{C} \leftarrow \mathtt{MMU}[\otimes, \oplus](\mathtt{A}, \mathtt{B}, \mathtt{C}) : \mathtt{C} \leftarrow \mathtt{C} \oplus \mathtt{A}^{\mathtt{N/T}} \otimes \mathtt{B}^{\mathtt{N/T}}.$$

Here, A, B and C are square matrices of (even) size N; while the $\otimes, \oplus$ operations originate from a matrix semiring; and $\mathtt{N/T}$ specify if a given matrix is to be treated as being in a standard ($\mathtt{N}$) or in a transposed ($\mathtt{T}$) form, respectively.

3.1. Reordering for the mesh-of-tori processing. Before discussing image processing, which is the main contribution of this paper, we have to consider the data input (e.g. from the sensor array, a medical scanner, or a cell phone photo camera) into the *MoTor* system. As shown in [20] (and followed in [11]), any input that is in the canonical (standard image / square matrix) arrangement, is not organized in a way that is needed for the matrix processing in a (doubly folded) torus. However, as shown in [19], the needed format can be obtained by an appropriate linear transform, through two matrix-matrix multiplications. Specifically, matrix product in the form $M \leftarrow R \times A \times R^T$, where A is the original / input ($N \times N$) matrix that is to be transformed, M is the matrix in the format necessary for further processing on the *MoTor* system, and R is the format rearranging matrix (for the details of the structure of the R matrix, consult [19]). Taking into account the implementation of the generalized MMU operation (see, equation 3), the needed transformation has the form:

$$M = R \times A \times R^T.$$

Note that, according to [20, 19], on the *MoTor* system: (a) operation $Z = R \times A$ is performed in place and requires N time steps, (b) operation $M = Z \times R^T$ is performed in place, and also requires N time steps and is implemented as a parallel matrix multiplication with a *different* data movement pattern than the standard multiplication. In other words, the matrix arrangement within the system remains unchanged for the transposed matrix operations, and the well-known problems related to row vs. column matrix storage (see, for instance, [17]) do not materialize (for more details, see [24]).

Observe that, when instantiating the *MoTor* system, it is assumed that an appropriate matrix R will be preloaded into the macro-μ-Cell, upon its creation (see, [11] for more details). Specifically, in addition to the operand registers dedicated to special elements $(\bar{0}, \bar{1})$ originating from selected semirings, appropriate elements of a transformation matrix, needed to perform operations summarized in [11] will be preloaded in separate operand registers. These operations include the transformation from the canonical to the "*MoTor* format" and back. Therefore, matrix R (of an appropriate size) will be preloaded into the *MoToR* system.

4. Image scrambling and unscrambling. Taking into account the background material presented thus far, let us focus on the main contribution of this paper. An interesting application of the matrix multiplication has been proposed in [18]. There, a linear transform, based on the Kronecker-delta function, was introduced. This transformation was then used to create a scrambling matrix C that could be used to scramble and unscramble images (via matrix-matrix multiplication). Since matrix denoted as "C" is very often used in different contexts (e.g. see, the above equation 3), for the purpose of this paper, let us re-name it as $SCRAM$. The complete description of the linear transform, as well as the specific structure of the $SCRAM$ matrix can be found in [18]. Let us recall that, while the discussion below considers "single element per *gefma*" model, the real assumption is that a "data block per *gefma*" format is used (for more details, see [18, 19]). Therefore, all claims should be seen as actually concerning a *blocked data format*. Finally, we assume that image / matrix A is of size $N \times N$ (with N even).

Similarly to the processing needed to transform the matrix / image from the canonical to the *MoTor* format (and back), image (represented as a matrix A) scrambling consists of a triple-matrix product (forward transform): $S \leftarrow SCRAM \times A \times SCRAM^T$; where S is the resulting scrambled image/matrix. Unscrambling

of the same image (S) is a result of the following triple-matrix product (inverse transform): $A \leftarrow SCRAM^T \times S \times SCRAM$.

In [18], two ways to apply this approach to scramble images have been proposed. First, scrambling was to be applied $1 \leq J \leq N$ times to the whole image (matrix). The second approach was a "progressive" one. Here, one had to pick a key (parameter) $1 \leq K_P \leq N$. Next, scrambling was applied to the left top corner of the image, of size $K_P \times K_P$, $2K_P \times 2K_P$, and proceeded until the whole matrix was scrambled (in the case when N was not divisible by K_P then the complete matrix / image was scrambled in the last step).

In both cases, the scrambling operation could be realized by a $Scramble(A, nb)$ function. This function was to perform the tripe-matrix product involving the left top corner of size $nb \times nb$, where $1 \leq nb \leq N$. In the first approach, scrambling was achieved by calling the $Scramble(A, N)$ function J times. Observe that, on the *MoTor* system, each scrambling step would be performed in place, and would take $2N$ time steps. Therefore, the total cost of scrambling would be $2N \leq 2NJ \leq 2N^2$ time steps. Here, to unscramble the image without knowledge of the key K, would require knowledge of the matrix $SCRAM$, and applying the unscrambling operation J times (using function $ScrambleInv(A, N)$) until the original image was to be revealed (to the human observer).

In the second case, the $Scramble(A, nb)$ function would be called N/K_P times (and possibly one more time if N was not divisible by K_P). On the *MoTor* system, each scrambling step would cost $2K_P$ time steps (plus the cost of cell fusion, and of re-instantiating the $SCRAM$ matrix (for the next image / matrix size); however, these costs would be negligible in comparison with the cost of matrix multiplication). Therefore, the total cost would be between $2N$ (for $K_P = N$) and $4N - 2$ for (for $K_P = 1$) time steps. Here, recovering the original image would require knowledge of the matrix $SCRAM$, and searching space of all possible values of K_P, by applying the unscrambling function $ScrambleInv(A, nb)$.

Let us stress that the material presented in [18], while conceptually originating from the same roots (dense matrix multiplication), was not directly related to the *MoTor* architecture. Furthermore, it did not consider practicalities of use of the proposed transformation for secure image (multimedia) transfer / communication. Earlier, we have already made some comments about possible implementation of the scrambling (and unscrambling) procedures on the *MoTor* system, conceptualized as above (and approached as in [11]). Let us continue this line of reasoning.

First, let us make an obvious observation. The goal of image scrambling is to be able to *hide* the information (make the image unrecognizable to unauthorized persons), pass it to the authorized recipient, and reveal it by unscrambling. In this context, let us consider in some detail the implementation of the proposed image scrambling on a *MoTor* system. Here, the first approach involves initialization of the $SCRAM$ matrix (similarly to the R matrix mentioned above, and to other transformation matrices summarized in [11]). Note that, only a single $SCRAM$ matrix is going to be needed, as the scrambling procedure consists of application of the $Scramble(A, N)$ function J times to the whole image (matrix). As stated above, this approach is not safe at all. As soon as the potential attacker knows the $SCRAM$ matrix, (s)he can repeatedly apply the $ScrambleInv(A, N)$ function and after J steps the original image will be revealed. Furthermore, the computational cost of unscrambling will be the same as that of scrambling (assuming that the attacker also has a *MoTor* system at her/his disposal).

The situation is different in the second approach. Here, even the knowledge of the $SCRAM$ matrix, and the possession of the *MoTor* system, do not help the potential attacker sufficiently. Without the knowledge of the K_P parameter, the search space to uncover the information is much larger (though, obviously, the unscrambling is not impossible). However, this approach would be somewhat difficult to efficiently implement on a *MoTor* system (as described above, and in [11]). Observe that the recursive approach would require cell fusion. For instance, if $K_P = 64$ then the first scrambling would be performed on a block of size 64×64 (on a macro-μ-Cell consisting of 32×32 μ-Cells). Next, cell fusion would have to be applied (to create a macro-μ-Cell consisting of 64×64 μ-Cells, and matrix $SCRAM$ would have to be re-initialized, to apply scrambling to a block of size 128×128). In other words, let us assume that image of size 1024×1024 is to be scrambled, with the key $K_P = 16$. This means that the macro-μ-Cells of size 8, 16, 32, 64, 128, 256 and 512 would have to be used, and the $SCRAM$ matrices of sizes $N = 16, 32, 64, 128, 256, 512, 1024$ would have to be instantiated after each cell fusion. Note that the $SCRAM$ matrices cannot be preloaded, among others because the user can pick any value for K_P (even values that are not the most effective from the perspective of the *MoTor* system; including odd

values of K_P). Here, it should be stressed that, while use of odd values of K_P is possible, this does not match the concept of the *MoTor* system, where the μ-Cell is the fundamental computational unit. Furthermore, this also means that there is no natural way of utilizing the cell fusion and splitting that could allow dynamic creation of the *MoToR* systems that match the size of the scrambled block. Finally, note that padding, proposed above for the full-matrix operations, is not an option in an "internal step" of image scrambling, as the scrambling operation has to be performed on a block of data within an image (matrix).

Summarizing, while very interesting conceptually, ideas for image scrambling presented in [18] are not best suited for the *MoTor* architecture. Therefore, we propose a slightly different approach to image scrambling (and unscrambling), seen as a mechanism for information hiding (and safe transmission) on a *MoTor* architecture. Let us assume that two users would like to communicate multimedia content using the scrambling / unscrambling method discussed above. Observe that, for any matrix A, as long as the $SCRAM$ matrix is known, the $ScrambleInv(A, nb)$ reverses the effect of application of the $Scramble(A, nb)$ function. This being the case, it is easy to see that all that is needed is that the sender applies the $Scramble(A, nb)$ function a specific number of times (e.g. L times) for the selected values of nb. Next, it communicates to the receiver a tuple $(nb_1, nb_2, \ldots, nb_L)$ that defines what were the values of nb_i used in each of the scrambling steps, and what was their order. Here, we assume that the $SCRAM$ matrix is known to both the sender and the receiver. Next, the receiver applies the $ScrambleInv(A, nb)$ function in an appropriate order (based on the known sequence of nb_i values) to recover the original image. A small "restriction" on this approach is such that at least one of the scrambling operations should involve the whole matrix A (as application of only blocked scrambling, without scrambling the whole image, leaves an unscrambled image strip; see [18, 19]). However, this approach makes it very difficult to unscramble the original image even if the attacker knows the form of the matrix $SCRAM$. The problem is in the fact that the sequence $(nb_1, nb_2, \ldots, nb_L)$ can be completely "random," while being known only to the sender and the receiver.

The interesting part of this approach is in the fact that the number different blocks used in scrambling does not have to be large (as they can be repeated in the scrambling sequence) and that they can be defined in such a way to match the structure of the *MoTor* system. Here, let us recall that the size of the macro-μ-Cell can be dynamically adjusted through cell division and fusion. However, the image (matrix) A is stored across the whole *MoTor* system and this fact is not going to change. Let us now assume that the following scrambling sequence is to be applied $nb = 1024, 256, 512, 1024$ to a matrix (image) of size 1024×1024. Here, it is possible (assuming that this is known in advance) to instantiate three $SCRAM$ matrices in the *MoTor* system. Each element of these $SCRAM$ matrices would be stored in a separate register. Now, the first scrambling would involve triple-matrix multiplication performed on the whole system. Next, the cell division would be applied to create a subsystem of size 256×256 (macro-μ-Cell consisting of 128×128 μ-Cells). Here, let us note that this subsystem would have preloaded $SCRAM$ matrix of the correct size. In the following step, cell fusion would be used to create a subsystem of size 512×512 (macro-μ-Cell consisting of 256×256 μ-Cells); where an appropriate $SCRAM$ matrix would be already preloaded. After the scrambling operation, cell fusion would be applied, again, to restore the original system, where the initial $SCRAM$ matrix would still be available. Another triple-matrix multiplication would end the process. The unscrambling will proceed in exactly the opposite order, dynamically splitting and fusing the meta-μ-Cells and using the preloaded $SCRAM$ matrices.

4.1. Object oriented implementation. In our earlier work (see, [23, 12, 11]), one of our goals was to develop a library of functions that will simplify writing codes for scientific computing applications (through application of matrix operations, represented in the style similar to that found in MATLAB / MATHEMATICA). In [11] we have introduced such a library, for a collection of operations involved in matrix / image manipulations, and supporting global reduction and broadcast operations. Obviously, there are multiple ways of implementing the proposed routines, and it is likely that such implementations are going to be vendor / hardware specific. Nevertheless, currently, object oriented (OO) programming is one of the more popular ways of writing codes in scientific computing and image processing. This being the case, we have conceptualized the top-level object oriented representation of the routines proposed in [11]. Since different OO languages have slightly different syntax (and semantics), we have used a generic notation, focusing on distinguishing information that needs to be made available in the interface and that to be placed in the main class. Here, we extend our proposal to include also scrambling and unscrambling operations. We start from the interface (see, also [11] for the

remaining matrices and operations that have been omitted in the snippets below).

```
/* T - type of matrix element */

interface Matrix_interface {
        public Matrix 0(int n) {/*generalized zero matrix*/}
        public Matrix I(int n) {/*generalized identity matrix*/}
        public Matrix operator + /*generalized  A+B*/
        public Matrix operator * /*generalized A*B*/
        public Matrix Canonical_to_Motor(Matrix A);
        /*reordering for the mesh-of-tori processing*/
        public Matrix Motor_to_Canonical (Matrix A);/*inverse of
        the reordering for the mesh-of-tori processing*/
        public Matrix Scramble (Matrix A,int nb);
        /*Matrix (Image) Scrambling*/
        public Matrix ScrambleInv (Matrix A,int nb);
        /*Matrix (Image) Unscrambling*/
}
```

This interface is to be used with the following class *Matrix* that summarizes the proposals outlined above.

```
class Matrix inherit scalar_Semiring
                implement Matrix_interface {
   T: type of element;/*double, single,...*/
   private Matrix SCRAM (int n) /*scrambling matrix
   private Matrix R (int n);/*matrix for MoTor
         transformation*/
   private Matrix ONES (int n) /* matrix of ones*/
   private Matrix PERMUT(int i,j,n)/*identity matrix
        with interchanged columns i and j*/
   //anti-diagonal matrix of ones
   private Matrix SWAP (int n)
        // Methods
   public Matrix 0(int n) {/*0 matrix*/}
   public Matrix I(int n) {/*identity matrix*/}
   public Matrix transpose(Matrix A){
                /*MMU-based transposition of A*/}
   public Matrix operator + (Matrix A,B)
                {return MMU(A,I(n),B,a,b)}
   public Matrix operator * (A,B:Matrix)
                {return MMU(A,B,matrix_0,a,b)}
        ...
        /*image scrambling*/
        public Matrix Scramble (Matrix A,int nb){
                return SCRAM * A * SCRAM^T};
        public Matrix ScrambleInv (Matrix A,int nb){
                return SCRAM^T * A * SCRAM};
        ...
   private MMU(A,B,C:Matrix(n)){
     return "vendor/implementer specific
         realization of MMU = C + A*B where
     + / * are from class scalar_Semiring"}
...
}
```

Obviously, this class and the interface would allow writing codes in the suggested manner. Here, the matrix operations (image scrambling and unscrambling) would be performed by calling simple functions, and hiding all implementation details (including the existence of the matrix $SCRAM$) from the user.

5. Concluding remarks. The aim of this paper was to extend and modify a recently proposed novel method of image scrambling. The proposed improvements were made from the perspectives of (1) practical use of image scrambling for secure transmission of multimedia content, and (2) realization of the proposed method on the *MoTor* architecture. We have started from the rationale behind the need for efficient processing of large scale multimedia content and used this to outline the fundamental properties of the *MoTor* architecture. Next, we have introduced the image scrambling and unscrambling method proposed in [18]. This method was then analyzed from the perspective of its practical applicability, which resulted in the proposed modifications. Finally,

we have illustrated how the proposed approach fits into the object oriented realization of matrix operations proposed in [11]. In the future we plan to implement the proposed approach on the virtual *MoTor* system and study its efficiency.

Acknowledgment. Work of Marcin Paprzycki was completed while visiting the University of Aizu.

REFERENCES

[1] *Kalray multi-core processors*. http://www.kalray.eu/.
[2] *Top500 list*. http://www.top500.org.
[3] *Canon EOS 5D*. http://www.usa.canon.com/cusa/consumer/products/cameras/slr_cameras/eos_5d_mark_iii, 2013.
[4] *Nikon D800*. http://www.nikonusa.com/en/Nikon-Products/Product/Digital-SLR-Cameras/25480/D800.html, 2013.
[5] *Nokia 808 Pureview*. http://reviews.cnet.com/smartphones/nokia-808-pureview-unlocked/4505-6452_7-35151907.html, 2013.
[6] *Nokia Lumia 1020*. http://www.nokia.com/global/products/phone/lumia1020/, 2013.
[7] *Wikipedia pixel*. http://en.wikipedia.org/wiki/Pixel, March 2013.
[8] P. Altevogt and A. Linke, *Parallelization of the two-dimensional ising model on a cluster of ibm risc system/6000 workstations*, Parallel Computing, 19 (1993), pp. 1041–1052.
[9] S. Chai and D. Wills, *Systolic opportunities for multidimensional data streams*, Parallel and Distributed Systems, IEEE Transactions on, 13 (2002), pp. 388–398.
[10] D. Fey and D. Schmidt, *Marching-pixels: a new organic computing paradigm for smart sensor processor arrays*, in CF '05: Proceedings of the 2nd conference on Computing frontiers, New York, NY, USA, 2005, ACM, pp. 1–9. doi:http://doi.acm.org/10.1145/1062261.1062264.
[11] M. Ganzha, M. Paprzycki, and S. Sedukhin, *Library for matrix multiplication-based data manipulation on a "mesh-of-tori" architecture*, in Proceedings of the 2013 Federated Conference on Computer Science and Information Systems, M. Ganzha, L. Maciaszek, and M. Paprzycki, eds., IEEE, 2013, pp. pages 455–462.
[12] M. Ganzha, S. Sedukhin, and M. Paprzycki, *Object oriented model of generalized matrix multiplication*, in FedCSIS, IEEE, 2011, pp. 439–442.
[13] J. L. Gustafson, *Algorithm leadership*, HPCwire, Tabor Communications (April 06, 2007).
[14] E. H. M. Heijne, *Gigasensors for an attoscope: Catching quanta in CMOS*, IEEE Solid State Circuits Newsletter, 13 (2008), pp. 28–34.
[15] P. Jacob, A. Zia, O. Erdogan, P. M. Belemjian, J.-W. Kim, M. Chu, R. P. Kraft, J. F. McDonald, and K. Bernstein, *3D memory stacking; mitigating memory wall effects in high-clock-rate and multicore CMOS 3-D processor memory stacks*, Proceedings of the IEEE, 97 (2009).
[16] S. Kyo, S. Okazaki, and T. Arai, *An integrated memory array processor architecture for embedded image recognition systems*, in Computer Architecture, 2005. ISCA '05. Proceedings. 32nd International Symposium on, June 2005, pp. 134–145.
[17] M. Paprzycki, *Parallel Gaussian elimination algorithms on a Cray Y-MP*, Informatica, 19 (1995), pp. 235–240.
[18] A. Ravankar and S. Sedukhin, *Image scrambling based on a new linear transform*, in Multimedia Technology (ICMT), 2011 International Conference on, 2011, pp. 3105–3108.
[19] A. A. Ravankar, *A new "mesh-of-tori" interconnection network and matrix based algorithms*, master's thesis, University of Aizu, September 2011.
[20] A. A. Ravankar and S. G. Sedukhin, *Mesh-of-tori: A novel interconnection network for frontal plane cellular processors*, 2013 International Conference on Computing, Networking and Communications (ICNC), (2010), pp. 281–284.
[21] ———, *An O(n) time-complexity matrix transpose on torus array processor*, in ICNC, 2011, pp. 242–247.
[22] S. G. Sedukhin and T. Miyazaki, *Rapid*closure: Algebraic extensions of a scalar multiply-add operation*, in CATA, 2010, pp. 19–24.
[23] S. G. Sedukhin and M. Paprzycki, *Generalizing matrix multiplication for efficient computations on modern computers*, in Parallel Processing and Applied Mathematics, R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Waśniewski, eds., vol. 7203 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2012, pp. 225–234.
[24] S. G. Sedukhin, A. S. Zekri, and T. Myiazaki, *Orbital algorithms and unified array processor for computing 2D separable transforms*, in Parallel Processing Workshops, International Conference on, Los Alamitos, CA, USA, 2010, IEEE Computer Society, pp. 127–134.
[25] Wikipedia, *Flops*. http://en.wikipedia.org/wiki/FLOPS.
[26] Á. Zarándy, *Focal-Plane Sensor-Processor Chips*, Springer, 2011.

Edited by: Dana Petcu
Received: Dec 1, 2013
Accepted: Jan 15, 2014

DOI 10.12694/scpe.v15i1.968
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 15, Number 1, pp. 89–99. http://www.scpe.org

TTL-CHORD: A CHORD-BASED APPROACH FOR SEMANTIC WEB SERVICES DISCOVERY

MOHAMED GHARZOULI *, YOUCEF MESSELEM †, AND MOHAMED ELHADI BOUNAS ‡

Abstract. Recently, P2P-based discovery methods for semantic Web services require special attention from collaboration and interoperability in distributed computing environment. In this paper, we present an approach for semantic Web services discovery through a structured P2P network based on Chord. This protocol is chosen because it is scalable and still achieves very good resilience and proximity performance. However, because the discovery of Web services is made semantically, the search time will be significant. For this reason, we integrate a proposed algorithm into original implementation of Chord, where we mark the sent requests by a TTL (Time To Live). After this proposition, we present some experimental results wherever we analyze and discuss the influence of the TTL on the discovery operation and on the network stability.

Key words: Web services discovery, P2P, Chord, distributed algorithm

99

1. Introduction. In recent years, Peer-to-Peer (P2P) computing has emerged as a popular model for distributed computing. Quickly, P2P infrastructures are increasing important attentions from both industry field and academic field. It enables large-scale aggregation of resources (e.g. files, applications, services, storage) geographically distributed and belonging to different domains. P2P networks have received great attention due to their inherent flexibility. In comparison to traditional client/server paradigm, P2P systems offer more advantages in term of scalability and robustness [1, 15].

Since their apparition, P2P networks have been used for files exchange between different participants. Until now, this application occupies the most important traffic rate of generated requests in P2P networks. However, the increasing popularity of P2P systems for file sharing indicates general interest in resources sharing [14]. In the last few years, research on P2P systems has been quite intensive, and has produced significant results in scalability, robustness, location, distributed storage, and system measurements. P2P systems are being increasingly used in many recent application domains [20]. Among these emergent research fields, Web services based on P2P computing require special attention from collaboration and interoperability in a distributed computing environment. In this field, a particular interest has been expressed on Web services discovery, in which important research works focus on new mechanisms for large-scale discovery of Web services [17, 24].

In the same context, to involve the automatic Web services discovery, which requires a more intelligible description, the Web semantic technologies have been used to improve the automatic discovery and composition of Web services. A semantic description is more comprehensible, which facilitates the dynamic discovery of Web services. In this field, many semantic languages are proposed like OWL-S [7], WSMO [21], METEOR-S [29] and other specifications.

Consequently, the P2P- based approaches for semantic Web services discovery are the result of the convergence of three IT disciplines: Web services, semantic Web and P2P computing. In this context, several research works improve the decentralized discovering strategies of semantic Web services in the P2P networks. An important number of them focus on new methods, algorithms and frameworks to efficiently and effectively select and compose semantic Web services over large-scale P2P overlay [4, 18, 27, 34].

In comparison with centralized discovery methods of Web services (such as those based on UDDI (Universal Discovery, Description and Integration) [35]), P2P-based discovery mechanisms provide a scalable alternative to centralized repositories by distributing the Web services among all peers. The P2P-based approaches offer a decentralized and self-organizing context, where Web services interact with each other dynamically [10].

However, although the results obtained by many researchers, P2P-based discovery methods pose many challenges in terms of protocols, infrastructures, fault tolerance, scalability and performance. An important issue arising in P2P applications is how to accurately and efficiently discover the required services in P2P

*Department IFA, Faculty of New Technologies of Information and Communication, University Constantine 2, Algeria. (gharzouli@gmail.com).
†Department IFA, Faculty of New Technologies of Information and Communication, University Constantine 2, Algeria.
‡Department IFA, Faculty of New Technologies of Information and Communication, University Constantine 2, Algeria.

networks. A number of criteria like: the type of the P2P overlay, the network size and the number of pertinent peers involve the quality of responses.

In contrast with our previous works ([9, 10, 11]), where we have employed the unstructured P2P networks for semantic Web services discovery and composition, in this paper, we present a new distributed algorithm based on an architecture that combines hybrid and structured topologies. We concentrate our efforts only on the discovery operation, where we try to implement a distributed algorithm for semantic Web services discovery by using an ameliorated version of Chord protocol [32], in which we have marked the research requests by a TTL (Time To Live).

The rest of this paper is structured as follows. In Section 2, we briefly introduce a background about discovery methods for Web services. Section 3 describes our solution for semantic Web services discovery. In this section, we present an architecture to implement this solution and we propose a distributed algorithm for Web services discovery in a hybrid/Chord P2P network. In Section 4, we discuss some experimental results. Section 5 reviews related works and Section 6 concludes the paper and discusses further research.

2. Discovery methods for semantic Web services. Before presenting the different categories of semantic Web services discovery methods, it is important to clarify first what precisely means "Web services discovery". Although various proposals for discovering Web services are available, the understanding of discovery is different, and has often been confused with terms such as selection and matching [36]. Among little number of definitions of services discovery in the literature, the most official definition is that presented in [38]:

"The act of locating a machine-processable description of a Web service-related resource that may have been previously unknown and that meets certain functional criteria. It involves matching a set of functional and other criteria with a set of resource descriptions. The goal is to find an appropriate Web service-related resource".

Originally, the main goal of Web service technology is to define Web services in a machine-understandable way, in order to be automatically discoverable by other actors (agents or services). Consequently, to find the appropriate Web services is considered as an important key to service composition, invocation and execution. Basically, Web services are discovered by measuring the similarity between service requirements given by a service requester and service advertisements from service providers [36]. However, independently to the category of technique used to measure the similarity measurements, we can classify the Web services discovery methods into two main types: centralized and decentralized.

2.1. Centralized Methods. Centralized discovery methods (such as UDDI) present the first generation of works done on Web services architectures, where Web services are described by service interface functions and they publish their capabilities and functionalities with a registry [37]. Consequently, the centralized discovery methods are not adapted to the dynamic interactions, they restrict the scalability of flexible and dynamic environment [12, 18], induce performance bottleneck and may result in single points of failure [26]. Moreover, the centralized control of published services suffers from many problems such as high operational and maintenance cost.

In addition, even if the Web services are described semantically, one of the major problems with existing structure is that UDDI does not capture the relationships between entities in its directory and therefore is not capable of making use of the semantic information to infer relationships during search [3]. Secondly, UDDI supports search based on the high-level information specified about businesses and services only. It does not get to the specifics of the capabilities of services during matching [28].

2.2. Distributed Discovery Methods. To solve the problems of the centralized methods, other architectures are proposed to facilitate the discovery and composition of Web services. Recently, many solutions are proposed to proceed to the distributed discovery of Web services. The majority of research works illustrate P2P-based discovery methods. Many of them are related to automated discovery and composition of semantic Web services in the P2P networks. These research works are categorized according to the different types of P2P networks: unstructured (Gnutella V0.4 [2]), hybrid (Gnutella 0.6 [25]) or structured (like Chord [32]).

In this research field, several research works improve the discovering methods of Web services in the structured P2P networks. These systems provide an effective routing of a point to another of the system, and are based on a concept of local knowledge: a node does not have total knowledge, but it can approaches to the required data.

Structured P2P systems are proposed to solve a number of problems appeared in first generation of these systems (hybrid and pure P2P systems). They are based on a Distributed Hashing Table (DHT). An example of a structured P2P system (the most used in the recent research works) is the protocol Chord.

3. A Chord-based Architecture for Semantic Web Services Discovery. This paper is interested to decentralized P2P networks. As we have already mentioned, in these systems, there are two methods to send queries: DHT and flooding-based algorithms. In comparison with unstructured P2P topologies, structured P2P systems have several advantages like: scalability, availability, and management of the fault tolerance.

On the other hand, unstructured P2P networks (such as Gnutella V0.4) require no centralized directories and no precise control over network topology or data placement. Though, the flooding-based query algorithm used in these systems does not scale; a query generates a large amount of traffic hence large systems become quickly overwhelmed by the query-induced load [6].

Our proposed architecture is based on Chord, which is chosen because it is simple and still achieves very good resilience and proximity performance. In the following subsection, we will briefly present the main characteristics of this protocol.

3.1. Chord Overview. Chord implements a DHT using an m-bit identifier ring, $[0, 2^m - 1]$, for routing and object location. In an $N - nodes$ system, a query can be routed via $O(\log N)$ hops (see Fig.3.1). The destination point of a query is determined by hashing its key. At each hop during routing, the query is forwarded to a peer whose identifier most immediately precedes the destination point in its finger table (contains the list of peers that a peer uses to send messages to) [4].

Chord is a dynamic system where the peers can enter or leave the system at anytime at will. Then, each node maintains information only about $O(\log N)$ other nodes, and resolves all lookups via $O(\log N)$ messages to other nodes. Chord maintains its routing information as nodes join and leave the system; with high probability each such event results in no more than $O(\log_2 N)$ messages [32].

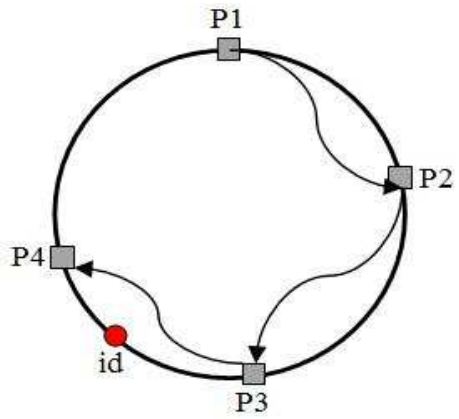


FIG. 3.1. *Example of Chord system*

Chord protocol has several advantages and important characteristics like: Scalability, availability, and management of the fault tolerance. However, in the context of semantic Web services discovery, the requester can search for a capacity, a goal or a property of a resource. In this case, one of the disadvantages of this protocol is the convergence of the request, especially, when the number of nodes is certainly large. For this reason, we have integrated the time aspect in our discovery algorithm, in which we have marked the requests by a TTL (Time To Live).

3.2. An overview of the proposed architecture . In this paper, we describe a P2P based architecture that combines the hybrid and the structured typology of P2P network, in which we employ Chord. This architecture supports the centralized and decentralized discovery methods of Web service. It uses a central common ontology, which used by the different providers to annotate semantically their Web services descriptions.

The P2P network is used to organize the service providers into a hybrid/Chord overlay and allows them to advertise and lookup services in a centralized/decentralized dynamic manner. Figure 3.2 shows the main components of this architecture.

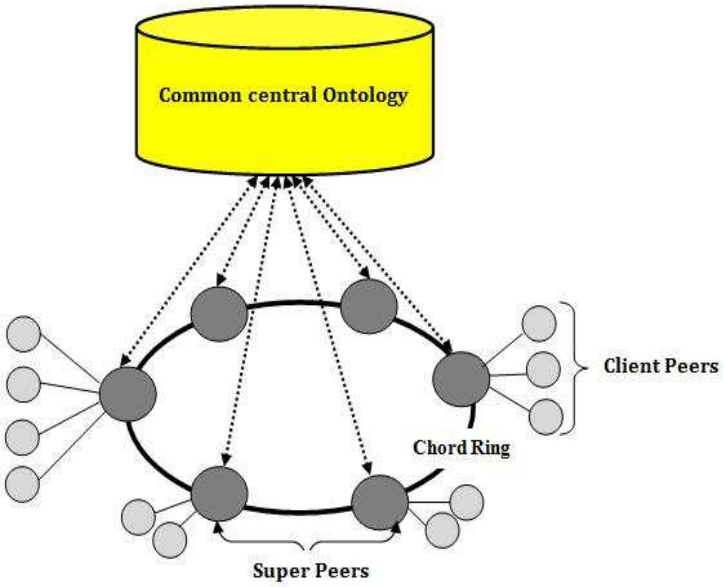


FIG. 3.2. *The main Architecture*

The main P2P network is presented by the whole of super peers organized as a Chord ring. The main system is decomposed on clusters presented by the Client/Server relationship between each super peer and their connected client peers. A cluster is a mini network, in which every super peer provides to the client peers an ensemble of services in term of discovery and publication of Web services. In the following subsections, we detailed the three main components of this architecture: Central common ontology, clusters and the distributed discovery over the Chord overlay.

3.3. Central Common Ontology. In a large distributed system, it is evident that the provided resources are strongly heterogeneous. In the same context, if the resources are Web services, this problem is vital. Even though the technical interoperability of Web services was assured essentially by the three main standards XML, SOAP [30] and WSDL [31], the semantic interoperability is already a real problem that influences the discovery of Web services, especially, in the large distributed networks.

Consequently, to guarantee the semantic interoperability between the different provider peers of the network, it's important to provide a mechanism that assures the dynamic discovery of Web services those distributed among the different participant peers.

In this framework, we propose to use a common central ontology that provides a large variety of concepts of great selection of Web services domains. In the first case, each provider peer can reuse one or more several descriptions that provided by this central base. An example of such ontology is that created by [33] that

provide about 240 OWL-S services descriptions based on a main ontology named *"concepts.owl"*. In this case, if the provider want to supply a Web service that corresponding to one of the provided OWL-S descriptions, he can reuse directly this one. Thus, in our proposed architecture each provider peer can use the common ontology to extend semantically the descriptions of their Web services.

Furthermore, in our work, we suppose that every Web service is defined by the tipple (input, output, goal). The following example presents a Web service that accepts in its input book information, in the output the price of this book and the goal specifies the used Money (EUR in this example):

LISTING 1

example of a semantic search request

```
Input : Title||Author||Edition||year-of-edition Output : Price
Goal:*input:=#book:Title||Author||Edition||year-of-edition
*output:=#book:Price//Money:EUR
```

3.4. Client Peers/Super Peers. A client peer joins the system by connecting to one and only one super peer. Then, it leaves the network by disconnecting from it's super node. Each super peer processes the different requests forwarded by their client peers. However, this description does not mean that client peers are only customers. They can provide some Web services; which are indexed and published within the Chord ring by super peers.

Furthermore, to the high availability and computing capacity, super peers are characterized by the number and the quality of the provided services and resources. Thus, after the first peer joins the system as a super peer, it designs one of its client peers with good resources as a new super peer. Furthermore, the super peer can observes the participation of each client peer, if anyone was a good participant; it will be supported by the super peer to be a new one.

In addition, the clustering organization of the peers (client peers/Super peers) offers many benefits:

- Whenever client peers just send queries and reply some service requests, super peers (peers provide a lots of services) benefit directly from the advantages of the Chord overlay network by doing all the message routing and indexing which minimizes the discovery time of the requested Web services when the requestor is a super peer.
- This gain for super peers, in term of resources and the search time by exploiting the distributed system (Chord ring), encourages the other peers (client peers) to provide more services and participate actively in the system. Thus, the number of the selfish peers in the system will be reduced, whenever, this one of the most important problems of the P2P systems (selfish peers are just consumers, they don't provide any resource for the other peers).
- For the super peers, the system is more flexible and scalable because of the simplicity of Chord in term of routing messages and locating resources. Whereas, the client peers can join (leave) the system by simply connecting to (disconnecting from) their super peers.

3.5. Distributed discovery over the Chord ring. To ensure the cooperation between the different super peers, we implement a distributed discovery algorithm over the Chord ring. As a first step of this work, we favorite to propose the Algorithm 13, that researches only for basic Web services. However, we have optimized the original implementation of Chord protocol through the insertion of a TTL (Time To Live) into sent requests. Because the discovery of Web services is made semantically, the search time, in each super peer, will be significant. For this reason, the marking of the request by a TTL becomes the discovery operation more realistic and reflects the exigencies of users in term of research time.

In this context, it's important to mention that the original implementation of Chord didn't design for "semantic discovery". It is more developed for "exact search" where the requester knows previously the precise identifier of the service. So, the "semantic discovery" is the most motivation to use the TTL in our solution. In this approach, the requester doesn't know the exact identifier of the requested service. The semantic discovery is based on keywords extracting from the common ontology. Consequently, the integration of the TTL becomes a necessity because the search time is important in such approach. Thus, in contrast with the original implementation of Chord, the requestor searches "semantically" for the adequate service until he finds it or the discovery operation stops when the TTL is finished.

The following Algorithm 13 is implemented in each super peer. It is executed when a super peer receives a request from a client peer or another super peer in the Chord ring.

From the Algorithm 13, we can deduce that the execution stops if we obtained the researched Web service or if the TTL is finished. For this reason, it's evident that the insertion of the TTL influences directly the obtained result. To study this effect, in the following section, we present some experimental results after the simulation of the Algorithm 13.

4. Simulation and Experimental Results. After the insertion of the Algorithm 13 in the Chord implementation for *PeerSim* simulator [19], we obtained various experimental results presented in the next sections.

Algorithm 13 Distributed Discovery Algorithm

Require: $Webservice(input, output, goal, TTL)$
Ensure: *service_contract*
  $Receive[Search\_Service(input, output, goal, TTL)]$;
  if (there is a Web service published in the cluster) **then**
    $Get(contract\_service)$;
  else
    $Decrease(TTL)$;
    $Send[Search\_Service(input, output, goal, TTL)]$;
  end if

In the following, we present different case studies by using some characteristic of the network: size (number of super peers), number of nodes that join/leave the network, stabilizations..etc. The observations of the different input/output results are summarized in Table 4.1.

TABLE 4.1
Observations of the hops number in the Chord ring.

Number of super peers	Super peers Join/leave the Network	Number of hops	Min/Max of hops	Hops Average
500	200/250	2049	0/13	3
1000	250/500	3094	0/13	3
2000	500/750	3169	0/15	4
5000	750/1000	4187	0/15	6
10 000	1000/1250	5322	0/14	6
50 000	2000/1000	6900	0/15	8

4.1. Number of Failures. The main difference between the original implementation of Chord protocol and our implementation is that we mark the discovery request by a TTL. For these reason, we made a simulation to detect the number of failures where the request doesn't achieve its goal because of the end of TTL. Furthermore, the number of super peers that can be leave the network or when they can break down. To calculate the initial TTL for each experience, we have added a method named `durationTime` to the classes `ChordProtocol` and `CreateNw`. The java code of this method is presented in the following listing:

LISTING 2
java code of the durationTime method

```
public long durationTime(long duration) {
 startTime = System.nanoTime();
.............................
 endTime = System.nanoTime();
duration= endTime-startTime;
return duration;
}
```

The main objective of this method is to evaluate the maximum time to cover the entire network. It's calculated as follow:

MaxTTL $= \sum_{i=1}^{n}(durationTime(super - peer(i))) + \sum_{j=1}^{m} Time\_create\_new\_peer(j)$. Where:

- n : number of super peers
- m : number of new super peers that joined the network

The "MaxTTL" (or TTL) is evaluated before the integration of the Algorithm 13 in the original implementation of Chord. Table 4.2 summarizes the evaluation of the TTL in correspondence with the network size.

In addition, before the integration of the Algorithm 13 in the source code of Chord protocol, we have obtained the observations presented in Table 4.3.

TABLE 4.2
Evaluation of the TTL.

Network size	MaxTTL (nano second)	TTL (second)
500	5 521 468 554	$\simeq 6$
1000	10 756 845 215	$\simeq 11$
2000	21 032 225 142	$\simeq 21$
5000	45 689 998 569	$\simeq 46$
10 000	112 231 543 268	$\simeq 112$
50 000	564 523 669 884	$\simeq 565$

TABLE 4.3
Number of failures without discovery algorithm.

Number of super peers	Super peers Join/leave the Network	observations	Number of failures	Stabilization
500	200/250	828	2	126
1000	250/500	1200	5	220
2000	500/750	3169	7	608
5000	750/1000	4187	8	828
10 000	1000/1250	5322	10	1750
50 000	2000/1000	6900	13	5458

After the integration of the discovery Algorithm in the Chord implementation, we have obtained the results presented in Table 4.4.

TABLE 4.4
Number of failures after the integration of Algorithm13

Number of super peers	Super peers Join/leave the Network	observations	Number of failures	Stabilization
500	200/250	828	9	544
1000	250/500	1200	13	627
2000	500/750	3169	15	700
5000	750/1000	4187	18	965
10 000	1000/1250	5322	25	2078
50 000	2000/1000	6900	32	7865

According to the experimental results summarized in the Table 4.4, we can conclude that the TTL increases the number of failures, because the request cannot continue its path in the Chord ring when the TTL is finished. However, in Table 4.3, the request stops, only, if a super peer (receiver or intermediary) is break down or leaves the network. Figure 4.1 schematizes a comparison between the number of failures before and after the integration of Algorithm 13 (where we mark the request by a TTL).

From the results presented in Figure 4.1, we can demonstrate that, after the implementation of the discovery algorithm of Web services (Algorithm 13), the TTL has increase the number of failures. In comparison with the original implementation of Chord protocol, the integration of the TTL becomes the Web services discovery operation more realistic. Furthermore, we have observed that the most practical number of super peers is 2000, where the percentage of the success request is approximately 50%. In this context, it's important to clarify that as a first step, we studied only the influence of the TTL on the success rate of the discovery operation. For this reason, the described simulation and experimental results offer better values in the non-TTL set of experiments. An additional work is necessary to decrease the number of failures in the TTL results.

4.2. Stabilizations. Another important characteristic that we have experimented is the stabilization of the network after the implementation Algorithm 13. It is presented by the number of operations realized per a super peer to update the DHT table. Figure 4.2 demonstrates the number of stabilizations before and after the addition of the discovery algorithm to the Chord implementation.

In Figure 4.2, we observe that the call of the stabilization method, by a super peer, increases when the size of the network augments. Once the number of super peers exceeds 10000, the network becomes non-stable.

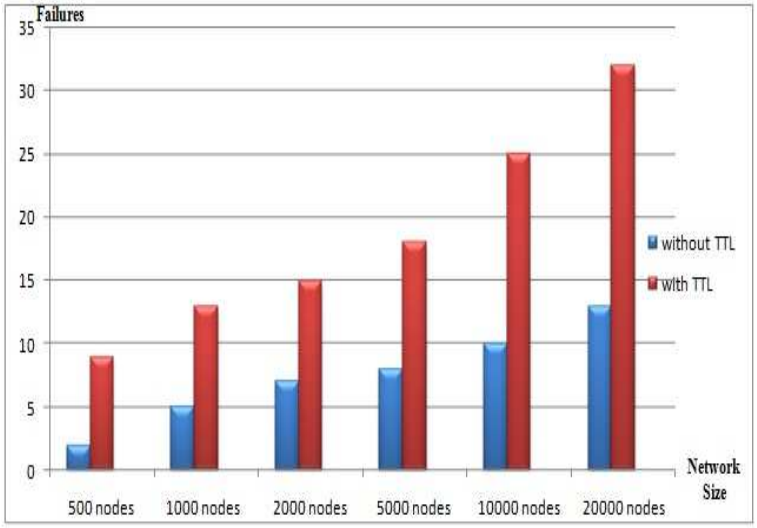


FIG. 4.1. *Number of failures with and without TTL*

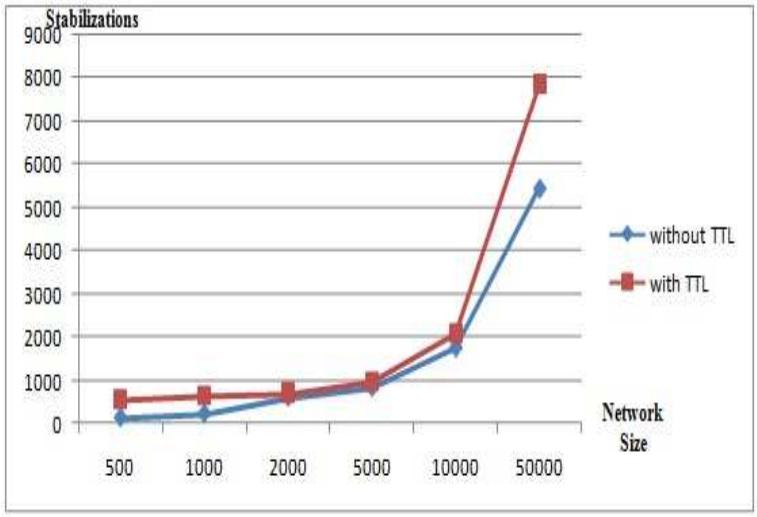


FIG. 4.2. *Number of stabilizations*

For this reason, the super peers call the stabilization method frequently. In addition, we note that, when the number of super peers is between 2000 and 5000, the number of stabilizations is very close in the tow cases. Most precisely, the number of calls of stabilization method is nearly the same when the number of nodes is 2000. This result confirms the conclusion that we had already deduced in the previous subsection.

4.3. Traffic generation. The last propriety that we have analyzed, in this experimental study, is the traffic generation. This last is defined by the number of sent requests generated by the nodes that research a particular Web service. Figure 4.3 presents the number of observations with and without integration of Algorithm 13.

In this case, we detect that the difference between the two results is almost negligible. This conclusion is resulted because the discovery algorithm traits the live time of a request but not the number of sent requests. This is generated arbitrary by the class *"trafficGnerated"* implemented originally by the Chord protocol. However, from a more precise observation, we can view that the exactitude between the two results is inspected

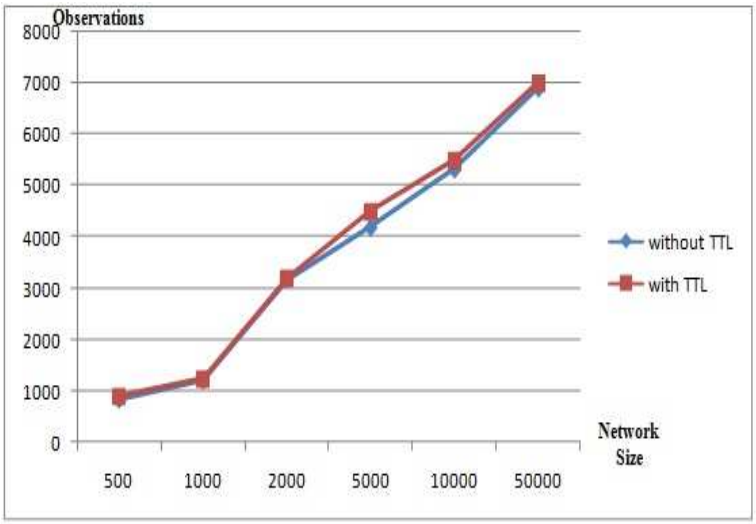


FIG. 4.3. *Number of observations in function of the size of the network*

when the number of super peers is less than 2000.

5. Related works. Firstly, it's important to clarify that this paper doesn't focus on a new family of P2P routing schemes or an extension version of Chord, like the works presented in [5, 8, 16]. Our work presents a new distributed approach for semantic Web services discovery. In this context, several research works improved the decentralized discovering methods of Web services in the P2P networks. For example, a number of centralized and P2P Web service discovery methods have been proposed in the area of the Web services composition, Web services based business process management and Web services discovery in P2P Clouds [13]. In comparison with our previous works [10, 11], where we have employed the unstructured P2P paradigm, in this work, we have used Chord, which is a structured P2P protocol. Another difference is that in our previous work [10], we have proposed a number of algorithms that realized the discovery and the composition of Web services in the same time. The proposed solution has posed some problems in term of convergence and the complexity of the algorithms. In contrast, this article presents an algorithm, which intended the discovery operation only. However, the proposed algorithm has proved its scalability and doesn't influence the stabilization and the traffic generation of Chord protocol.

Another related work is that of Z. Zhengdong et al [39], that designs a localized mechanism of semantic Web services in CAN-based P2P networks to ensure that each service is registered in a specific node of the public Web. The registration services of public Web nodes are divided by the area and shared by all nodes. This work processes the CAN-based P2P networks; in contrast, in our work, even we presented a simple discovery algorithm; the proposed solution is generic and can be extended to a multi-layer Chord model like in [5].

In the same area, F. Mandreoli et al. [18] present the architecture of FLO2WER, which is a framework that supports large scale interoperation of semantic Web services in a dynamic and heterogeneous P2P context. They have adopted a hybrid approach that exploits the advantages of centralized registries for service discovery and composition, as well as the dynamism and the scalability of non-structured P2P networks. The main idea of FLO2WER framework is that, while decentralizing the knowledge of what specific services are available in the system, they keep centralized knowledge of what objectives may be satisfied within the network namely Goals. Each Goal specifies therefore a sub network of specific services, and it is stored in an appropriate repository, called Goal Repository. However, it is not described and detailed how the goals have been discovered. Moreover, the use of a central repository of goals is similar to central discovery methods based on Web services functionalities. In contrast, our discovery method combines decentralized and centralized solution in the same time.

In another work, T. Essafi et al. [34] present a scalable P2P approach to service discovery using ontology. This work incorporates input/output matching algorithm proposed in paper [23] and extends the solution

described in paper [22] by adding an encoding that locates servers in a P2P network to simplify rerouting of query messages. Idem to the precedent work [24], this project adopts only network centralization and hierarchy. However, the main objective of our work is to ensure dynamic Web service discovery. Also, in this work, they still relay on the old DAML-S, which proved to be less flexible than the new OWL-S.

One of the adjacent researches works is that of O. D. Sahin et al [27] where they proposed a similar architecture of us. Though, this paper presents the different semantic techniques for Web services discovery in P2P Chord network. It was very poor in term of experimental results and the improvement of the original Chord implementation. In contrast, our work focuses on Chord proprieties before and after the implementation of the algorithm (more precisely, the influence of the TTL).

6. Conclusion. In this paper, we proposed a P2P based discovery approach for semantic Web services in a large distributed environment. In this approach we employed Chord, which is a structured P2P protocol that improved a high degree of scalability. At this purpose, we adopted a hybrid architecture that exploits the advantages of the decentralized design of the Chord protocol and the clustering organization of the different peers.

This architecture contains two main components: a common central ontology and a P2P network. To guarantee the semantic interoperability in such large environment, which characterized by high degree of heterogeneity, we supposed that the participant peers offer semantic descriptions of their proposed Web service by using the same ontology. This proposition ensures the interoperability and facilities the discovery operation in the P2P network.

The main contribution of our work is the proposition of distributed algorithm for semantic Web services discovery. To achieve this goal, we have updated the original Chord implementation (for Peersim simulator) by injecting the proposed algorithm. The main change that we have realized is the addition of the TTL to the sent requests.

After the simulation, we have obtained experimental results about the effect of the TTL in term of number of failures, stabilization and the traffic generation in the network. According to the achieved results, we have concluded that the TTL increases the number of failures, because the request cannot continue its path in the Chord ring when the TTL is finished. However, for the network stabilization and the traffic generation, the implementation of the discovery algorithm doesn't strongly influence the network. For the network stabilization, we have observed that the call of the stabilization method increases when the number of super peers exceeds 10000 nodes. Though, for the last propriety that we have analyzed (traffic generation), we have detected that the difference between TTL-Chord and original Chord is almost negligible.

However, this task presents the first step of this work. In the future, we plan to optimize the proposed algorithms by the following characteristics:

- Evaluation of the quality of the discovered services (QoS),
- Improvement of the proposed algorithm to discover composite Web services,
- Proposition of a probabilistic algorithm that ameliorates the proposed one.

As a short objective, we are, now, trying to apply the proposed solution for semantic Web services discovery in a P2P Cloud environment.

REFERENCES

[1] Tien Tuan AnhDinh, GeorgiosTheodoropoulos, and Rob Minson. Evaluating large scale distributed simulation of p2p networks. In *12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*, pages 51–58, Vancouver, Canada, October 2008.

[2] The annotated Gnutella protocol specification v0.4. http://rfcgnutella.sourceforge.net/developer/stable/, 2001.

[3] Bawa S. Batra, S. Review of machine learning approaches to semantic web service discovery. *Advances in Information Technology*, 1(3):146–151, August 2010.

[4] Fatih Emekçi, Ozgur D. Sahin, Divyakant Agrawal, and Amr El Abbadi. A peer-to-peer framework for web service discovery with ranking. In *ICWS*, pages 192–199, 2004.

[5] Yung-FaHuang Eric Jui-LinLu and Shu-ChiuLu. Ml-chord: A multi-layered p2p resource sharing model. *Journal of Network and Computer Applications*, 32:578–588, 2009.

[6] Ronaldo A. Ferreira, Murali Krishna Ramanathan, Asad Awan, Ananth Grama, and Suresh Jagannathan. Search with probabilistic guarantees in unstructured peer-to-peer networks. In *Peer-to-Peer Computing*, pages 165–172, 2005.

[7] OWL-S: Semantic Markup for Web Services. http://www.w3.org/submission/owl-s/, 2004.

[8] Mikael Hammar Alberto Negro Gennaro Cordasco, Luisa Gargano and Vittorio Scarano. Fchord:improved uniform routing on chord. In *11th Colloquium on Structural Information and Communication Complexity (SIROCCO 2004),Lecture Notes in Computer Science,Vol. 3104*, 2004.

[9] Mohamed Gharzouli and Mahmoud Boufaïda. A generic p2p collaborative strategy for discovering and composing semantic web services. In *ICIW*, pages 449–454, 2009.

[10] Mohamed Gharzouli and Mahmoud Boufaïda. A distributed p2p-based architecture for semantic web services discovery and composition. In *NOTERE*, pages 315–320, 2010.

[11] Mohamed Gharzouli and Mahmoud Boufaïda. Pm4sws: A p2p model for semantic web services discovery and composition. *Journal of Advances in Information Technology, Special Issue on Advances in P2P Technology*, 2(1), 2011.

[12] Jianqiang Hu, Changguo Guo, Huaimin Wang, and Peng Zou. Web Services Peer-to-Peer Discovery Service for Automated Web Service Composition. In *ICCNMC*, pages 509–518, 2005.

[13] Zhongzhi Shi Jing Zhou, Nor Aniza Abdullah. A hybrid p2p approach to service discovery in the cloud. *I.J. Information Technology and Computer Science*, 3(1):1–9, 2011.

[14] Peep Küngas and Mihhail Matskin. Semantic web service composition through a p2p-based multi-agent environment. In *AP2PC*, pages 106–119, 2005.

[15] Dongsheng Li, Xicheng Lu, Yijie Wang, and Nong Xiao. Efficient search in gnutella-like "small-world" peer-to-peer systems. In *GCC (1)*, pages 324–331, 2003.

[16] Shiming Zheng Wenjie Sun Li Chen, Zilin Song and Zhanfeng Wang. A model of web service discovery based on balancechord. *Journal of Computational Information Systems*, 7(7):2241–2247, 2011.

[17] Jie Liu and Hai Zhuge. A semantic-link-based infrastructure for web service discovery in p2p networks. In *Special interest tracks and posters of the 14th international conference on World Wide Web*, WWW '05, pages 940–941, New York, NY, USA, 2005. ACM.

[18] Federica Mandreoli, Antonio Massimiliano Perdichizzi, and Wilma Penzo. A p2p-based architecture for semantic web service automatic composition. In *DEXA Workshops*, pages 429–433, 2007.

[19] Alberto Montresor and Mark Jelasity. Peersim: A scalable p2p simulator. In *Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09)*, 2009.

[20] Hajar Mousannif, Ismail Khalil, and Gabriele Kotsis. The cloud is not 'there', we are the cloud! *IJWGS*, 9(1):1–17, 2013.

[21] WSMO: Web Service Modeling Ontology. http://www.w3.org/submission/wsmo/, 2004.

[22] Kawamura T. Payne T.R. Sycara K. Paolucci, M. Semantic matching of web services capabilities. In *First International Semantic Web Conference (ISWC)*, pages 333–347, Sardinia, Italy, 2002.

[23] Sycara K. Nishimura T. Srinivasan N. Paolucci, M. Using daml-s for p2p discovery. In *International Conference on Web Services (ICWS)*, pages 203–207, Las Vegas, Nevada, USA, 2003.

[24] Mike P. Papazoglou, Bernd J. Krämer, and Jian Yang. Leveraging web-services and peer-to-peer networks. In *CAiSE*, pages 485–501, 2003.

[25] Gnutella protocol Development v0.6. http://rfc-gnutella.sourceforge.net/src/rfc-06-draft.html, 2002.

[26] K. Rageb. An Autonomic ¡K, D¿-Interleaving Registry Overlay Network for Efficient Ubiquities Web Services Discovery Service. *Journal Information Processing Systems (JIPS)*, 4(2), 2008.

[27] Ozgur D. Sahin, Cagdas Evren Gerede, Divyakant Agrawal, Amr El Abbadi, Oscar H. Ibarra, and Jianwen Su. SPiDeR: P2P-Based Web Service Discovery. In *ICSOC*, pages 157–169, 2005.

[28] Winterhalter C. Schmidt, A. User context aware delivery of e-learning material: Approach and architecture. *Universal Computer Science (JUCS)*, 10, 2004.

[29] METEOR-S: Semantic Web services and Process. http://lsdis.cs.uga.edu/projects/meteor-s/, 2004.

[30] SOAP specification. http://www.w3.org/tr/soap/, 2007.

[31] WSDL specification. http://www.w3.org/tr/wsdl20, 2007.

[32] Morris R. Karger D. Kaashoek M.F Balakrishnan H. Stoica, I. Chord: A scalable peer-to-peer lookup service for internet applications. In *ACM SIGCOM*, pages 149–160, California, USA, 2001.

[33] Semantic Web Central: Project sws tc. http://projects.semwebcentral.org/projects/sws-tc/, 2006.

[34] N. DORTA T. ESSAFI and D. SERET. A scalable peer-to-peer approach to service discovery using ontology. In *Proc of 9th World Multiconference on Systemics, Cybernetics and Informatics*, Orlando, 2005.

[35] UDDI:. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=uddi-spec, 2005.

[36] Xia Wang and Wolfgang A. Halang. *Discovery and Selection of Semantic Web Services*, volume 453 2013. Springer-Verlag Berlin Heidelberg, studies in computational intelligence edition, 2013.

[37] W3C Working group Web services Architecture. http://www.w3.org/tr/ws-arch/, 2004.

[38] WS-Gloss:. http://www.w3.org/tr/ws-gloss/, 2004.

[39] Yahong H. Ronggui L. Weiguo W. Zengzhi L. Zhengdong, Z. A p2p-based semantic web services composition architecture. In *IEEE International Conference on e-Business Engineering*, pages 403–408, Macau, China, 2009.

Edited by: Dana Petcu
Received: Sep 1, 2013
Accepted: Mar 4, 2014

DOI 10.12694/scpe.v15i1.969
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 15, Number 1, pp. 101–119. http://www.scpe.org

A NEW DECENTRALIZED PERIODIC REPLICATION STRATEGY FOR DYNAMIC DATA GRIDS*

HANÈNE CHETTAOUI† AND FAOUZI BEN CHARRADA‡

Abstract. Data grids provide scalable infrastructure for storage resource and data files management, which support data-intensive applications that need to access to huge amount of data stored at distributed locations around the world. The size of these data can reach the scale of terabytes or even petabytes in many applications. These applications require reaching several main goals, namely efficient accessing, storing, transferring and analyzing a large amount of data in geographically distributed locations. In this situation, replication is a general and simple technique used in data grids to achieve these goals. Indeed, it has as main purposes improving data access efficiency, providing high availability, decreasing bandwidth consumption, improving fault tolerance and enhancing scalability. In this paper, we propose a new classification of replication strategies through two complementary criteria as well as a survey of the induced categories of strategies. In addition, we introduce a new decentralized periodic replication strategy for dynamic data grids assuming limited storage for replicas, called DPRSKP, which stands for Decentralized Periodic Replication Strategy based on Knapsack Problem. This strategy takes into consideration the changing availability of sites. DPRSKP is based on two polynomial-time complexity algorithms. The first one starts by selecting the best candidate files for replication while the second places them in the best locations. The replication problem in DPRSKP is formulated according to the Knapsack problem. In addition, DPRSKP extends the well known LRU and LFU strategies. The simulation experiments were carried out using OptorSim and a dynamic period rather than a static one. The obtained results show that DPRSKP can effectively improve response time, bandwidth consumption, remote file accesses number and local file accesses number as compared with other replication strategies.

Key words: data grid, replication, knapsack problem, strategy, period, decision taking, complexity

119

1. Introduction and motivations. Nowadays, a huge amount of data is generated in many fields of science and engineering, some of which are geographic information science [43], astrometry [45], medical [32], and remote instrumentation [47]. These produced data need to be distributed around the world for the sake of data sharing and collaboration. Hence, ensuring an efficient and fast access to such massive data is a challenge that must be addressed. The data grid, which represents a type of grid computing [15], is a solution for this problem. A data grid connects a collection of hundreds of geographically distributed computers and storage resources located in different parts of the world to facilitate sharing of data and resources [23]. Biomedical Informatics Research Network (BIRN)[1], Large Hadron Collider (LHC)[2] and, the European DataGrid Project (EDG)[3] are some examples of existing data grids.

In this paper, we focus on data grids. Since the data represent the most important resource in data grids, how to decrease access time, reduce the bandwidth consumption and improve the data availability and system reliability have become challenging tasks. Replication is a key technique often used to achieve these tasks. The general idea of replication is to create multiple copies of the same data in several storage resources to improve response time, reduce the bandwidth consumption, increase data availability, and to improve system reliability. Currently, data replication in grids mostly deals with file replication [20]. Several replication strategies have been proposed in the literature. A replication strategy must answer the following questions [9, 21, 38]:

1- Which files must be replicated?
2- Where to place the candidate files for replication (*i.e.*, the new replicas)?
3- How to select the best replica of a file among many replicas available in the grid?

In the literature, replication strategies are categorized according to several criteria. Indeed, replication strategies can be classified according to the adopted grid model. We then distinguish between strategies dedicated for hierarchical grids and other dedicated for Peer-To-Peer or hybrid grids. In addition, replication strategies can be classified according to the taking decision type. Hence, two strategy types appeared which are centralized and decentralized strategies. On the one hand, a centralized replication strategy is based on a central site that collects all information about files and grid sites. The central site makes the replica and

*This paper is a largely extended version of our work proposed in [14].

†Department of Computer Sciences, Faculty of Sciences of Tunis, Tunis, Tunisia (hanene1ch@gmail.com).
‡Department of Computer Sciences, Faculty of Sciences of Tunis, Tunis, Tunisia (f.charrada@gnet.tn).

[1]BIRN: http://www.birncommunity.org/
[2]LHC accelerator project: http://lhc.web.cern.ch/lhc/
[3]EU Data Grid Project: http://www.eu-datagrid.org/

placement decisions. On the other hand, in a decentralized approach, all grid sites keep files track in order to decide which files to replicate or to remove locally. We also find static or dynamic replication strategies. A static replication way decides the distribution files initially and retains files location during the grid running. However, in a dynamic replication type, replicas are automatically created and deleted according to changes in the grid users' behavior.

The majority of strategies on data replication have considered only static grids (having invariable number of sites). For several types of grids, this assumption does not reflect reality. Indeed, new sites can join the grid and others can leave, to join it possibly again. For this reason, the dynamicity becomes a fundamental criterion to establish an effective replication strategy.

In this paper, we propose a new decentralized periodic replication strategy, called DPRSKP (Decentralized Periodic Replication Strategy based on Knapsack Problem) for dynamic data grids. In fact, DPRSKP has two main features:

- the storage space of each site is supposed to be limited, contrary to several approaches of the literature which suppose it unlimited.
- the number of grid sites is variable in the sense that sites can leave or join the grid at any time.

In our strategy, we will address these issues characterizing the grid dynamicity. For each site, the DPRSKP strategy selects files to be replicated or deleted based on an adaptation of the well known Knapsack problem [22, 30]. We then evaluate the benefit of our strategy through simulations. The obtained experiment results, using OptorSim, show that our strategy outperforms other replication strategies in terms of several parameters. It is important to note that, as this will be shown hereafter, the proposed strategy is periodic-fee in the sense that it can also be used as a non periodic strategy.

The remainder of the paper is organized as follows. Section 2 gives an overview of previous work on data grid replication. Section 3 presents our replication strategy DPRSKP as well as an illustrative example of its different components. Section 4 describes a study of the complexity of our proposed strategy. Section 5 provides the simulation framework and the obtained experimental results. The last section summarizes our contributions and depicts future work.

2. Related work. Replication in data grids is a technique used to reduce the response time and the bandwidth consumption. Replication also increases data availability thereby enhancing system reliability. In this section, we classify and survey the main replication strategies of the literature.

We propose in this paper another type of classification according to the following two complementary criteria:

- Periodicity: this criterion specifies when the replication strategy is triggered, at each demand of a file or after a given period of time.
- Taking decision type: this criterion distinguishes between centralized and decentralized replication strategies.

As a consequence, replication strategies in the literature can be categorized as follows:

- A non periodic replication strategy is triggered by the requesting site when the needed file is not found locally and assumes limited storage for replicas. In that case, each strategy offers a replacement strategy when there is not enough space to replicate a new file. For this type of strategy, the decision about replicating or deleting files is taken by each requesting site in the grid. The associated strategies are then called non periodic decentralized strategies.
- A periodic replication strategy is triggered at each period. In general, these strategies assume unlimited replica storage. For this type of strategies, the selection of files to replicate and of sites to place them is done either by a central site or by the requesting site. We then distinguish between centralized periodic and decentralized periodic replication strategies.

The proposed classification of replication strategies is depicted in Fig. 2.1.

The following two subsections describe the main centralized and decentralized periodic and decentralized non periodic strategies of the literature.

2.1. Periodic strategies.

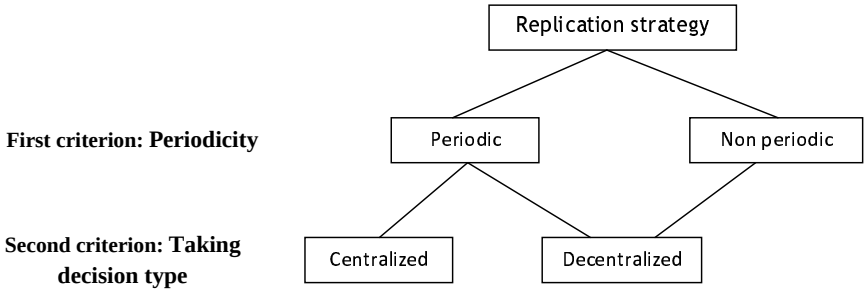


FIG. 2.1. *Classification of replication strategies*

2.1.1. Centralized strategies. This part exposes according to the chronological order the centralized periodic replication strategies where the replication algorithm is triggered by a central site.

In [37, 38], Ranganathan and Foster propose six different replication strategies (periodic and non periodic) for hierarchical data grids. They distinguish between replication and caching. Indeed, the authors consider replication as a server-side phenomenon and caching is assumed to be a client-side phenomenon. The proposed periodic centralized strategies are:

- No Replication: the entire data set is available at the root of the hierarchy. During the grid execution, the number of replicas and their locations remain the same.
- Cascading Replication: once the requests number of a file exceeds a given threshold, a replica is created at the next level which is on the path to the best client.

All these strategies take into account the requests number of each file to identify files to replicate and sites where to place them. In addition, a replacement strategy is triggered in the case there is a shortage of storage space at the selected site to replicate a new file. The least popular file is expunged. If more than one file are equally unpopular, the oldest file is removed.

In [34], Rahman *et al.* propose four replication algorithms based on utility and risk. The utility and risk are calculated using the current network load and the user requests. All these algorithms select one site to host the new replica. An improvement is proposed in [35, 36] by selecting p sites to place the new file using three models, namely p-median, p-center and multi-objective. Note that the number p is a parameter specified by the grid administrator.

In [18, 19], the authors propose a replication strategy by organizing the data into several data categories that it belongs to. They consider the requests number of each file to decide which one needs to be replicated. In addition, they take into account the file size and the bandwidth width to choose the best place for the new replica.

To trigger replica creation, Al Mistarihi and Yong [21] present a replication strategy for data grids that uses the requests number of each file as well as the time starting from the creation date of a replica until the current date. In addition, the candidate file for replication is identified based on the copies number [4] and the requests number of each file. Finally, the new replica is placed at the best site based on the number of requests and the response time.

Rasool *et al.* introduce a replication strategy, called Fair Share Replication (FSR) [40], for hierarchical data grids. The proposed strategy takes into account the requests number of each file to select the candidate file for replication. On the other hand, the replica placement is based on the server workload and the requests number of each client.

Chang *et al.* propose a replication strategy called Least Access Largest Weight (LALW) [13]. The information received at different time intervals has different weights. The concept of half-time is used to set the information weights. LALW works in three steps. The first step finds the candidate file for replication. The second computes how many replicas are required. The last step identifies clusters to host new replicas. These three steps are based on the requests number and the weight of each file.

[4]The copies number of a file is equal to the number of its replicas in the grid.

In [8, 9], Ben Charrada *et al.* propose a replication strategy called Periodic Optimiser for data grids. The first step of Periodic Optimiser identifies the candidate files for replication based on the requests number and the copies number associated to each file. The second step selects the best sites to host new replicas based on the requests number of each file and the availability of each site. The selection of the best replica depends on the availability of sites and the bandwidth.

Lee *et al.* propose a replication strategy called Popular File Replicate First (PFRF) [25] for hierarchical data grids containing a global replica controller (GRC) and several clusters connected to the GRC. After each period, the GRC selects the candidate files for replication based on the popularity weight for each file. The popularity weight is calculated according to the requests number of each file. For each candidate file for replication, the GRC selects the candidate clusters to place it. The candidate file is replicated in a site belonging to the candidate cluster when there is sufficient storage space to accommodate it. Otherwise, PFRF deletes some files which are less popular than the candidate file to be replicated.

2.1.2. Decentralized strategies. This part surveys, w.r.t. the chronological order, the decentralized periodic replication strategies where the replication algorithm is triggered by a requesting site.

As mentioned above (*see* page 102), Ranganathan and Foster propose in [37, 38] six, periodic and non periodic, replication strategies for hierarchical data grids. The authors design a decentralized periodic replication strategy called Best Client. In this strategy, each node maintains a detailed history describing each file that it contains (more precisely the requests number of each file and its requesting nodes). At each period, each node selects the popular files (having a requests number exceeding a given threshold) and a replica is created at the best client that has the largest number of requests for the file.

Ranganathan *et al.* introduce a replication strategy for a decentralized P2P environment [39]. The proposed strategy computes first the number of replicas needed per file based on the file availability and the accuracy of the RLS (Replica Location Service). Then, the best location of each new replica is chosen. The best site is then that maximizing the difference between replication benefits and replication costs.

Suri and Singh present a replication strategy, called DR2 (Two-Stage Dynamic Replication), for data grids [44]. DR2 works in two stages. Indeed, each site starts by selecting the files to replicate and then determines the best sites to transfer these files. The selection of the candidate file for replication takes into account the requests number, the size and the copies number associated to each file. The best replica is determined according to the bandwidth and the number of hops between the grid sites. In addition, LRU [3] is adopted as a replacement strategy.

Cui *et al.* propose a replication strategy called BSCA (Based on Support, Confidence and Access numbers) for hierarchical data grids [17]. After each period, each site selects the candidate files for replication whose the requests number exceeds a given threshold. Each candidate file and those files strongly correlated to it are replicated if there is enough space to accommodate them. Otherwise, the weakly correlated files are removed. If the storage space is still insufficient to replicate these files, BSCA deletes files having a requests number lower than a threshold. It is important to mention that no indication on the threshold choice is made.

2.2. Non periodic strategies. This section provides an overview in the chronological order of decentralized non periodic replication strategies. These strategies often replicate files at the request of any site and for any file. Therefore, a replacement strategy is used when there is not enough space to replicate a new file.

As mentioned above (*see* page 102), Ranganathan and Foster [37, 38] propose non periodic replication strategies for hierarchical grids. These strategies are:

- Plain Caching: the client that requests a file stores a copy locally.
- Caching plus Cascading Replication: this strategy combines Cascading Replication (*cf.* page 103) and Plain Caching strategies. Each client stores the requested files locally. In addition, at each period, each server identifies the popular files and replicates them down the hierarchy.
- Fast Spread: this strategy replicates files at each node along its path to the client.

All these strategies take into account the requests number of each file to identify files to replicate and sites where to place them. In addition, a replacement strategy is triggered in the case there is a shortage of storage space at the selected site to replicate a new file. The least popular file is expunged. If more than one file are equally unpopular, the oldest file is removed.

In [4], a replication strategy called LFU (Least Frequently Used) is introduced. LFU always replicates the requested file in the site where the job needing the replica is running. If there is not enough space to accommodate the replication, the least accessed file in the local storage is deleted. The strategy LRU (Least Recently Used) [3] proceeds in the same way as the LFU strategy, except that the oldest replica is removed.

Park *et al.* [33] introduce a replication strategy, called BHR (Bandwidth Hierarchy based Replication), which is based on bandwidth hierarchy of Internet. Nearest sites are located in the same network region. BHR tries to replicate files that will be used within the same region in the near future. If there is not enough space to accommodate the new replica, the least accessed files that are replicated in other sites within the same region will be deleted. If the available space is still insufficient to host the new replica, those files having a lower number of requests than the new replica are expunged.

In [26, 27], the authors propose a replication strategy called MinDmr (Minimize Missing Data Rate) which performs as follows. If the required file is not present locally, a new replica is created if the available free storage space is large enough. Otherwise, the candidate files for deletion are selected depending upon their weights. The file weight is computed according to the availability, the requests number, the size, and the copies number of the file.

Bsoul *et al.* [10] present a replication strategy named MFS (Modified Fast Spread) that is based on the Fast Spread strategy [37, 38]. MFS considers a network topology in which there is one server node and a number of client nodes. If there is enough storage space to replicate the required file, a new replica is created. Otherwise, a group of replicas needs to be removed if it is less important than the new replica. The file importance depends on the available storage space, the size, and the requests number of the file.

Zhao *et al.* [48] propose a replication strategy called VBRS (Value-Based Replication Strategy) that first calculates a threshold deciding whether the requested file should be replicated or not. The computed threshold depends on the requests number of each file. If the available storage space is insufficient to place the new replica, some files will be removed based on access time, bandwidth and file size.

Sashi and Thanamani [42] propose a strategy, called Modified BHR, to overcome the limitations of the BHR strategy (*cf.* page 105). According to Modified BHR, sites that are located in the same network region are grouped together. The main idea of Modified BHR is to replicate files within a region and to store replica in a site where the file has been accessed frequently. If there is not enough space to replicate the new replica in a site and the replica is duplicated in other sites within the region, no action is done. If there is no duplication, a replacement strategy is executed. It sorts files using the LFU strategy [4]. Then, it deletes files that are duplicated in other sites within the same region. If the available space is still insufficient to hold the new file, unpopular files having a requests number smaller than the new file requests number are deleted.

Ben Charrada *et al.* propose a replication strategy called LWF (Least Weight File) [5] which performs as follows. The required file is replicated in the requesting site if there is enough space to host the new replica. Otherwise, the candidate files for deletion are selected according to their weights. If the weight of the new replica is greater than the sum of the candidate files weights, they will be replaced by the new replica. Note that the file weight takes into account file-related parameters, *i.e.*, its size and requests number, as well as topology-related parameters, namely bandwidth and site availability.

Mehraban *et al.* propose a replication strategy called Prediction Replica Replacement Strategy (noted PRA) [31]. PRA is started when a site requests a file and does not have sufficient space to accommodate it. In this case, the requesting site computes the value of each file stored locally based on the requests number and the copies number of each file. Then, the requesting site determines the candidate files for deletion according to their sizes. Once the candidate files for deletion are selected, they will be sorted in ascending order according to their values. The files having the least values will be removed and the requested file will then be replicated.

In [2], Beigrezaei *et al.* propose a replication strategy called FUZZY_REP improving the modified BHR method [42]. According to the FUZZY_REP strategy, when there is not enough storage space to replicate the new requested file, the potential candidates for replacement will be chosen according to their weight. The weight computation is based on a fuzzy inference system and takes into account two factors, namely the number of replica access and the difference between the current time and the last access time of replica. The candidate replicas for replacement are sorted according to their respective weights in ascending order and, then, minimum weight replicas will be deleted. If the available storage space is still insufficient to place the new one, FUZZY_REP

tries to delete the replicas that were not stored in the other grid sites of the current region.

In [46] Vashisht *et al.* propose a replication strategy called EDRA (Efficient Dynamic Replication Algorithm) adapting agents for data grid. In EDRA, a data grid contains several regions connected with a master node. A number of grid nodes collectively form one sub-region. EDRA tries to place a requested replica in each sub-region. The head node located in regions is responsible for scheduling the jobs to the nodes where the replica is placed. An agent is placed on each head node of the region, which keeps information of the nodes located in the sub-regions. The new replica is placed on the nodes based on two factors, namely high data access frequency and sufficient storage capacity. If there is enough space to accommodate the new file, the replica is placed on the node. If the node has insufficient space, EDRA checks free space in the nearby node within the sub-region and replicates the new file. If there is insufficient space in the sub-region, old files are deleted from the node. The file having less access frequency is thus deleted and is replaced by the new replica.

Mansouri and Dastghaibyfard propose in [29] a replication strategy called Modified Dynamic Hierarchical Replication (MDHR) which is an enhanced version of Dynamic Hierarchical Replication (DHR) [28]. In MDHR, a data grid contains several regions. Each region contains several LAN. When a requested file is not available in the local storage, MDHR selects the best site to replicate the new file. The best site has the highest number of requests of the new file. If there is enough space in the selected site, the new file is replicated. Otherwise, if the new file is available within the LAN of the requested site, the file is accessed remotely. If the file is not available in the same LAN and there is not enough space to replicate it, files existing in the best site as well in the local LAN are deleted using the LRU strategy. Moreover, if the available storage is still insufficient to accommodate the new file, files existing in the best site as well in the local region are deleted using the LRU strategy. If the required space is still insufficient, remaining files in the best site are deleted also using the LRU strategy.

In the next section, we introduce our new strategy for dynamic data grids, called Decentralized Periodic Replication Strategy based on Knapsack Problem (DPRSKP). A dynamic data grid has a variable number of sites *i.e.* new sites can join the grid and others can leave, to join it possibly again.

3. DPRSKP strategy. As mentioned above, the majority of strategies on data replication have considered only static grids *i.e.* those having an invariable number of sites. For several types of grids, this assumption does not reflect the reality. Indeed, new sites can join the grid and others can leave it, to join it possibly again after. For this reason, the dynamicity becomes a fundamental criterion to design an effective replication strategy. In other words, by "dynamicity" we mean that the availability of a given site, *i.e.* the fact that it can be joined or not at given moment, will be taken into consideration in our approach.

In this context, we proposed in [8, 9] a centralized periodic replication strategy called Periodic Optimiser. Centralized replication is based on a central site that collects all information about files and grid sites. The central site makes the replica and placement decisions. However, in a decentralized strategy, all grid sites keep files track in order to decide which files to replicate or to remove locally. The benefit of a decentralized decision is that there is no single point of failure [1]. This motivates our choice of a decentralized way for replicating files. Moreover, in [5], we proposed a decentralized non period replication strategy called Least Weight File (LWF). LWF computes the weight of each file in a site S_i requesting a file F_{new}. In the first step of LWF, these files of S_i are sorted in an ascending order according to their weight. In the second step, LWF searches for the smallest index p of the files list as the sum of their sizes is greater than or equal to the missing space allowing to store F_{new}. Finally, F_{new} will be replicated if the sum of the weights of these p files is less than the weight of F_{new}. However, the choice of p is not performed optimally as shown in the following counter example. Let $Size_{F_1} = 2$, $Size_{F_2} = 3$, $Size_{F_3} = 5$ and $Size_{F_4} = 6$ where $Size_{F_k}$ denotes the size of a file F_k locally stored in the site S_i. Also, let $FileWeight_{F_1} = 4$, $FileWeight_{F_2} = 5$, $FileWeight_{F_3} = 6$, $FileWeight_{F_4} = 10$ and $FileWeight_{F_{new}} = 7$ where $FileWeight_{F_k}$ denotes the weight of a file of S_i (for F_1, F_2, F_3 and F_4) and that to be replicated (F_{new}). Suppose the missing space is 5, *i.e.*, $Size_{F_{new}}$ - $S_i.FreeSpace = 5$. LWF chooses F_1 and F_2 whose total weight is 9 (> 7). Therefore, F_{new} will not be replicated. On the other side, if F_3 is chosen ($FileWeight_{F_3} = 6$), then F_{new} is replicated and F_3 is deleted from S_i since (i) $FileWeight_{F_3} = 6 < 7$, (ii) $Size_{F_3} = 5$.This counter-example can be avoided through making the replication problem as a Knapsack problem with bivalent variables [22, 30, 41].

3.1. Proposed strategy. In this subsection, we present our decentralized periodic replication strategy DPRSKP for dynamic P2P data grids assuming limited replica storage. DPRSKP aims at increasing files availability, improving response time and reducing bandwidth consumption.

DPRSKP performs as follows. After each period, each site S_i identifies the candidate files for replication. These files are replicated if there is enough storage space. However, if the available storage space is insufficient to hold these files, the site S_i should place the files with the highest priority among the candidate files and local ones of S_i by using the Knapsack problem. Thus, our strategy mainly deals with the following steps:

1. Selection of candidate files for replication;
2. Placement of a set of files in the site.

It is worth mentioning that, for answering the third question required by a replication strategy (*cf.* page 101), we rely on the formula proposed in [9].

3.1.1. Selection of the candidate files for replication. At each period, each site S_i identifies the candidate files for replication based on the following parameters:

- $\#Request_{S_i,F_k}$: the number of times that a file F_k has been requested by the site S_i.
- $\#Replica_{F_k}$: the number of available copies of a file F_k on the whole grid.

We consider that a file F_k needs to be replicated in S_i if it has been requested many times by S_i and there are not enough copies of F_k in the grid. For that purpose, a metric is introduced, called average value of a file F_k for a site S_i, noted $AVG\_VALUE_{S_i,F_k}$, and defined as follows:

$$AVG\_VALUE_{S_i,F_k} = \frac{\#Request_{S_i,F_k}}{\#Replica_{F_k}}$$

The average value of a file F_k for a site S_i represents the average number of requests for a replica of F_k by S_i.

Similarly to the previous metric, we introduce a metric called average value of a site S_i, denoted $AVG\_VALUE_{S_i}$, and defined by the following expression:

$$AVG\_VALUE_{S_i} = \frac{\sum_{k=1}^{n} \#Request_{S_i,F_k}}{\sum_{k=1}^{n} \#Replica_{F_k}}$$

where n is the total number of requested files by the site S_i.

These two metrics are used in Algorithm 3.1.1 for choosing the best candidate files for replication in a site S_i. Algorithm 3.1.1 takes as entry the requests number and the copies number of each file requested by S_i as a list of triplets $S_F = \{(F_k, \#Request_{S_i,F_k}, \#Replica_{F_k}) \mid k = 1..n\}$, where n is the total number of requested files by the site S_i.

In the first loop (*cf.* Lines 5 - 8), we calculate the average value of each requested file by the site S_i. In the second loop (*cf.* Lines 10 - 12), we select the candidate files for replication. A file F_k is considered as a candidate file for replication if its average value exceeds the average value of the site S_i.

3.1.2. Replica placement. After selecting the candidate files for replication in a site S_i, it is necessary to check the available storage space of S_i that contains locally a set of files noted $\mathcal{F}$. If the free storage space is sufficient to hold the set of all candidate files for replication, noted $\mathcal{F}_{ToReplicate}$, then the replication of these files will be conducted without any constraint. If there is not enough storage space, the potential candidate files for replacement will be chosen to place all or part of $\mathcal{F}_{ToReplicate}$ in S_i. As our strategy aims to reduce the response time and the bandwidth consumption, the choice of the potential replacement candidate files is important because they can be requested in the future by the same site S_i.

To achieve these objectives by choosing the best potential candidate files for replacement in a site S_i, we introduce the average number of requests of each file $F_k \in \mathcal{F} \cup \mathcal{F}_{ToReplicate}$ by S_i, noted $\#AVG\_Request_{S_i,F_k}$,

Algorithm 1: Selection of the candidate files for replication

Input: $S_F = \{(F_k, \#Request_{S_i,F_k}, \#Replica_{F_k})\}$
Output: $CandidateFile$: the set of candidate files for replication

```
Begin
  Request_Sum ← 0; /*Computation of the numerator of the formula (3.2)*/
  Replica_Sum ← 0; /*Computation of the denominator of the formula (3.2)*/
  CandidateFile ← ∅;
  For k = 1 to | S_F | do
      AVG_VALUE_{S_i,F_k} ← #Request_{S_i,F_k} / #Replica_{F_k};
      Request_Sum ← Request_Sum + #Request_{S_i,F_k};
      Replica_Sum ← Replica_Sum + #Replica_{F_k};
  AVG_VALUE_{S_i} ← Request_Sum / Replica_Sum;
  For k = 1 to | S_F | do
      If AVG_VALUE_{S_i,F_k} ≥ AVG_VALUE_{S_i} then
          CandidateFile ← CandidateFile ∪ {F_k}
  return CandidateFile
End
```

and defined as follows:

$$\#AVG\_Request_{S_i,F_k} = \frac{\#Request_{S_i,F_k}}{T_{Current} - T_{FirstRequest_{S_i,F_k}} + 1}$$

where:

- $\#Request_{S_i,F_k}$: the total number of requests of F_k by S_i;
- $T_{Current}$: the number of the current period;
- $T_{FirstRequest_{S_i,F_k}}$: the period number of the first request of F_k by S_i.

Then, we associate to each file $F_k \in \mathcal{F} \cup \mathcal{F}_{ToReplicate}$ a file weight (denoted $FileWeight_{F_k}$) defined by the following expression:

$$FileWeight_{F_k} = \frac{Size_{F_k} \cdot \#AVG\_Request_{S_i,F_k}}{BW_{(S_i,S_j)F_k} \cdot P_{S_j}}$$

where:

- $Size_{F_k}$: the size of F_k;
- $\#AVG\_Request_{S_i,F_k}$: the average number of requests of F_k by S_i;
- $BW_{(S_i,S_j)F_k}$: the bandwidth between the requesting site S_i and the site S_j ($j \neq i$) containing the best replica of F_k. The choice of the best replica is done using the formula proposed in [8, 9] and is determined as follows:

$$\underset{j=1}{\overset{|\Xi_{F_k}|}{Max}} \left(BW_{(S_i,S_j)F_k} \cdot P_{S_j}\right)$$

  where Ξ_{F_k} is the set of sites containing a replica of F_k.
- P_{S_j}: the availability of the site S_j defined by the following expression [8, 9]:

$$P_{S_j} = 1 - \frac{\#Failure_{S_j}}{\#SiteRequest_{S_j}}$$

  where $\#Failure_{S_j}$ is the number of times the site S_j is failing (*i.e.*, the number of requests to S_j which are not satisfied) and $\#SiteRequest_{S_j}$ is the total number of times the site S_j has been requested.

For the site S_i, according to the expression (3.1.2), the most popular files situated in unstable sites having a low bandwidth with S_i are the most preferred to be placed in it. Indeed, it is preferred to have such files in

S_i since it requests them several times. On the other side, if they are not locally stored in S_i, each time they will be of need in a job executed by S_i, their use will be high costly due to the low quality of bandwidth with S_i for the best site where they are contained as well as the low availability of these latter.

For each site S_i, the goal of our strategy DPRSKP is to find a set of files $\Omega \subset F \cup \mathcal{F}_{ToReplicate}$ that can be stored in S_i and having a maximum total files weights.

To achieve this goal, our strategy is formulated according to the Knapsack problem to select files to be replicated and those to be deleted. The replication problem is then formulated as follows.

We associate to each file $F_k \in \mathcal{F} \cup \mathcal{F}_{ToReplicate}$ the value $X_k \in \{0,1\}$ defined by:

$$X_k = \begin{cases} 1\text{: } F_k \text{ should be placed in } S_i \\ 0\text{: } F_k \text{ should not be placed in } S_i \end{cases}$$

$$(K) \begin{cases} \sum_{k=1}^{N} (Size_{F_k} \cdot X_k) \leq SE_{S_i} \text{ such that } X_k \in \{0,1\} \\ \sum_{k=1}^{N} (FileWeight_{F_k} \cdot X_k) = Z(Max) \end{cases}$$

where:
N is the cardinality of the set $\mathcal{F} \cup \mathcal{F}_{ToReplicate}$;
$Z(Max)$ is the objective function to maximize;
SE_{S_i} is the storage space of S_i.

3.1.3. Resolution algorithm. The resolution of our replication problem formulated through to the Knapsack problem by exact algorithms using the Branch and Bound method, the polyhedral techniques or the Dynamic Programming can be costly in terms of response time. In such a situation, it would be better to use approximation methods with a reasonable compromise between response time and approximation quality. In general, the approximation algorithms are effective. This is the case of greedy algorithms having generally a low complexity. The idea often used in such algorithms is to keep the option of the local optimum according to some criteria in order to have a global optimal solution [16, 22, 30].
Our proposed algorithm gives an approximate solution that is acceptable given response time constraints. The greedy algorithm 3.1.3 presents the placement of a set of files in a site S_i.

Algorithm 2: Replication of a set of files in a site S_i

```
   Input: S_i, SE_{S_i}, F = {F_1, F_2, ..., F_p}, F_ToReplicate
1  Begin
2    If S_i has enough free space for F_ToReplicate then
3      For Each file F_k of F_ToReplicate do
4        Get F_k from S_j and replicate it in S_i /*S_j denotes the site containing
         the best replica of F_k*/
5    Else
6      For Each file F_k of F ∪ F_ToReplicate do
7        Calculate Efficiency_{F_k} = FileWeight_{F_k} / Size_{F_k}
8      Sort the files of F ∪ F_ToReplicate in descending order according to the
       Efficiency metric in the list denoted SortList
9      AllocateSpace=0
10     For Each file F_k of SortList do
11       If AllocateSpace + Size_{F_k} ≤ SE_{S_i} then
12         X_k ← 1
13         AllocateSpace ← AllocateSpace + Size_{F_k}
14       Else
15         X_k ← 0
16 End
```

In Algorithm 3.1.3, we use the following terminology. We introduce the file efficiency for a file F_k, noted $Efficiency_{F_k}$, as the ratio of the weight to the size of F_k (*cf.* Line 7 of Algorithm 3.1.3). The more the weight

of F_k is large relative to the storage capacity consumption, the more F_k is interesting. To summarize:

- For each file F_k such as $X_k = 1$: F_k will be placed in S_i.
- For each file F_k such as $X_k = 0$:

$$\begin{cases} F_k \textit{ is deleted from } S_i \textit{ if } F_k \in \mathcal{F} \\ F_k \textit{ is not replicated in } S_i \textit{ if } F_k \in \mathcal{F}_{ToReplicate} \end{cases}$$

3.2. Illustrative example. Let us consider the data grid, shown in Fig. 3.1, which contains four sites. The arcs sketch the available bandwidth between sites in Mb/s. The availability of each site is given into brackets. For instance, the availability of S_2 is 0.5. The master files[5] are placed in the site S_0. All the other sites have a storage capacity of 5 Gb. We will apply DPRSKP strategy to the site S_1.

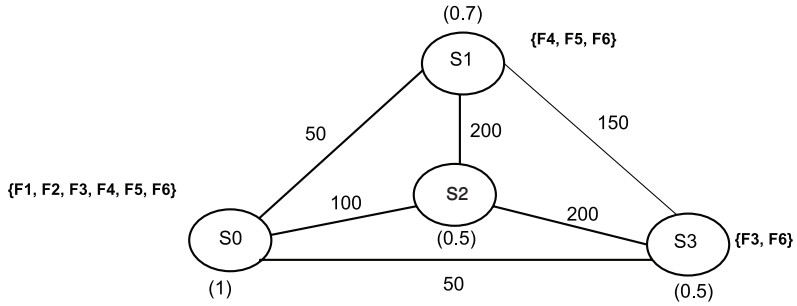


FIG. 3.1. *Topology of the grid*

Step 1: Determination of the candidate files for replication in S_1

Table 3.1 sketches the requested files by the site S_1. The execution of Algorithm 3.1.1 provides:

- The average value of S_1 is $AVG\_VALUE_{S_1} = 300$.
- F_1 and F_3 as candidate files for replication in S_1 since the associated average values are greater than or equal to 300.

TABLE 3.1
The values of $\#Request_{S_1,F_k}$, $\#Replica_{F_k}$ and $AVG\_VALUE_{S_1,F_k}$ for the three files requested by S_1

F_j	$\#Request_{S_1,F_k}$	$\#Replica_{F_k}$	$AVG\_VALUE_{S_1,F_k}$
F_1	1 000	2	500.00
F_2	500	3	166.66
F_3	300	1	300.00

Step 2: Files placement in S_1

Table 3.2 describes the candidate files for replication (F_1 and F_3) and those stored locally (F_4, F_5 and F_6) for the site S_1. We note that the number of the current period ($T_{Current}$) is 10.

We remark that the free space at the site S_1 is not enough to replicate F_1 and F_3. Indeed, the size of the files stored in S_1 is 4 Gb. Therefore, the available free space in S_1 is 1 Gb which is insufficient to replicate F_1 and F_3 because the total size of F_1 and F_3 is 3 Gb. Thus, we need to calculate the weight and the efficiency of these files and those local ones. Table 3.3 sketches the obtained results.

The execution of Algorithm 3.1.3 provides the following result: the selected files to be placed in S_1 are F_1, F_3, F_4 and F_6. Consequently, the file F_5 will be deleted from S_1 while F_1 and F_3 will be replicated in S_1.

[5]A master file contains the original copy of some data sample and cannot be deleted [4].

TABLE 3.2
Characteristics of the files set $\mathcal{F} \cup \mathcal{F}_{ToReplicate}$

F_k	$\#Request_{S_1,F_k}$	$Size_{F_k}$ in Gb	S_j	$T_{FirstRequest_{S_1,F_k}}$
F_1	1 000	2	S_2	10
F_3	300	1	S_3	10
F_4	800	1	S_2	7
F_5	300	2	S_2	6
F_6	700	1	S_3	5

TABLE 3.3
Weight and efficiency of $\mathcal{F} \cup \mathcal{F}_{ToReplicate}$

F_k	$Size_{F_k}$ in Gb	$FileWeight_{F_k}$	$Efficiency_{F_k}$
F_1	2	20.00	10.00
F_3	1	4.00	4.00
F_4	1	2.00	2.00
F_5	2	1.20	0.60
F_6	1	1.55	1.55

3.3. DPRSKP extends the LFU and LRU strategies. A main feature of our strategy DPRSKP is that it allows subsuming the LFU and LRU strategies. To explain this, it is necessary to prove that each file retained by either the LRU or the LFU strategy is also retained by DPRSKP strategy. Recall that:

- The file weight of a file F_k is defined as follows:

$$FileWeight_{F_k} = \frac{Size_{F_k} \cdot \#AVG\_Request_{S_i,F_k}}{BW_{(S_i,S_j)F_k} \cdot P_{S_j}}$$

- the file efficiency for a file F_k is defined as follows:

$$Efficiency_{F_k} = \frac{FileWeight_{F_k}}{Size_{F_k}} = \frac{\#AVG\_Request_{S_i,F_k}}{BW_{(S_i,S_j)F_k} \cdot P_{S_j}}$$

$$= \frac{\#Request_{S_i,F_k}}{(T_{Current} - T_{FirstRequest_{S_i,F_k}} + 1) \cdot BW_{(S_i,S_j)F_k} \cdot P_{S_j}}$$

Thus, it is clear that the ratio defined in the expression 3.3 has a low value when the value of $\#Request_{S_i,F_k}$ is low and the value of $(T_{Current} - T_{FirstRequest_{S_i,F_k}} + 1)$ is high.

In addition, we have:

- In the LFU strategy, the least accessed file in the local storage is deleted. In our proposal, this is also the case for each file F_k having a low value of $\#Request_{S_i,F_k}$.
- In the LRU strategy, the oldest replica in the local storage is removed. In our case, this is done for each the file F_k having a high value of $(T_{Current} - T_{FirstRequest_{S_i,F_k}} + 1)$.

4. Computational complexity of DPRSKP. In this section, we evaluate the theoretical computational complexity in the worst case of the two algorithms described in Sect. 3.

4.1. Complexity of the algorithm determining the candidate files for replication. Let m be the number of requested files by the site S_i. The complexity in the worst case of Algorithm 3.1.1 (*cf.* page 107) is function of this parameter.

- The lines 2, 3 and 4 are simple initializations and thus their complexity is in $O(1)$.

- The loop, between line 5 and line 8, has a complexity of $O(m)$.
- The line 9 has a complexity of $O(1)$.
- The loop, between line 10 and line 12, has a complexity of $O(m)$.

Thus, the complexity in the worst case of Algorithm 3.1.1 is $O(1) + O(m) + O(1) + O(m) = O(m)$.

4.2. Complexity of the files replication algorithm. Let n be the cardinality of the set $\mathcal{F} \cup \mathcal{F}_{ToReplicate}$. The complexity in the worst case of Algorithm 3.1.3 is function of this parameter.

- The block of instructions, between line 2 and line 4, has a complexity of $O(n)$.
- The loop, between line 6 and line 7, has a complexity of $O(n)$.
- The instruction of line 8 has a complexity of $O(n \cdot log(n))$.
- The instruction of line 9 has a complexity of $O(1)$.
- The loop, between line 10 and line 15, has a complexity of $O(n)$.
- The block of instructions, between line 5 and line 15, has thus a complexity of $O(n) + O(n \cdot log(n)) + O(1) + O(n) = O(n \cdot log(n))$.

Therefore, the complexity in the worst case of Algorithm 3.1.3 is $Max(O(n)\ , O(n \cdot log(n))) = O(n \cdot log(n))$.

It is however important to note that the majority of replication strategies have a polynomial-time complexity as shown in [1]. The experiments presented in the next section prove the effectiveness of our strategy DPRSKP.

5. Experiment setup and results. We use the OptorSim simulator [11] to evaluate and compare our strategy DPRSKP with the DR2 [44] and Best Client [37, 38] strategies. We compare our strategy with Best Client and DR2 since these strategies can be applied in P2P grids and are periodic strategies. In addition, Best Client and DR2 are two decentralized replication strategies that are based on the requests number of each file to choose the files needed to be replicated. Moreover and in order to show that our replacement strategy is better than LRU and LFU strategies, we compare DPRSKP to Best Client and DR2 which use LRU and LFU as replacement strategies. We note that we have made extensions to the OptorSim simulator in order to model dynamic grids. Indeed, we have added new classes and methods in the simulator to satisfy our needs. Moreover, we have made some changes to the configuration files to support the grid dynamicity.

5.1. Simulation environment. OptorSim is a simulator written in Java to evaluate the performance of replication strategies in data grids. It adopts the model of EU DataGrid Project[6] for the grid structure [12].

In OptorSim, a grid is composed of a User, a Resource Broker (RB) and several sites. A site contains a Computing Element (CE), a Replica Manager (RM) and a Storage Element (SE). These components are described as follows:

- User: the user submits jobs to the RB according to a pattern.
- Resource Broker: the RB handles the scheduling of the jobs to sites.
- Storage Element: each SE has a size in Mb and contains some files.
- Computing Element: each CE contains a given number of "worker nodes" with a given processing power and having for task to run jobs.
- Replica Manager: the RM performs the movement of data associated with jobs between sites. Moreover, the RM contains a Replica Optimiser (RO) responsible for both replica selection and the creation and deletion of replicas.

In OptorSim, the execution of a simulation is given as below:

1- Initially, the master files are placed in the SEs of the sites as specified in the configuration file. In our strategy, we assume the existence in the grid of a specific site called "Master Site" containing the master files. The Master Site is always connected to the grid.

2- The SEs register these master files in the Replica Catalog (RC).

3- The RB schedules each job to the appropriate site according to a scheduling strategy.

4- The CE of the chosen site in the previous step runs the job and selects the best replicas if the requested file is not present in the SE of the site.

5- The CE saves the transfer history of replicas.

[6]The European DataGrid Project. http://www.edg.org

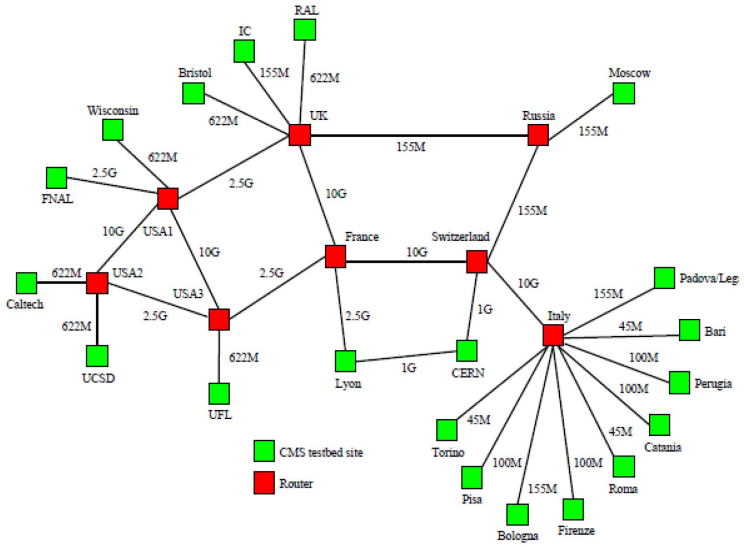


FIG. 5.1. *Testbed grid topology*

6- After a period (number of jobs), each site determines and replicates the candidate files for replication.
7- Each site updates the RC.

5.2. Grid topology. We base all simulation studies on the testbed used during a large scale production effort for the high energy physics experiment CMS [12]. The grid topology (see Fig. 5.1) consists of 20 sites in Europe and the USA. CERN and FNAL have a storage capacity of 100 Gb. Every other site has a CE and initially an empty storage element of a capacity of 50 Gb. CERN stores a master copy of each file. Table 5.1 describes the parameters used in the simulation experiments.

TABLE 5.1
Grid and jobs configurations

Parameter	Value
Size of a single file	1 Gb
Number of files	97
Access pattern	Sequential
Scheduling	access cost for current job + all queued jobs

5.3. Experimental results.

5.3.1. Performance evaluation metrics. In OptorSim, the experiments consist in submitting a variable number of jobs to the RB. At the end of each simulation, we get the response time, the number of remote file accesses, the number of local accesses and the ENU (the Effective Network Usage). This latter parameter [11, 21] is measured as follows:

$$ENU = \frac{N_{remote\ file\ accesses} + N_{file\ replications}}{N_{remote\ file\ accesses} + N_{local\ file\ accesses}}$$

ENU defines the ratio of files transferred to files requested. The parameter ENU is an important parameter in quantifying the effectiveness of any replication strategy in data grids. A low value indicates the efficiency of the adopted replication strategy [11, 24]. In addition, the response time is a good measure for evaluating the effectiveness of a replication strategy. Indeed, each job requests a set of files. If a file is present in the requesting

site, file transfer time is assumed to be zero, otherwise the file must be transferred from the best site containing it. Thus, the response time incorporates the time required to transfer files in addition to the time needed to process jobs. The best replication strategy will have the lowest response time. On the other hand, the number of remote file accesses and the number of local file accesses are two measures to evaluate the effectiveness of a replication strategy. Indeed, the number of remote file accesses represents the number of times where the sites read the necessary files from another one. Thus, the best replication strategy will have the lowest number of remote file accesses. However, the number of local accesses represents the number of times where the sites read the necessary files locally. Thus, the best replication strategy will have the greatest number of local file accesses.

5.3.2. Choice of the period. As mentioned above, DPRSKP is a decentralized periodic replication strategy, *i.e.*, DPRSKP is triggered at each period. The choice of this period is important as proven in [6, 7]. Indeed, the period greatly affects the efficiency of any periodic replication strategy. In this respect, a dynamic period, *i.e.* whose value changes according to the grid behavior, is shown to be much better than a static period, *i.e.* of constant value, whatever the adopted replication strategy. Thus, in our experiments, we use the model of dynamic period.

Now, we briefly sketch the formula used for computing the period (for more details, interested readers are referred to [6, 7]) . To take into account the replicas placement, the $(n+1)^{th}$ period T_{n+1} is defined based on the replications number (denoted $\#Replica_{T_n}$) made during the previous period T_n.

The period T_{n+1} is indeed given by the following formula:

$$T_{n+1} = \frac{T_n}{(\#Replica_{T_n}/T_n) + 1}$$

The next subsection depicts the obtained results which will be discussed in subsection 5.3.4.

5.3.3. Simulation results. This part presents the obtained results for the different metrics.

• *Evaluation of response time:* Table 5.2 shows the response time in milliseconds of each strategy and for different numbers of jobs. The last column indicates the percentage gain of our strategy compared to the other ones and is calculated as follows:

$$Gain\ in\ \% = \frac{Min(Value_{Best\ Client}, Value_{DR2}) - Value_{\mathrm{DPRSKP}}}{Min(Value_{Best\ Client}, Value_{DR2})} \cdot 100$$

The same experiments of Table 5.2 are represented by histograms in Fig. 5.2.

• *Evaluation of the ENU parameter:* Table 5.3 and Fig. 5.3 give the ENU evaluation for the three strategies under the same conditions. The last column of Table 5.3 represents the percentage gain of DPRSKP compared to the other strategies and is calculated according to expression 5.3.3.

• *Evaluation of the number of remote file accesses:* Table 5.4 sketches the number of remote file accesses for each strategy and for different numbers of jobs. The last column computes the gain percentage of DPRSKP compared to Best Client and DR2 according to expression 5.3.3. The results of Table 5.4 are also represented through histograms in Fig. 5.4.

• *Evaluation of the number of local file accesses:* Table 5.5 and Fig. 5.5 show the number of local file accesses of each strategy and for different numbers of jobs. The last column in Table 5.5 indicates the percentage gain of DPRSKP compared to Best Client and DR2. This percentage is computed as follows:

$$Gain\ in\ \% = \frac{Value_{\mathrm{DPRSKP}} - Max(Value_{Best\ Client}, Value_{DR2})}{Max(Value_{Best\ Client}, Value_{DR2})} \cdot 100$$

5.3.4. Discussion. It is clear from the obtained experimental results that the proposed DPRSKP strategy aims at increasing files availability, improving response time and reducing bandwidth consumption. Indeed, we notice, from Table 5.2 and Fig. 5.2, that the response time of our strategy is always better than that of the other strategies. In addition, our strategy offers an ENU much better than Best Client and DR2 whatever the

TABLE 5.2
Response time in milliseconds and associated percentage gain of DPRSKP *compared to Best Client and DR2*

Number of jobs	Best Client	DR2	DPRSKP	Gain (%)
100	3 580	3 070	2 989	2.64
500	8 790	8 623	6 417	25.58
1 000	18 067	16 847	14 897	11.57
1 500	25 922	23 663	19 138	19.12
2 000	41 650	49 504	32 653	21.60
2 500	54 396	76 713	50 203	7.71
3 000	69 500	72 406	56 687	18.44

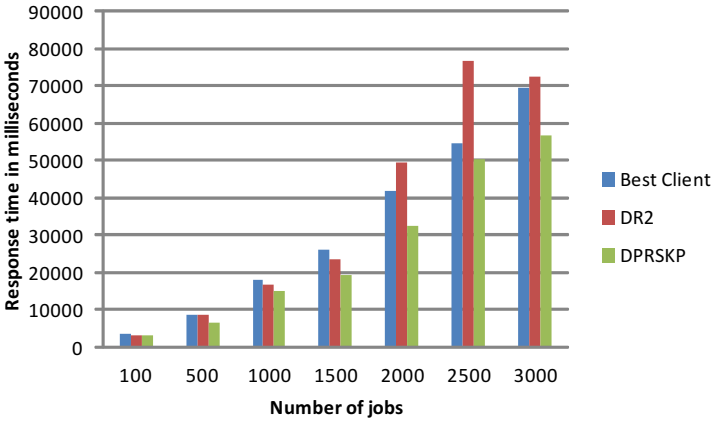


FIG. 5.2. *Response time for the different replication strategies*

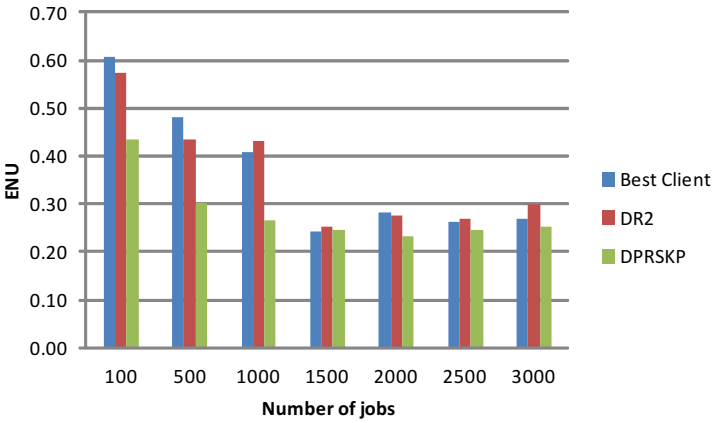


FIG. 5.3. *ENU for the different replication strategies*

number of jobs. Indeed, the obtained average gain of our strategy compared to both others is about 20.74% (*cf.* Table 5.3). On the other hand, we remark from Table 5.4 that the number of remote file accesses of DPRSKP is always lower than Best Client and DR2 strategies. We notice, also from Table 5.5 and Fig. 5.3, that the number of local file accesses of our strategy is always higher than the other strategies. All these obtained results are explained by the fact that the files deletion in the case of a shortage space is not automatic in our approach.

TABLE 5.3
ENU and associated percentage gain of DPRSKP *compared to Best Client and DR2*

Number of jobs	Best Client	DR2	DPRSKP	Gain (%)
100	0.61	0.58	0.44	24.17
500	0.48	0.43	0.30	30.68
1 000	0.41	0.43	0.27	35.02
1 500	0.24	0.25	0.25	-1.38
2 000	0.28	0.27	0.23	15.20
2 500	0.26	0.27	0.25	6.60
3 000	0.27	0.30	0.25	6.22

TABLE 5.4
Number of remote file accesses and associated percentage gain of DPRSKP *compared to Best Client and DR2*

Number of jobs	Best Client	DR2	DPRSKP	Gain (%)
100	801	599	502	16.19
500	3 057	2 437	1 710	29.83
1 000	4 973	5 048	3 291	1.26
1 500	4 509	4 656	4 452	18.33
2 000	6 776	6 281	5 534	11.89
2 500	8 042	8 103	7 328	8.88
3 000	10 041	9 681	9 233	4.63

Indeed, contrary to DR2 and Best Client strategies which always removes files in the case there is a shortage in the storage space at the selected site to replicate a new file, DPRSKP cannot replicate candidate files if their added values are less than those existing in a site and then it preserves those existing files in the site. In other words, DPRSKP deletes files from a given site only if they are qualified as not beneficial in the future and replaces them with more beneficial files. All these results confirm the efficiency of our strategy.

6. Conclusions and future work. Data grids are large scale systems that connect a collection of several machines and storage resources distributed around the world. In such systems, an optimized management and access of distributed resources is the primary purpose. Data grids aggregate a collection of distributed resources placed around the world to enable data intensive applications to share data and resources. As the data is the most important resource in data grids, their quick access time and efficient management have become challenging tasks. Replication is a technique used to ensure these tasks. In this paper, we proposed a new decentralized periodic replication strategy, called DPRSKP, formulated according to the Knapsack problem with storage constraints. Our strategy takes into account the dynamicity of grid sites. Indeed, the dynamicity of sites is an important challenge in grids. We designed first an algorithm which selects the best candidate files for replication based on the requests number and the copies number of each file. We then proposed an algorithm which chooses the candidate files for deletion when there is not enough space to accommodate new files. This algorithm introduces the file efficiency of a file F_k, $Efficiency_{F_k}$, which is equal to $\frac{\#AVG\_Request_{S_i,F_k}}{BW_{(S_i,S_j)F_k} \cdot P_{S_j}}$. As a result, the file efficiency does not depend on the file size. Thus, contrary to the majority of work in the literature, all large and small files have the same chance to be placed in the site. We believe that this notion of file efficiency is at the root of the good performances of our strategy.

In future work, we plan to validate our strategy by deploying the replication algorithm in a real grid environment. In addition, in practice, the files are required simultaneously with others, *i.e.*, there exists a

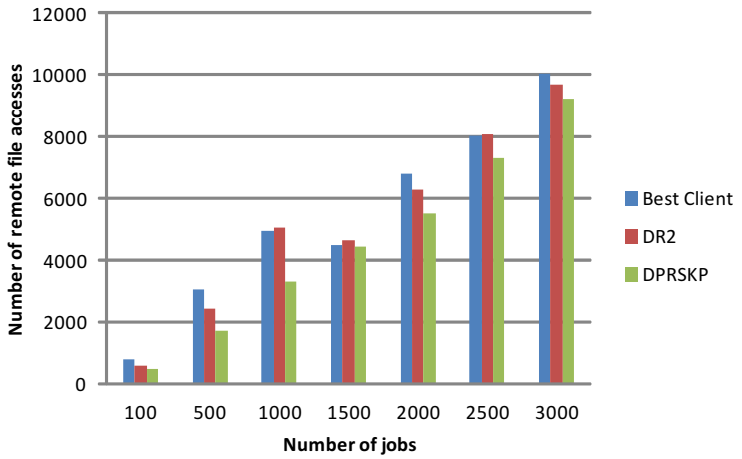


FIG. 5.4. *Number of remote file accesses for the different replication strategies*

TABLE 5.5
Number of local file accesses and associated percentage gain of DPRSKP *compared to Best Client and DR2*

Number of jobs	Best Client	DR2	DPRSKP	Gain (%)
100	596	564	823	38.09
500	3 467	3 374	4 387	26.54
1 000	7 501	6 846	9 721	29.60
1 500	14 964	14 801	15 527	3.76
2 000	18 361	18 636	19 431	4.27
2 500	23 864	23 886	23 957	0.30
3 000	28 641	28 309	28 808	0.58

strong correlation between requested files. Thus, it is possible to investigate replication strategies which consider as granularity a set of correlated files instead of a single file. Moreover, the grid files can be modified by some applications. Since these data files are replicated in grid sites, it is necessary to update them to ensure maintaining coherent copies. It is then important to look for an efficient way for extending the replication strategies proposed in the literature in order to be able to take into consideration file consistency.

REFERENCES

[1] T. AMJAD, M. SHER, AND A. DAU, *A survey of dynamic replication strategies for improving data availability in data grids*, Future Generation Computer Systems, 28 (2012), pp. 337–349.

[2] M. BEIGREZAEI, A. T. HAGHIGHAT AND A. KHADEMZADEH, *A new fuzzy based dynamic data replication algorithm in data grids*, Journal of Basic and Applied Scientific Research, 3 (2013), pp. 350–358.

[3] W. BELL, D. CAMERON, L. CAPOZZA, A. MILLAR, K. STOCKINGER, AND F. ZINI, *Simulation of dynamic grid replication strategies in OptorSim*, Lecture Notes in Computer Science, 2536 (2002), pp. 46–57.

[4] ———, *OptorSim - a grid simulator for studying dynamic data replication strategies*, International Journal of High Performance Computing Applications, 17 (2003), pp. 403–416.

[5] F. BEN CHARRADA, H. CHETTAOUI, AND R. MILI, *A new replication strategy for data grids*, in Proceedings of the 10th IASTED International Conference on Parallel and Distributed Computing and Networks, 2011, pp. 184–189.

[6] F. BEN CHARRADA, H. OUNELLI, AND H. CHETTAOUI, *Dynamic period vs static period in data grid replication*, in Proceedings of the 3rd International Workshop on Simulation and Modeling of Emergent Computational Systems, 2010, pp. 565–568.

[7] ———, *A model for dynamic period in data grid replication*, International Journal of Modeling and Optimization, 3 (2013), pp. 163–166.

[8] ———, *An efficient replication strategy for dynamic data grids*, in Proceedings of the 5th International Conference on P2P,

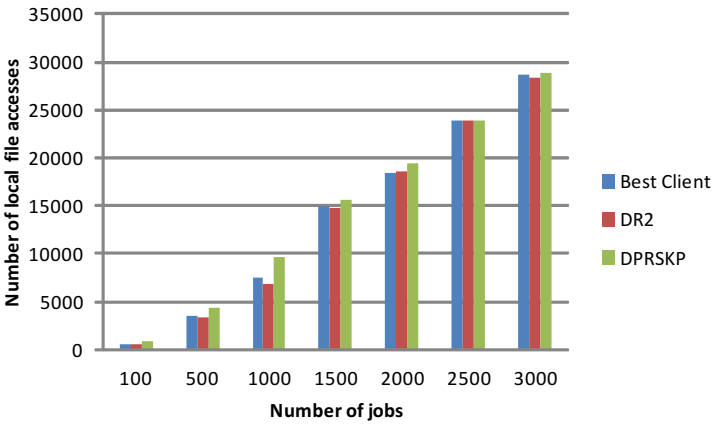


FIG. 5.5. *Number of local file accesses for the different replication strategies*

Parallel, Grid, Cloud and Internet Computing, 2010, pp. 50–54.

[9] ———, *An efficient replica placement strategy in highly dynamic data grids*, International Journal of Grid and Utility Computing, 2 (2011), pp. 156–163.

[10] M. BSOUL, A. AL-KHASAWNEH, Y. KILANI, AND I. OBEIDAT, *A threshold-based dynamic data replication strategy*, The Journal of Supercomputing, (2010).

[11] D. CAMERON, R. CARVAJAL-SCHIAFFINO, J. FERGUSON, A. MILLAR, C. NICHOLSON, K. STOCKINGER, AND F. ZINI, *OptorSim v2.1 installation and user guide*, tech. report, 2006.

[12] D. CAMERON, A. MILLAR, C. NICHOLSON, R. CARVAJAL- SCHIAFFINO, F. ZINI, AND K. STOCKINGER, *Analysis of scheduling and replica optimisation strategies for data grids using OptorSim*, Journal of Grid Computing, 2 (2004), pp. 57–69.

[13] R. CHANG, H. CHANG, AND Y. WANG, *A dynamic weighted data replication strategy in data grids*, in Proceedings of the IEEE/ACS International Conference on Computer Systems and Applications, 2008, pp. 414–421.

[14] H. CHETTAOUI AND F. BEN CHARRADA, *A Decentralized Periodic Replication Strategy based on Knapsack Problem*, in Proceedings of the 13th ACM/IEEE International Conference on Grid Computing, 2012, pp. 3–11.

[15] U. CIBEJ, B. SLIVNIK, AND B. ROBIC, *The complexity of static data replication in data grids*, Parallel Computing, 31 (2005), pp. 900–912.

[16] T.H. CORMEN, C.E. LEICERSON, R.L. RIVEST, AND C. STEIN, *Introduction to Algorithms*, Third Edition, MIT Press, 2009.

[17] Z. CUI, D. ZUO, AND Z. ZHANG, *Based on support and confidence dynamic replication algorithm in multi-tier data grids*, Journal of Computational Information Systems, 9 (2013), pp. 3909–3918.

[18] N. DANG, S. HWANG, AND S. LIM, *Improving job scheduling performance with dynamic replication strategy in data grids*, Lecture Notes in Computer Science, 4671 (2007), pp. 194–199.

[19] N. DANG AND S. LIM, *Combination of replication and scheduling in data grids*, International Journal of Computer Science and Network Security, 7 (2007), pp. 304–308.

[20] D. DULLMANN, W. HOSCHEK, J. JAEN-MARTINEZ, B. SEGAL, A. SAMAR, H. STOCKINGER, AND K. STOCKINGER, *Models for replica synchronisation and consistency in a data grid*, in Proceedings of IEEE International Symposium on High Performance Distributed Computing, 2001, pp. 67–75.

[21] H. E. AL MISTARIHI AND C. YONG, *Replica management in data grid*, International Journal of Computer Science and Network Security, 8 (2008), pp. 22–32.

[22] H. KELLERER, U. PFERSCHY, AND D. PISINGER, *Knapsack problems*, Springer, 2004.

[23] H. LAMEHAMEDI, Z. SHENTU, AND B. SZYMANSKI, *Simulation of dynamic data replication strategies in data grids*, in Proceedings of the Heterogeneous Computing Workshop, 2003.

[24] H. LAMEHAMEDI AND B. SZYMANSKI, *Decentralized data management framework for data grids*, Future Generation Computer Systems, 23 (2007), pp. 109–115.

[25] M. LEE, F. LEU, AND Y. CHEN, PFRF*: An adaptive data replication algorithm based on star-topology data grids*, Future Generation Computer Systems, 28 (2012), pp. 1045–1057.

[26] M. LEI AND S. VRBSKY, *A data replication strategy to increase data availability in data grids*, in Proceedings of the International Conference on Grid Computing and Applications, 2006, pp. 221–227.

[27] M. LEI, S. VRBSKY, AND X. HANG, *An on-line replication strategy to increase availability in data grids*, Future Generation Computer Systems, 24 (2008), pp. 85–98.

[28] N. MANSOURI AND GH. DASTGHAIBYFARD, *A dynamic replica management strategy in data grid*, Journal of Network and Computer Applications, 35 (2012), pp. 1297–1303.

[29] ———, *Improving data grids performance by using modified dynamic hierarchical replication strategy*, Iranian Journal of Electrical & Electronic Engineering, 10 (2014), pp. 27–37.

[30] S. MARTELLO AND P. TOTH, *Knapsack problems: Algorithms and computer implementations*, John Wiley & Sons, 1990.

[31] M. MEHRABAN, A. KHADEMZADEH, AND M. SALEHNAMADI, *A prediction-based replica replacement strategy in data grid*,

Journal of Basic and Applied Scientific Research, 2 (2013), pp. 928–939.
[32] J. Montagnat, V. Breton, and I. Magnin, *Using grid technologies to face medical image analysis challenges*, in Proceedings of the Workshop on Biomedical Computations on the Grid, 2003, pp. 588–593.
[33] S. Park, J. Kim, Y. Ko, and W. Yoon, *Dynamic data grid replication strategy based on internet hierarchy*, in Proceedings of the International Workshop on Grid and Cooperative Computing, 2003, pp. 838–846.
[34] R. Rahman, K. Barker, and R. Alhajj, *Replica placement in data grid: Considering utility and risk*, in Proceedings of the International Conference on Information Technology: Coding and Computing, 2005, pp. 354–359.
[35] ———, *Replica placement design with static optimality and dynamic maintainability*, in Proceedings of the IEEE International Symposium on Cluster Computing and Grid, 2006, pp. 434–437.
[36] ———, *Replica placement strategies in data grid*, Journal of Grid Computing, 6 (2008), pp. 103–123.
[37] K. Ranganathan and I. Foster, *Design and evaluation of dynamic replication strategies for a high-performance data grid*, in Proceedings of the International Conference on Computing in High Energy and Nuclear Physics, 2001.
[38] ———, *Identifying dynamic replication strategies for a high performance data grid*, in Proceedings of the Second International Workshop on Grid Computing, 2001, pp. 75–86.
[39] K. Ranganathan, A. Iamnitchi, and I. Foster, *Improving data availability through dynamic model-driven replication in large peer-to-peer communities*, in Proceedings of Global and Peer-to-Peer Computing in Large Scale Distributed Systems Workshop, 2002, pp. 376–381.
[40] Q. Rasool, J. Li, G. Oreku, and E. Munir, *Fair-share replication in data grid*, Information Technology Journal, 7 (2008), pp. 776–782.
[41] M. Sakarovitch, *Optimisation Combinatoire*, Hermann Edition, 1984.
[42] K. Sashi and A. Thanamani, *Dynamic replication in a data grid using a modified* BHR *region based algorithm*, Future Generation Computer Systems, 27 (2011), pp. 202–210.
[43] Y. Shi, A. Shortridge, and J. Bartholic, *Grid computing for real-time distributed collaborative geoprocessing*, in Proceedings of the Symposium on Geospatial Theory, Processing and Applications, 2002.
[44] P. Suri and M. Singh, DR2*: A two-stage dynamic replication strategy for data grid*, International Journal of Recent Trends in Engineering, 2 (2009), pp. 201–203.
[45] H. Takeuchi, T. Kondo, Y. Koyama, and J. Nakajima, *Vlbi@home-vlbi correlator by GRID computing system*, in Proceedings of the International VLBI Service for Geodesy and Astrometry, 2004, pp. 200–204.
[46] P. Vashisht, R. Kumar and A. Sharma, *Efficient dynamic replication algorithm using agent for data grid*, The Scientific World Journal Volume (2014), 10 pages, http://dx.doi.org/10.1155/2014/767016
[47] G. von Laszewski, M. Su, J. Insley, I. Foster, J. Bresnahan, C. Kesselman, M. Thiebaux, M. Rivers, S. Wang, B. Tieman, and I. McNulty, *Real-time analysis, visualization, and steering of microtomography experiments at photon sources*, in Proceedings of the 9th SIAM Conference on Parallel Processing for Scientific Computing, 1999.
[48] W. Zhao, X. Xu, Z. Wang, Y. Zhang, and S. He, *Improve the performance of data grids by value-based replication strategy*, in Proceeding of the International Conference on Semantics, Knowledge and Grids, 2010, pp. 313–316.

Edited by: Dana Petcu
Received: Feb 9, 2014
Accepted: Mar 11, 2014

AIMS AND SCOPE

The area of scalable computing has matured and reached a point where new issues and trends require a professional forum. SCPE will provide this avenue by publishing original refereed papers that address the present as well as the future of parallel and distributed computing. The journal will focus on algorithm development, implementation and execution on real-world parallel architectures, and application of parallel and distributed computing to the solution of real-life problems. Of particular interest are:

Expressiveness:

- high level languages,
- object oriented techniques,
- compiler technology for parallel computing,
- implementation techniques and their efficiency.

System engineering:

- programming environments,
- debugging tools,
- software libraries.

Performance:

- performance measurement: metrics, evaluation, visualization,
- performance improvement: resource allocation and scheduling, I/O, network throughput.

Applications:

- database,
- control systems,
- embedded systems,
- fault tolerance,
- industrial and business,
- real-time,
- scientific computing,
- visualization.

Future:

- limitations of current approaches,
- engineering trends and their consequences,
- novel parallel architectures.

Taking into account the extremely rapid pace of changes in the field SCPE is committed to fast turnaround of papers and a short publication time of accepted papers.

INSTRUCTIONS FOR CONTRIBUTORS

Proposals of Special Issues should be submitted to the editor-in-chief.

The language of the journal is English. SCPE publishes three categories of papers: overview papers, research papers and short communications. Electronic submissions are preferred. Overview papers and short communications should be submitted to the editor-in-chief. Research papers should be submitted to the editor whose research interests match the subject of the paper most closely. The list of editors' research interests can be found at the journal WWW site (`http://www.scpe.org`). Each paper appropriate to the journal will be refereed by a minimum of two referees.

There is no a priori limit on the length of overview papers. Research papers should be limited to approximately 20 pages, while short communications should not exceed 5 pages. A 50–100 word abstract should be included.

Upon acceptance the authors will be asked to transfer copyright of the article to the publisher. The authors will be required to prepare the text in LaTeX 2_ε using the journal document class file (based on the SIAM's `siamltex.clo` document class, available at the journal WWW site). Figures must be prepared in encapsulated PostScript and appropriately incorporated into the text. The bibliography should be formatted using the SIAM convention. Detailed instructions for the Authors are available on the SCPE WWW site at `http://www.scpe.org`.

Contributions are accepted for review on the understanding that the same work has not been published and that it is not being considered for publication elsewhere. Technical reports can be submitted. Substantially revised versions of papers published in not easily accessible conference proceedings can also be submitted. The editor-in-chief should be notified at the time of submission and the author is responsible for obtaining the necessary copyright releases for all copyrighted material.