Disk Resident Extendible Arrays

Ekow Otoo and Doron Rotem

Lawrence Berkeley National Laboratory

Outline

1. The Extendible Array File and Structural Properties
2. Motivation and Applications
3. Disk Resident Extendible Arrays
4. Parallel Access of Disk Resident Dense Extendible Array
5. Implementation Status
6. Managing Sparse Array Files
7. Comparison with Alternatives and Complexity
8. The New Desired Features and Future Work

Outline

1. The Extendible Array File and Structural Properties
2. Motivation and Applications
3. Disk Resident Extendible Arrays
4. Parallel Access of Disk Resident Dense Extendible Array
5. Implementation Status
6. Managing Sparse Array Files
7. Comparison with Alternatives and Complexity
8. The New Desired Features and Future Work

Outline

1. The Extendible Array File and Structural Properties
2. Motivation and Applications
3. Disk Resident Extendible Arrays
4. Parallel Access of Disk Resident Dense Extendible Array
5. Implementation Status
6. Managing Sparse Array Files
7. Comparison with Alternatives and Complexity
8. The New Desired Features and Future Work

Outline

1. The Extendible Array File and Structural Properties
2. Motivation and Applications
3. Disk Resident Extendible Arrays
4. Parallel Access of Disk Resident Dense Extendible Array
5. Implementation Status
6. Managing Sparse Array Files
7. Comparison with Alternatives and Complexity
8. The New Desired Features and Future Work

Outline

1. The Extendible Array File and Structural Properties
2. Motivation and Applications
3. Disk Resident Extendible Arrays
4. Parallel Access of Disk Resident Dense Extendible Array
5. Implementation Status
6. Managing Sparse Array Files
7. Comparison with Alternatives and Complexity
8. The New Desired Features and Future Work

Outline

1. The Extendible Array File and Structural Properties
2. Motivation and Applications
3. Disk Resident Extendible Arrays
4. Parallel Access of Disk Resident Dense Extendible Array
5. Implementation Status
6. Managing Sparse Array Files
7. Comparison with Alternatives and Complexity
8. The New Desired Features and Future Work

Outline

1. The Extendible Array File and Structural Properties
2. Motivation and Applications
3. Disk Resident Extendible Arrays
4. Parallel Access of Disk Resident Dense Extendible Array
5. Implementation Status
6. Managing Sparse Array Files
7. Comparison with Alternatives and Complexity
8. The New Desired Features and Future Work

Outline

1. The Extendible Array File and Structural Properties
2. Motivation and Applications
3. Disk Resident Extendible Arrays
4. Parallel Access of Disk Resident Dense Extendible Array
5. Implementation Status
6. Managing Sparse Array Files
7. Comparison with Alternatives and Complexity
8. The New Desired Features and Future Work

Main Characteristics

- The basic structure is a multidimensional array stored under a simple Unix file system or in a parallel file system
- any dimension is allowed to extend arbitrary
- parallel applications read/write/manipulate entire array or sub-arrays
- array can be extended without reorganizing previously allocated elements,
- define a mapping function and its inverse for element access.
- basic data types are: integers, floats, double and complex types.
- Feature extension, addressed later, to meet the requirements of ArrayDB storage system — Documents of the SciDB Meetings
- In particular, extensions should be along extents of the dimensions, number of dimensions, and resolution of the array cells where applicable.

Main Characteristics

- The basic structure is a multidimensional array stored under a simple Unix file system or in a parallel file system
- any dimension is allowed to extend arbitrary
- parallel applications read/write/manipulate entire array or sub-arrays
- array can be extended without reorganizing previously allocated elements,
- define a mapping function and its inverse for element access.
- basic data types are: integers, floats, double and complex types.
- Feature extension, addressed later, to meet the requirements of ArrayDB storage system — Documents of the SciDB Meetings
- In particular, extensions should be along extents of the dimensions, number of dimensions, and resolution of the array cells where applicable.

Main Characteristics

- The basic structure is a multidimensional array stored under a simple Unix file system or in a parallel file system
- any dimension is allowed to extend arbitrary
- parallel applications read/write/manipulate entire array or sub-arrays
- array can be extended without reorganizing previously allocated elements,
- define a mapping function and its inverse for element access.
- basic data types are: integers, floats, double and complex types.

- Feature extension, addressed later, to meet the requirements of ArrayDB storage system — Documents of the SciDB Meetings
- In particular, extensions should be along extents of the dimensions, number of dimensions, and resolution of the array cells where applicable.

Motivation and Applications

- The earliest application known for the need of extendible arrays is in telecommunication network analysis — Arnold Rosenberg and Stockmeyer.
- Scientific data analysis use multidimensional arrays as their fundamental data structures. Examples of **Array Files**:
  - HDF/HDF5 and variants
  - NetCDF/pNetCDF
  - FITS
  - Quaternary Triangular Mesh (QTM), Hierarchical Triangular Mesh (HTM), etc.
  - Global Array toolkit
  - Quad-tree, Linear Quad-codes and in general structures that are based on spatial mappings of space filling curves.
- Application in data warehousing

Motivation and Applications

- The earliest application known for the need of extendible arrays is in telecommunication network analysis — Arnold Rosenberg and Stockmeyer.
- Scientific data analysis use multidimensional arrays as their fundamental data structures. Examples of **Array Files**:
    - HDF/HDF5 and variants
    - NetCDF/pNetCDF
    - FITS
    - Quaternary Triangular Mesh (QTM), Hierarchical Triangular Mesh (HTM), etc.
    - Global Array toolkit
    - Quad-tree, Linear Quad-codes and in general structures that are based on spatial mappings of space filling curves.
- Application in data warehousing

Motivation and Applications, Cont.

- **ArrayDB** (Most recent proposal from SciDB meeting):
  - allow extent of each dimension to expand
  - allow the number of dimensions to expand
  - time is implicit i.e., insertion is automatically associated with time and deletion is only logical.
  - The ArrayDB can be versioned at any instant in time.
  - to be implemented with combined of features of HDF/HDF5, R-Trees, HTM, Map-Reduce, Vertica, Postgress

Illustration of a Dense Extendible Array

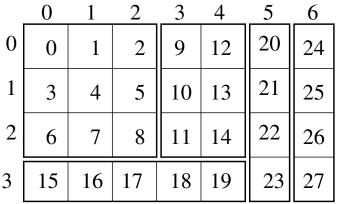
- A 2-D array initially defined as $A[3][3]$ and then extended by 2 columns, then by 1 row, followed by 1 column and so on.

	0	1	2	3	4	5	6
0	0	1	2	9	12	20	24
1	3	4	5	10	13	21	25
2	6	7	8	11	14	22	26
3	15	16	17	18	19	23	27

- The labels in the cells are location addresses of the elements.
- An element $A\langle 2, 5\rangle$ maps to location 22
- The address calculation is done by a function denoted as: $\mathcal{F}_*(i_0, i_1, \ldots, i_{k-1}) \rightarrow I$
  and an inverse $\mathcal{F}_*^{-1}(I) \rightarrow \langle i_0, i_1, \ldots, i_{k-1}\rangle$

Illustration of a Dense Extendible Array

- A 2-D array initially defined as $A[3][3]$ and then extended by 2 columns, then by 1 row, followed by 1 column and so on.



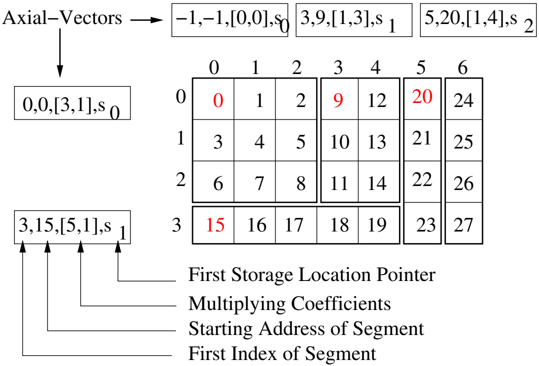
	0	1	2	3	4	5	6
0	0	1	2	9	12	20	24
1	3	4	5	10	13	21	25
2	6	7	8	11	14	22	26
3	15	16	17	18	19	23	27

- The labels in the cells are location addresses of the elements.

- An element $A\langle 2, 5\rangle$ maps to location 22

- The address calculation is done by a function denoted as:
  $\mathcal{F}_*(i_0, i_1, \ldots, i_{k-1}) \rightarrow I$
  and an inverse $\mathcal{F}_*^{-1}(I) \rightarrow \langle i_0, i_1, \ldots, i_{k-1}\rangle$

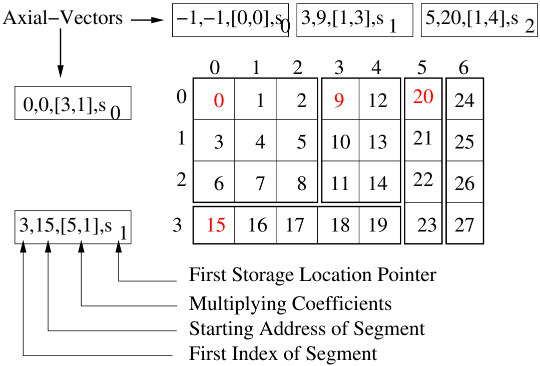
Linear Mapping for a Dense Extendible Array



	0	1	2	3	4	5	6
0	0	1	2	9	12	20	24
1	3	4	5	10	13	21	25
2	6	7	8	11	14	22	26
3	15	16	17	18	19	23	27

- The element $A\langle 2, 5\rangle$ is located in either segment of row 2 with start address 0 or segment of column 5 with start address 20.
- It is always allocated in segment with maximum starting address.
- The address of $A\langle 2, 5\rangle$ is computed by the algorithm $\mathcal{F}_*()$.

Linear Mapping for a Dense Extendible Array



Axial–Vectors → −1,−1,[0,0],s_0 | 3,9,[1,3],s_1 | 5,20,[1,4],s_2

0,0,[3,1],s_0

3,15,[5,1],s_1

	0	1	2	3	4	5	6
0	0	1	2	9	12	20	24
1	3	4	5	10	13	21	25
2	6	7	8	11	14	22	26
3	15	16	17	18	19	23	27

First Storage Location Pointer

Multiplying Coefficients

Starting Address of Segment

First Index of Segment

- The element $A\langle 2,5\rangle$ is located in either segment of row 2 with start address 0 or segment of column 5 with start address 20.
- It is always allocated in segment with maximum starting address.
- The address of $A\langle 2,5\rangle$ is computed by the algorithm $\mathcal{F}_*()$.

Disk Resident Extendible Arrays

- The elements are first grouped into *chunks* of some predefined *Chunk-Shape*, $A[I_0][I_1] \ldots A[I_{k-1}]$
- The chunks form the units of transfer between memory and a parallel file system.
- The mapping functions discussed are now applied to address the chunks and the array elements within a chunk can now be accessed using conventional array element address calculation.
- The *Axial-Vectors* are retained in a Meta-Data file but read into memory at each session.
- Additional information in the Meta-Data include the bounds of the array, the chunk-shapes, etc.

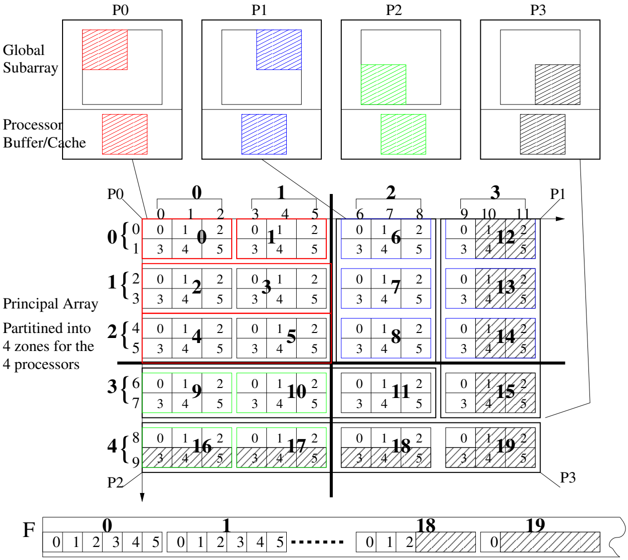
Disk Resident Extendible Arrays

- The elements are first grouped into *chunks* of some predefined *Chunk-Shape*, $A[l_0][l_1]\ldots A[l_{k-1}]$
- The chunks form the units of transfer between memory and a parallel file system.
- The mapping functions discussed are now applied to address the chunks and the array elements within a chunk can now be accessed using conventional array element address calculation.
- The *Axial-Vectors* are retained in a Meta-Data file but read into memory at each session.
- Additional information in the Meta-Data include the bounds of the array, the chunk-shapes, etc.

Disk Resident Extendible Arrays

- The elements are first grouped into *chunks* of some predefined *Chunk-Shape*, $A[l_0][l_1] \ldots A[l_{k-1}]$
- The chunks form the units of transfer between memory and a parallel file system.
- The mapping functions discussed are now applied to address the chunks and the array elements within a chunk can now be accessed using conventional array element address calculation.
- The *Axial-Vectors* are retained in a Meta-Data file but read into memory at each session.
- Additional information in the Meta-Data include the bounds of the array, the chunk-shapes, etc.

The Allocation Scheme



Layout of Array Chunks in a File

Accessing Dense Extendible Arrays (The pDRXA Library)

- Array is distributed by Block, Block partitioning scheme and along chunk boundaries. Block-Cyclic partitioning not yet
- A process controls a region of sub-array called a *zone* and an application can sub-arrays with either independent or collective I/O.
- Each process then makes its zone accessible by creating a memory window for RMA access.
- Since each process has all the distribution information, it can access an element locally, if it controls the zone of the element; otherwise it accesses the element remotely via functions like *MPI_Get()*, *MPI_Put()* and *MPI_Accumulate(), etc.*
- The processing model is consistent with the Global-Array toolkit model for parallel processing of arrays.
- The idea then is to define the access functions to be consistent with the Disk Resident Array library of GA and leverage the scientific processing capability of GA.

Accessing Dense Extendible Arrays (The pDRXA Library)

- Array is distributed by Block, Block partitioning scheme and along chunk boundaries. Block-Cyclic partitioning not yet
- A process controls a region of sub-array called a *zone* and an application can sub-arrays with either independent or collective I/O.
- Each process then makes its zone accessible by creating a memory window for RMA access.
- Since each process has all the distribution information, it can access an element locally, if it controls the zone of the element; otherwise it accesses the element remotely via functions like *MPI_Get()*, *MPI_Put()* and *MPI_Accumulate(), etc.*
- The processing model is consistent with the Global-Array toolkit model for parallel processing of arrays.
- The idea then is to define the access functions to be consistent with the Disk Resident Array library of GA and leverage the scientific processing capability of GA.

Accessing Dense Extendible Arrays (The pDRXA Library)

- Array is distributed by Block, Block partitioning scheme and along chunk boundaries. Block-Cyclic partitioning not yet
- A process controls a region of sub-array called a *zone* and an application can sub-arrays with either independent or collective I/O.
- Each process then makes its zone accessible by creating a memory window for RMA access.
- Since each process has all the distribution information, it can access an element locally, if it controls the zone of the element; otherwise it accesses the element remotely via functions like *MPI_Get()*, *MPI_Put()* and *MPI_Accumulate(), etc.*
- The processing model is consistent with the Global-Array toolkit model for parallel processing of arrays.
- The idea then is to define the access functions to be consistent with the Disk Resident Array library of GA and leverage the scientific processing capability of GA.

Current Status

- The dense extendible array is completed and the parallel counterpart is usable much like global array with MPI-2 but does not do true out-of-core array operations.
- The interface implementation to allow it to be used as a replacement for DRA of GA is ongoing.
- Two challenges posed for its use in managing HDF5 chunks:
  - How does its performance compare with skiplist indexed array chunks?
  - How does it manage sparse array chunks?
  - How does it manage multi-resolution arrays?
  - Can it be used for multi-threaded applications?

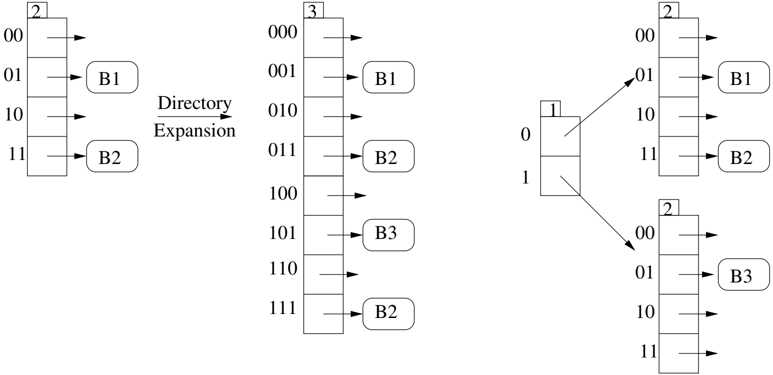
Current Status

- The dense extendible array is completed and the parallel counterpart is usable much like global array with MPI-2 but does not do true out-of-core array operations.
- The interface implementation to allow it to be used as a replacement for DRA of GA is ongoing.
- Two challenges posed for its use in managing HDF5 chunks:
  - How does its performance compare with skiplist indexed array chunks?
  - How does it manage sparse array chunks?
  - How does it manage multi-resolution arrays?
  - Can it be used for multi-threaded applications?

Managing Sparse Arrays

- The general technique is by chunking, compression, and indirect addressing
- HDF5 uses B^+-Tree.
- The new research direction is with the use of **Skip-List** in place of B^+-Tree.
- Our approach for managing and indexing array chunks is with Balanced Extendible Hashing instead of **Skip-List**.
- The basic BEH is completed and to avoid future questions on its use, it is being modified to be thread safe.
- We are implementing the Skip-List ideas for performance comparisons.

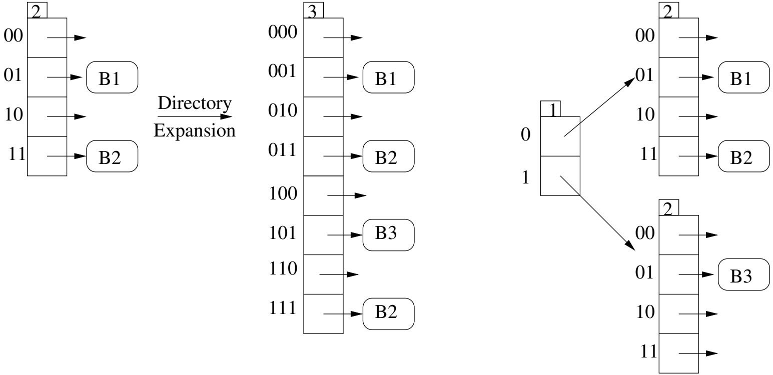
Idea of the Balanced Extendible Hashing Scheme



Bal. Ext. Hashing

- Instead of allowing the directory to double we impose a bound on the number of bits used to address into it to say l bits.
- For a bitstring of 64, setting l to 12 gives at most 4 or 5 levels.
- Caching further reduces number of disk accesses.

Idea of the Balanced Extendible Hashing Scheme



Bal. Ext. Hashing

- Instead of allowing the directory to double we impose a bound on the number of bits used to address into it to say *l* bits.
- For a bitstring of 64, setting *l* to 12 gives at most 4 or 5 levels.
- Caching further reduces number of disk accesses.

Comparison with Alternatives and Complexity

	Index Method	**Complexity of Chunk Address**	**No Accesses**	**Extendibility**
Dense	B^+-Tree (+ sparse Arrays)	$O(B \log_{\lceil B/2 \rceil} n_p)$	$O(\log_{\lceil B/2 \rceil} n_p)$	Any Dimen.
	Extendible Array	$O(k \log \log n_p)$	1	Any Dimen.
	Skip-List (+ sparse Arrays)	$O(\log n_p)$	$O(\log n_p)$	One Dimen.
Sparse	Bal. Ext. Hash.	$O(1)$	$l \leq 4$	Any Dimen.

The New Desired Features, Why and Future Work

- The why is simply — provide the storage system desired by the ArrayDB model.
- Current work already meets some of the storage requirement of ArrayDB:
  - usage in a cluster environment and parallel file systems
  - has extendible extents and very easy to extend number of dimensions
  - can leverage the capability of the Global Array tookit for array and matrix operations in memory.

The New Desired Features, Why and Future Work

- The why is simply — provide the storage system desired by the ArrayDB model.
- Current work already meets some of the storage requirement of ArrayDB:
    - usage in a cluster environment and parallel file systems
    - has extendible extents and very easy to extend number of dimensions
    - can leverage the capability of the Global Array tookit for array and matrix operations in memory.

The New Desired Features and Future Work, Cont.

- Future requirements to be met include:
  - Make elements of the arrays to be tuples, i.e., base elements - floats, integers

    + elements that are multidimensional arrays with the limitation that the valued-attributes can only be basic types.
  - Usage in multiple clusters as a Peer-to-Peer distributed file for K-fault tolerance
  - allow multi-resolution array with regular and irregular extents.
  - allow time as an automatic variable
  - ensure no updates in places:

    Old values are time stamped with deletion time and

    allow the array data to be versionable
  - Write Meta-data file in XML and maintain it in the array data file.
  - leverage the capability of the Global Array toolkit but with out-of-core array and matrix operations.