This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2541142, IEEE Transactions on Cloud Computing

Segmented In-advance Data Analytics for Fast Scientific Discovery

Jialin Liu
Department of Computer Science
Texas Tech University
Lubbock, Texas, USA
Email: jaln.liu@ttu.edu

Yong Chen
Department of Computer Science
Texas Tech University
Lubbock, Texas, USA
Email: yong.chen@ttu.edu

***Abstract*—Scientific discovery usually involves data generation, data preprocessing, data storage and data analysis. As the data volume exceeds a few terabytes (TB) in a single simulation run, the data movement, which happens during each cycle of the scientific discovery, continues to be the bottleneck in most scientific big data applications. A lot of research works have been conducted on reducing the data movement. Among the existing efforts and based on our previous research, reusing the analysis results shows a significant potential in optimizing the data movement between analysis operations. In this work, we propose the *Segmented In-Advance* (SIA) data analytics approach for optimizing the data movement and we also provide a cloud-based elastic distributed in-memory database to manage the intermediate analysis results. The fundamental idea of this *Segmented In-Advance* approach is to analyze the history operations and to predict the future interesting analytics operations. The predicted analysis operation is in-advance performed on the finer segmented dataset and the segmented results are distributed in an in-memory key-value store for future reuse. The evaluation shows that the segmented in-advance data analytics approach achieves 1.2X-6.1X speedup. The evaluation also shows a good scalability of the in-memory distributed data store. The proposed *Segmented In-Advance* data analytics approach is a promising data movement reduction solution for scientific big data applications and fast scientific discovery.**

***Keywords*-segmented in-advance data analytics, big data, data intensive computing, scientific computing**

I. INTRODUCTION

Scientific discovery face challenges as the data volume keeps increasing. As the computing power tends to increase much more faster than the storage capability, the poor I/O performance continues to be a critical bottleneck. The main cause of the I/O bottleneck is slow disk-read speeds compared to both the CPU speed and memory bandwidth. The data collected from scientific instruments, simulations, and applications keep increasing I/O access cost which eventually dominate the overall scientific discovery time. We use the table I to show some sample data requirements of representative scientific applications run at Argonne Leadership Computing Facility (ALCF) through the DOE's INCITE program[31]. We can see from the table that the data volume processed online by many applications has exceeded TBs or even tens of TBs; the off-line data is near PBs of scale.

Reducing data movement is an important technique to address the I/O bottleneck issue. Recent studies including in-situ/in-transit data processing [36], [9], [17] (processes the data simultaneously at the time the data are generated), data compression [21], [11], [16] (compresses the data before written to storage), and active storage [29], [26], [35] (moves the computation to storage and performs the analysis directly in place where the data are located) are all along this direction. Among those techniques, the in-situ/in-transit processing usually desires enough prior knowledge of how scientists conduct the analysis, therefore it is sometimes hard for applications that the scientists prefer to perform iterative and diverse post-analysis. The active storage attempts reducing data movement during analysis phase with an assumption that the storage device is able to deliver the computing power that the processing requires. However, many scientific computations desire more computational power than what storage devices can offer, especially when working on big data problem. Data compression reduces the data movement by requiring an additional step to compress and decompress the data.

Both of our previous works (segmented analysis [22], in-advance data analysis [23]) and the Resilient Distributed Datasets (RDD) design in the Spark data processing framework [37], demonstrate that caching the intermediate analysis results in memory can further reduce data movement dramatically. The core idea is to re-use the cached results to reconstruct the future analysis and reduce the future data

TABLE I: Data Requirements of Representative INCITE Applications at ALCF [31]

Project	On-Line Data	Off-Line Data
FLASH: Buoyancy-Driven Turbulent Nuclear Burning	75TB	300TB
Reactor Core Hydrodynamics	2TB	5TB
Computational Nuclear Structure	4TB	40TB
Computational Protein Structure	1TB	2TB
Performance Evaluation and Analysis	1TB	1TB
Climate Science	10TB	345TB
Parkinson's Disease	2.5TB	50TB
Plasma Microturbulence	2TB	10TB
Lattice QCD	1TB	44TB
Thermal Striping in Sodium Cooled Reactors	4TB	8TB

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2541142, IEEE Transactions on Cloud Computing

movement.

In this paper, we propose the 'Segmented In-advance' data analytics approach with a distributed data store to manage the analysis results and predict the future analysis results. The fundamental idea is to analyze the existing analytics logs to extract the analytics pattern. When there is a new analytics operation, the future operation is predicted according to the pattern and the data are segmented and processed to generate both current sub-results and future sub-results. The results generated from such segmented in-advance analytics, together with the results from demanded analyses, are all cached in an elastic distributed in-memory store for fast retrieval and reuse. When the future task hits the in-memory results, those part of data are not needed to read and recompute again. The result is directly re-used, and the data movement can be substantially reduced.

The rest of the paper is organized as follows. Section II discusses a motivating example and presents two different scenarios. Section III formalizes the properties and rules of the proposed approach, as well as the challenges. Section IV presents the pattern definition and prediction of future analysis operation. Section V shows the system design and related algorithms. Section VI briefly discusses the implementation. We present performance evaluation results in Section VII. Section VIII discusses related work and compares them with our proposed strategy. We concludes the discussion and briefly discusses future work in Section IX.

II. Motivation

In climate science community, data are either collected from instruments, e.g., satellite, or generated from simulations with climate models, e.g., general circulation model(GCM [3]), Global Coupled Climate Models(CM2 [12]) and Community Climate System Model(CCSM [2]. The dataset usually covers a 3-dimensional grid with multiple parameters, e.g., temperature, pressure, wind velocity, etc. Scientists query and access different subsets of the data to perform the analysis, which involves large amount of data movement. In this section, through a motivation example, we show how our segmented in-advance analysis technique can reduce the data movement. Most previous efforts reduce the data movement with a focus on the raw data, e.g., compressing the raw data and processing the raw data *in situ*. Our work proposes to reduce the data movement by reusing the results (not raw data).

In Figure 1, two operations that calculate the 'mean' value are performed on two different subsets. As long as there is data overlap among the two operations, we can cache and reuse the results from the previous operation. These two operations perform the same computation contiguously, e.g,. calculating the mean. The operation op_1 is a 'mean' computation, which covers 16 segments (the segment refers to a fixed size subset). The op_2 shows a 'mean' operation after op_1. To complete op_2, all touched segments are needed to be read from disk, which is 18 segments totally. However, the overlapping parts between these two operations is as large as 12 segments. These segments are essentially not needed to be accessed again if

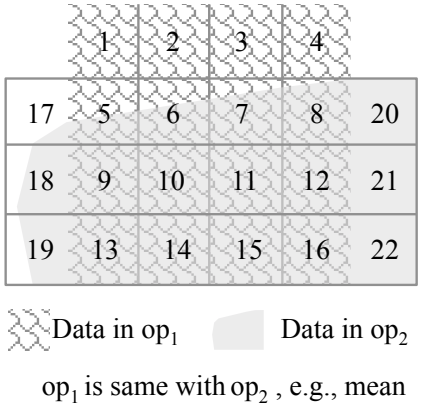

Fig. 1: Segmented Reusing Previous Analysis Result

their results are known. If there is a way to detect and reuse the results of the overlapping part, the data movement could be significantly reduced. Compared to traditional cache and data reuse, this result caching idea has less requirement for the cache size(with an assumption that the size of result is usually far less than the raw data). Furthermore, in Figure 1, there are four types of segments, i.e.,

1) Segments accessed by op_1 only, i.e., (1-4)
2) Segments accessed by op_2 only, i.e., (17-19, 20-22)
3) Segments accessed partially by op_1 and op_2, i.e., (5-8)
4) Segments accessed fully by op_1 and op_2, i.e., (9-16)

The op_2 needs to access the (2,3,4) types of segments. Instead of accessing these types of segments, however, type 4 can be skipped, since it is already computed in earlier task, i.e., op_1. Therefore, op_2 only needs ten segments(5-8,17-19,20-22) instead of eighteen. Without considering other overhead, the performance speedup is $18/10 = 1.8$X. The I/O can be further reduced by partitioning the segment into finer segments. For example, during the analysis of op_1, we can partition the data into finer segments. and compute the sub-results. This can further increase the reusability of the existing results. Different finer segmentation strategies have been explored, one of which is the dimension-driven segmentation [22]. In this paper, we only focus on the concept and idea of the in-advance segmented analysis.

Finer segments can generate more additional results for future analysis, but reusing the results still has limit, i.e., the operations must be the same. As shown in Figure 1, the two operations both compute the mean value, therefore, the mean value of the overlapping part can be re-used. If the computation of the two operations are different, the overlap can not bring benefit(unless we cache the raw data). We try to break such limit by looking at a longer period of analysis. We argue that the operations do not have to be the same.

Figure 2 shows a typical workflow that the computation is different among operations, in which different operations are issued to access different subsets of the data. In operation 1, when the subsets (where the arrow points to) are ready in the compute nodes, we usually delete the data when operation 1 is

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2541142, IEEE Transactions on Cloud Computing

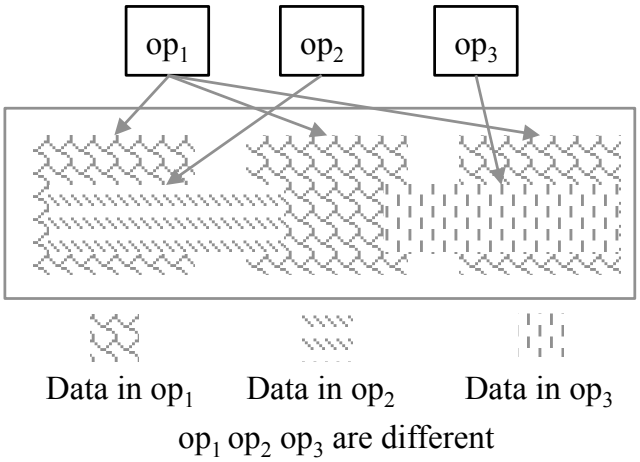

Fig. 2: In-advance Computing Future Analysis Results

done. But we can find the later operation, e.g., operation 3 will also need the operation 1's data as part of its I/O. Therefore, if we perform the operation 3 in-advance at the time of operation 1, such that the data movement in operation 3 is also reduced.

One real analysis script to calculate the monthly number of clear and cloudy days [20] is shown in (Listing 1). The analysis requires user to conduct a sequence of operations. After each operation, there is an output file for storing the intermediate results. The later operations in this analysis are based on the former operation and its results. As shown in the script, the first operation is to calculate the interpolation weights (operator 'genbil'). These interpolation weights (in 'we.nc') are then reused for every remapping process with the operator 'remap'. In the final operation, all intermediate values are added up using the ensemble sum operator 'enssum'. This example is a typical example of the scientific data analysis, where the order of operations forms a fixed sequence to complete the analysis. The operation sequence can be represented as $seq = \{op1, op2, op3\} = \{genbil, remap, enssum\}$. For exploring unknown knowledge from the huge data, these kind of sequences are repeated over different data regions constantly. If we can trace the analysis pattern, we may be able to pro-actively compute the required result in advance and reduce the future data movement.

```
Operation 1: Compute Weights
    cdo -f nc -genbil, grid.txt -import_cmsaf
        CFCdm${day}.hdf we.nc
Operation 2: Remap and Selection
    while [$day -le 20100831]
    do
    cdo -f nc -gec, 75
        -remap,grid.txt, we.nc -import_cmsaf
            CFCdm${day}.hdf
         CFCdm${day}.nc
   day= 'expr $day + 1'
   done
Operation 3: Add Values in All Files
   cdo enssum CFCdm201008??.nc cl.nc
```

Listing 1: Operation Sequence for Monthly Number of Clear/Cloudy Days

III. Problem Statement, Challenges and Assumption

From the previous discussion and the motivation examples, we have seen the potential of the segmented in-advance approach. The fundamental idea is to utilize the analysis results, either reusing previous analysis result, or predicting future analysis results. Such approach trades off the computation for better data movement. As a promising approach in scientific big data computing, there are challenges and we need to formalize the problem and make the assumption clear. First of all, we have five challenges:

1) What kind of applications can use this technique?
2) What kind of result can be re-used?
3) How to design the segmentation?
4) Where to store the analysis results?
5) How to do in-advance prediction?

We define the analysis result as 'data, plus the operation performed on the data' 1. In other words, the result is another form of data, but enhanced with computation efforts.

$$Result = (data, operation) \tag{1}$$

When the existing result, e.g., R_i, is to be re-used in the new analysis, e.g., R_x, R_i must first have the properties that,

$$\begin{aligned} R_i(data) &\subset R_x(data) \\ R_i(operation) &= R_x(operation) \end{aligned} \tag{2}$$

For example, $R_0 = (a[0, 1, 2, ..., 10], mean)$ and $R_x = (a[0, 1, 2, ..., 20], mean)$, then R_0 can be re-used in R_x. Secondly, the operations should be 'additive' or 'sub-optimal'. For additive operation, e.g., sum, the final result can be constructed from small components. Additive operation only covers a small range of statistics. Sub-optimal operation has a broader range than additive operation, The sub-optimal operation is defined as 'the previous steps generate the desired input of the later steps'. For example, in analyzing the dataset generated from large eddy simulation, different computation shares the common analysis steps [19].

We can decide whether the application can benefit from our approach based on the properties and rules. However, other challenges are still fundamental issues to the Segmented In-Advance data analytics approach. To address them, we design the finer segmentation algorithm to flexibly partition the dataset, and we also propose to use the pattern recognition technique to parse the analysis logs and to predict the future analysis operation. We propose to use the distributed in-memory data store to manage all the intermediate analysis results.

IV. Pattern Definition and Detection

As we have shown in the motivation section, the users' analytic activity usually reveals a certain pattern. The Segmented In-Advance framework recommends a potential analysis based on such pattern. The pattern itself, however, varies from application to application, and differs among users. We first define the common patterns based on different scenarios. We

also design the algorithms to detect the patterns and discuss how the pattern can be used to guide the pre-analysis for reducing data movement.

By analyzing different data analytics in many domain sciences [20], [34], we observe two general patterns that can be defined as a *memorable pattern* and a *memoryless pattern*. For the memorable pattern, which are stateful pattern, the users' analytic task is consisted of an ordered operation sequence. A latter operation in this sequence can not proceed without the intermediate results of the former operation in this sequence. The second general pattern, memoryless pattern, which are stateless, represents that future analytics only depend on the current operation, and are not related to the past operations. We discuss in details in the following paragraphs.

A. Memorable Pattern

Our system uses an in-memory database to record each analytic operation (please see detailed discussion in subsection V-C), but the challenge for detecting such memorable pattern, or memorable sequence, is how to find the 'meaningful sequence' (which is defined as 'a serial of operations can be used in fixed order to complete one analysis task'). As we have observed in the climate science example, that the inputs of each operation within one sequence are not arbitrary. The latter operation's input is based on the former operation's output, therefore, the overlapping data is the key to detect the meaningful sequence.

We use a directed graph to represent the sequence and describe the data overlap. In the graph, the nodes represent the operations. Each operation is a node, and nodes are connected when the data accessed in operations has an overlap. Different nodes can represent same operation, because they are conducted on different data or different time. The sequence in Figure 2 is now represented in the following graph (Figure 3, left subfigure).

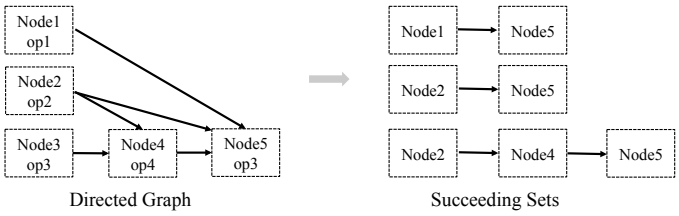

Fig. 3: Succeeding Set Derived from Directed Graph

To build the directed graph from users' historical analytics, we follow two rules, i.e.,

1) Operations are in order
2) Operations access overlapped data sets

New nodes are added in and connected with existing nodes by following these two rules. For example, in the directed graph of Figure 3, we start with the first operation (e.g., op1: monadd) as node 1. The next operation is op2, e.g., 'ymonmul', does not have overlap I/O with the first one, then we make it as another starting root node. Same with Node 3. As 'op4' accessed the data that has overlap with both 'op2' and 'op3', so we creat the new node 'Node4' and connect it with 'Node2' and 'Node3'. The last operation in our example is 'op3' which has occured previously. And since it has overlap with the 'op1,2,4', we connected it with all the corresponding nodes. Overall, for any new operation, we check from the closest operation, until we find there is no data overlap with any previous nodes, then we start a new root node and continue this process. If there is overlap, we connect them.

After building the directed graph from the history analytics, we use depth first search algorithm to derive each sequence, which can be represented as a succeeding set. The operation's succeeding set tells what are the operations that happen after this operation and has overlap on I/O. For example,

$$Succeeding(monadd) = \{ymonmul, max\}$$
$$Succeeding(mean) = \{min, ydayadd, yhourmul, enssum\}$$
$$Succeeding(mean) = \{max, ensstd, enskhistspace\} \quad (3)$$

in which, Succeeding(X)={operations come after X}. The operation 'ymonmul' means 'Multiply multi-year monthly time averages'. Others can be found in the CDO website [1]; The succeeding set implies what sequence of operations will soon occur on the same or overlapped data when certain analytic task occurs; for example, ymonmul and max appear after monadd. Therefore, we can in advance conduct the analytic tasks in the succeeding set before the data are deleted or files are written back to storage nodes. In Section V-A, we will discuss how to detect the sequence pattern and conduct in-advance analytics.

B. Memoryless Pattern

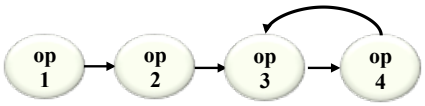

Fig. 4: Markov Chains of Analytic Operations

The previous memorable pattern is commonly observable in users' analytic pattern. In practice, many cases also exist, in which no pre-defined sequence is followed. For example, scientists may neither know much about what is inside the data, nor what analytic can quickly lead to the interesting discovery [28]. Therefore, the analytic activity tends to be random and spanning over the dataset. As the users' analytic activity also tends to be memoryless, random and the future operations only depends on the current analytic operation (no relation with the past operations). Such pattern has the same characteristics with the Markov Chains, thus we represent the pattern using *Markov Chains*. For example, we can represent the previous example in Figure 4. We consider the users' memoryless analytic activity as a stochastic process,

$$\{X^{(n)}, n = 0, 1, 2, ...\} \quad (4)$$

in which, the $X^{(n)}$ can be any operation, e.g., 'cdo mean'.

Suppose the analytic process is a Markov Chain process,we can have

$$P(X^{(n+1)} = i | X^{(n)} = i_{n-1}, ..., X^{(0)} = i_0) = P_{i,j} \quad (5)$$

in which, $P_{i,j}$ represents the probability that the user's next operation will be i given the current analytic operation j. $P_{i,j}$ is independent of time such that future operation only depends on the current operation. Clearly one has

$$P_{i,j} \geq 0, \sum_{i=0}^{\infty} P_{i,j} = 1, j = 0, 1, ... \quad (6)$$

Therefore, we can build a matrix to represent all the operations and its transition probability.

$$P = \begin{pmatrix} P_{0,0} & P_{0,1} & \cdots & P_{0,m} \\ P_{1,0} & P_{1,1} & \cdots & P_{1,m} \\ \vdots & \vdots & \vdots & \vdots \\ P_{m,0} & P_{m,1} & \cdots & P_{m,m} \end{pmatrix} \quad (7)$$

where each element represents the probability of event that operation j comes after operation i.

The above represents one-step transition probability matrix P for a Markov Chain. Using this one-step transition matrix, we can recommend only one future step of analytic operation. For an n-step transition probability, we can derive the P using Equation 8:

$$P_{ij}^{(n)} = P_{ij}^{n} \quad (8)$$

Given an observed analytic sequence $SEQ = \{X\}$, e.g., $seq = \{min, ydayadd, yhourmul, enssum, min, ydayadd, max, ensstd, enskhistspace\}$, we build the Markov chain in four steps. First, we compute the transition frequency F_{ij} by counting the number of transitions from operation i to operation j in one step, e.g., $F_{min,ydayadd} = 2$ and $F_{max,ensstd} = 1$, etc. Second, we construct one-step transition matrix for the sequence X as follows:

$$F = \begin{pmatrix} F_{0,0} & F_{0,1} & \cdots & F_{0,m} \\ F_{1,0} & F_{1,1} & \cdots & F_{1,m} \\ \vdots & \vdots & \vdots & \vdots \\ F_{m,1} & \cdots & \cdots & F_{m,m} \end{pmatrix} \quad (9)$$

Third, from F, we can get the estimates for $P_{i,j}$ in Equation 7. where

$$P_{i,j} = \begin{cases} \dfrac{F_{i,j}}{\sum_{i=1}^{m} F_{i,j}} & \text{if } \sum_{i=1}^{m} F_{i,j} > 0 \\ 0 & \text{if } \sum_{i=1}^{m} F_{i,j} = 0 \end{cases} \quad (10)$$

Fourth, the n-step transition matrix can be calculated using the same method with one-step matrix (Equation 7, 8, 9, 10), except that the frequency should be the number of transitions from operation i to operation j in n steps, e.g., $F_{min,yhourmul} = 1$ and $F_{min,ydayadd} = 0$ for $n = 2$.

V. System Design

We introduce the Segmented In-Advance Data Analytics system and its workflow in this section. As shown in the Figure 5, the system has three major components, i.e., a recommendation component, an in-advance analytics kernel component, and elastic distributed in-memory database. One of the core parts is its recommendation system, in which the statistical methods are applied to formalize and detect the analytics pattern. The kernel processes the current task and also performs the recommended analytics for future reuse. The current and recommended analytic results (and sub-results) are stored in the key-value database, which is used to speedup the result retrieval and facilitate the analysis. We discuss each component in the following subsections.

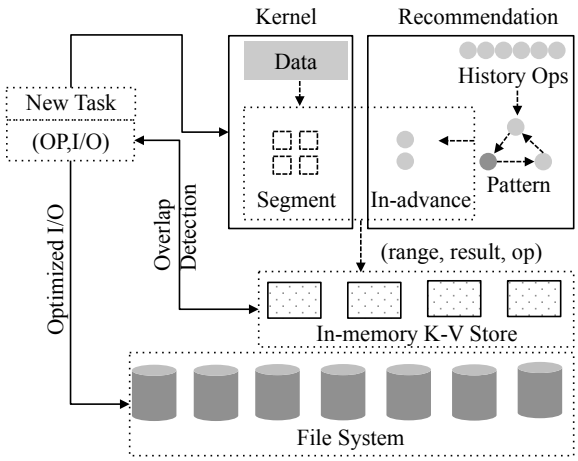

Fig. 5: Segmented In-Advance Data Analytics System

A. Pattern Detection and Analysis Recommendation

As we have described in the Section IV, the result of pattern detection includes at least the memoryless Markov pattern (since memorable sequence may not exist). Either only one pattern exists, or both patterns exist, i.e., memorable sequence pattern P_s and memoryless Markov pattern P_m, the Segmented In-Advance Data Analytics System is designed to choose the operations with memory constraints.

Initially, the system randomly selects one analytic operation from available operations (current test is based on CDO operators [1]), when there is no historical logs. After two iterations (two is the minimum number to construct both patterns), the system begins to construct each pattern, using the methods as discussed in Section IV. This pattern construction is conducted periodically, e.g., user can define a $period = 1$, which means pattern construction happens for 'every' new task, such that the existing pattern sequences or transition matrix are updated constantly. Since the recommended operations are performed on the finer-grained segments (the original accessed data is logically chunked into segments, please see Section V-B), the additional sub-results will consume increasing amount of memory; therefore, for each recommendation, we set a limited

size, ls. If the recommended operations will generate more analysis results than the allocated memory, we will not recommend it. In practice, memorable sequences are much more preferable than memoryless patterns, so the recommended operations are chosen from the memorable sequence ($p_s(x)$) first and memoryless pattern ($p_m(x)$) second within this memory limit. A pseudocode of this memory-constrained recommendation algorithm is shown in the Algorithm 29, in which, $nseq, nmar$ are the number of operations recommended from sequence pattern and Markov pattern separately; $size$ denotes the size of additional results, initially zero; $size(op, ds)$ is a function for estimating the result size given operation op and data set ds; $rec(x)$ is the final recommendation given current operation x, in other words, rec is a list of operations that are conducted along with the current operation x, in order to generate potential interesting analysis results for the future operation.

Algorithm 1 Memory-Constrained Recommendation

```
Thread 1:Pattern Construction
if num(newop) = period then
  if num(log) < 2 then
    Select random one
  else
    Construct Memorable Pattern, P_s
    Construct Memoryless Pattern, P_m
  end if
end if
Thread 2:Recommendation
/*Check the Memory Capacity*/
while size < ls&&(nseq + nmar) > 0 do
  if nseq! = 0
   then
    /*Accept Sequence Pattern First*/
    if size + size(p_s(x)_i, ds) < ls  then
      rec(x)+ = p_s(x)_i;
      i + +;
      nseq − −;
    end if
  end if
  if nmar ! = 0 then
    /*Accept Markov Pattern Second*/
    if size + size(p_m(x)_j, ds) < ls  then
      rec(x)+ = p_m(x)_j;
      j + +;
      nmar − −;
    end if
  end if
end while
```

B. Finer Segmentation and In-Advance Analysis Kernel

The recommended analytics require additional computational resources, though the computing is considered "virtually free" for big data problems where data movement dominates the runtime. As shown in Figure 5, the 'Kernel' part contains 'Data' and 'Chunk'. The 'Data' here refers to the current operation's input, which is already fetched from storage nodes to compute nodes. Before removing the data or writing back the update to storage nodes, the Segmented In-Advance Data Analytics System recommends the potential analytic operations, i.e., 'Rec-op', and performs on the ready 'Data'. If the future operation has overlapping with existing analytic results, the I/O is reduced by reusing them. The 'Data' are segmented (in this paper, we use the term 'segment' interchangeably with 'chunk') in order to gain finer granularity analytic sub-results and to construct the future analytic results [22]. The finer the sub-results, the more chances the sub-results can be re-used, because the possibility of useable overlap among data is increased. Besides the granularity, how to segment the data plays an important role in the re-using ratio. We adopt a 'dimension-driven segmentation' method [22], and add the 'Rec-op' in the kernel. Therefore, the kernel in this system has more analytic work to perform on the segmented datasets, and generates more analytic sub-results for future re-use.

C. Elastic Distributed In-memory Database

As shown in the Figure 5, the Segmented In-Advance Data Analytics System maintains all analytic results, including segmented sub-results of the recommended analytics, in database for future overlap detection and results re-using. We demonstrated an initial design in [22], where only the host node is used to store the intermediate results as a file. Each time the system detects the overlapping, one process will open the intermediate file. With neither consideration of query performance nor memory limit, the major drawback of such design is its poor scalability. In current big data analytics, both the raw data and the intermediate results tend to increase. Our system utilizes a distributed memory database to manage all the analytic intermediate results, such that the available cache size is large and the query response is fast. There are many sucessful distributed in-memory databases [6], for example, IBM (Informix, DB2-BLU, SolidDB), Oracle (RDBMS,Coherence, Exalytics, TimesTen), and WebDNA. Since the system only desires a simple elastic database, without strong requirement of fault-tolerance or data consistency, we choose memcached as the database server and libmemcached as the client library.

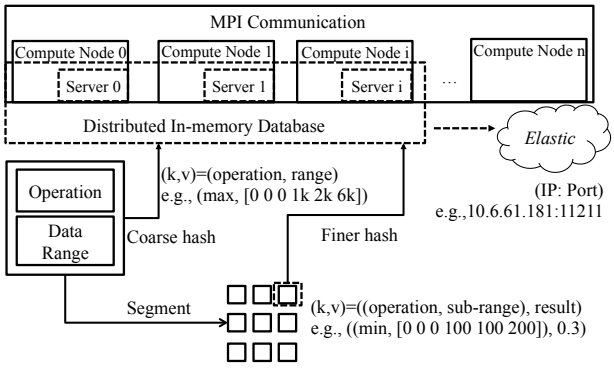

Fig. 6: Distributed In-memory Database

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2541142, IEEE Transactions on Cloud Computing

The challenge lies in the mapping from our system's data structure to the memcached's key-value structure. Currently, we design a two-level hashing to address this. As shown in Figure 6, a cached result eventually consists of three parts, i.e., 'operation', 'data range' and 'result value'. The 'operation' defines what kind of analytic operation is used, 'data range' specifies what part of dataset is analyzed by this operation, and the 'result value' is the analytic result.

The database is designed to facilitate the overlap detection. A new analysis has overlap with cached result if the operation is same (current design also requires the operations to be additive) and the data range has overlap. For example, one new analysis is 'max', and its data range is $start[0,1,2] = \{0,0,0\}$ and $length[0,1,2] = \{100,500,100\}$. In the database, if there is an existing 'max' result and the data range is a subset of the analysis data range, e.g., $start[0,1,2] = \{50,100,50\}, length[0,1,2] = \{50,200,10\}$, the system can re-use it. Clearly, given new analysis data range $(start, length)$ and cached data range $(cstart, clength)$, we can use Equation 11 to check this overlap.

$$\begin{aligned} &cstart_i >= start_i \\ &clength_i <= length_i - (start_i - cstart_i) \\ &i = 0,1,..,n \end{aligned} \tag{11}$$

To fit into the 'key-value' memcached database, the 'key' of a cached result is $(Range, Operation)$. However, we can not simply compact the $(Range, Operation)$ as 'key' and hash it into the database, because there is no way to detect the overlap $(Range)$ using the unique hashed key$(Range, Operation)$, unless the data range is exactly the same. In order to detect the overlap and utilizing the key-value's fast search, we design a two-level hashing method. In the first level, which is a coarse level, we set the 'operation' as key, the data range of original I/O (not segmented) as value. We use '$memcached\_append$' to append the same operation's different range. During the overlap detection, we first query the database with the 'operation', and get a list of coarse ranges, then we compare each coarse range with the current new range to detect the intersection. There are many powerful intersection detection library, e.g., in this paper, we use CGAL [32] (which has lots of efficient and reliable geometric algorithms). In the second level, after we select the intersected coarse ranges, we continue with a finer level overlap detection. Given the segmentation knowledge, each of the coarse range in the remaining list will be re-applied the segmentation algorithm to form the sub-chunk ranges. Using Equation 11, we detect the sub-ranges that are fully overlapping with any of the new analysis sub-range. Then we can retrieve the result for this sub-range from database. To store this finer-level information into database, since the sub-chunk range is minimum range unit, therefore we do not need to keep it, and we compact it with the 'operation' as key, and the 'result value' as value. For example, suppose the current data range is $range0 : start[0,0,0], length[100,100,100]$, and the current operation is max. The Segmented In-Advance Data Analytics System will first query the database for the same operation max, and return a list of data ranges that are previously accessed. Suppose the database returns the following sub-ranges,

1) $range1 : start[10,10,0], length[100,100,300])$
2) $range2 : start[300,500,1000], length[100,100,100]$
3) ...

then we use the CGAL's intersection detection algorithm [32] to detect which range has intersection with the current data range. In this example, the $range2$ will be filtered out, because it does not have any intersection with the $range0$. The remaining list $(range1, ...)$ will be re-applied the segmentation algorithm to construct the segments. For example, if the segmentation algorithm segments the third dimension with size of 100, then we can get a list of segments from $range1$, i.e.,

1) $segment1 : start[10,10,10], length[100,100,100]$
2) $segment2 : start[10,10,100], length[100,100,100]$
3) $segment3 : start[10,10,200], length[100,100,100]$

Since the segments are the minimum data set in the database, we can re-use it only if it can be covered by the current data range. In this case, only $segment1$ can be re-used. Then the result of $segment1$ is queried from database use the key "segment1, max". Clearly, with even finer segmentation, there will be more sub-results.

In Figure 6, we show the design of in-memory database for the Segmented In-Advance Data Analytics System. The database is designed to be elastic, which means that the users can submit the resource requirement based on their application. If the application re-uses many previous analysis results iteratively, then the user can ask for more database servers. The system allocates the required capacity by requesting the IP, port number, and memory size. Such cloud-based elastic data store augments the scientific big data analysis and has great potential to help various applications in adopting our Segmented In-Advance approach.

VI. Implementation

The Segmented In-Advance (SIA) system is currently implemented based on memcached 1.4.18 [7], libmemcached 1.0.18 [5], pnetcdf 1.4.1 [8], CDO 1.6 [1]. We implement the SIA system to accept the users' job script. The SIA will first build the connection with the memcached database server from the users' first job. For each job, the SIA will query the database for the analysis overlap, and then perform the optimized I/O by submitting the parallel job in a computing cluster. The SIA will also recommend operations based on the current operation and compute the data when they are ready on the compute nodes. Initially, we allocate several memcached servers on compute nodes using 'Qlogin'. The user, however, does not have to use all the available memcached databases. Depending on how much the application needs, the SIA system will connect to the database with IP and port numbers. The user can specify more memories or database servers in the next job, and the SIA will look up the available sources and add on more servers. Such design makes the in-memory database elastic for applications.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2541142, IEEE Transactions on Cloud Computing

VII. Evaluation

A. Experimental Setup

We ran all the experiments on the Hrothgar [4], a 640-node (7680 cores) Linux cluster. Each node in the cluster contains two Intel Xeon 2.8 GHz 6-core processors with 24 GB of memory. The nodes are connected with DDR Infiniband. The main Hrothgar cluster has a DataDirect network storage system capable of providing storage for up to one petabyte of data. The cluster file system is based on Lustre and provides a shared file system. We generated several 4-D climate datasets using pNetCDF. The size of those datasets are 769MBs, 179GBs, 358GBs, and 448GBs.

B. Evaluation and Results Analysis

When applying the Segmented In-Advance Data Analytics system, we are interested in how the system performs in terms of the I/O performance, query performance, scalability as well as practicality. Therefore, we set up two groups of tests. In group one, we evaluated the hit rate of the recommendation algorithm and the I/O performance of Segmented In-Advance Data Analytics system with/without the recommendation. For the evaluation of the recommendation, we have two patterns, i.e., memorable pattern and memoryless pattern, to be measured. Our system predicts all operations within a pattern sequence as long as this pattern is detected. For memoryless pattern, we observe the hit rate with different number of prediction steps. The number of prediction steps refers to the number of operations to be selected based on the transition probability from high to low.

Regarding the training and test set, we have a small set of real scripts collected from climate community [20]. The majority of operations in the scripts are memorable/fixed patterns. We also generated two synthetic scripts that simulate the random analytics. The two training sets have 100 and 1000 operations separately. In practice, though the different patterns are mixed, our current setup evaluates them separately (using different training data) is close to the real situations.

The number of unique operations is 52, and the test set is another 100 and 1000 operations set. The same set of experiments are repeated 180 times and the result plotted is their average. Figure 7 shows the hit rate along different prediction steps. As shown in the Figure 7, the more the training set, the more accurate the prediction. However, Figure 7 shows that the real script is immune to the number of prediction steps, and the real scripts get higher hit rate even with one predicton step. This is because the real script contains more meaningful analytics pattern, and once detected, later occured operations are easily predicted. On the other hand, the randomly generated training sets have few meaningful patterns, and only the prediction steps can affect their hit rate. The experimental results confirm our analysis. Besides, in Figure 7, the 'Theory' line shows a theoretical hit rate, i.e., the number of steps divided by total unique operations. The Markov algorithm outperformed the pure random selection due to the accumulation of the transition probability. This transition

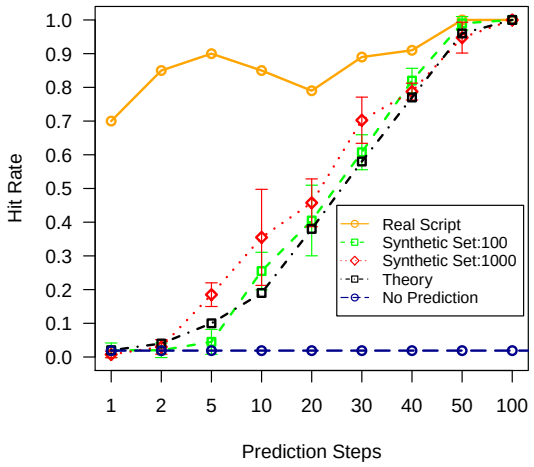

Fig. 7: Hit Rate with Different Prediction Steps

probability matters more in practice. At last, 'No Prediction' clearly has no capability to predict the future, which is used in the previous work [22].

The hit rate is only one factor that can have effect on the I/O performance though. A high hit rate still can not guarantee a high I/O performance, but it is always better to achieve a good hit rate. Therefore, we would use the Markov algorithm to predict 30 or more steps (users can decide) to gain 50%+ hit rate. For memorable pattern, we only use one prediction step, which is enough to achieve high hit rate.

To observe how the I/O can be improved with the recommendation, we run the Segmented In-Advance Data Analytics system over a 179GB synthetic climate dataset. In this test, we initialized one memcached instance, with 1024 MB capacity. The training scripts for the recommendation is the previous set. We submitted analytics operations randomly to the Segmented In-Advance Data Analytics system and measured the the I/O time with and without the recommendation. The tests were conducted with 96 processes and 100 total OSTs. Figure 8 shows that the I/O cost becomes less as the recommendation providing higher hit rate and the data has more overlapping. This observation confirms two necessary conditions, in which the data movement is reduced only if both the computation and I/O are overlapped. Observing the surface in the figure, we can also find that the I/O performance is not guaranteed given certain pair of hit rate and overlap percentage. For instance, the case with hit rate=0.2, overlap percentage=0.9, should be better than the case with hit rate=0.1 and overlap percentage=0.9, but it turns out the opposite. This situation occurs because that the hit rate is a probability that the recommended operation is hit, and does not mean it will be hit. Therefore, we need to use our recommendation algorithm to predict more steps of operations and cache more results for future. On the other hand, the overlap percentage is proportional to the I/O

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2541142, IEEE Transactions on Cloud Computing

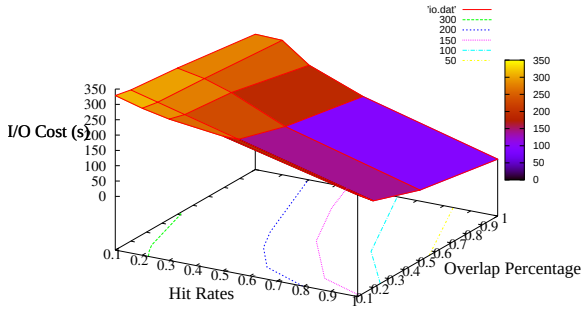

Fig. 8: I/O Performance with Recommendation

performance. This observation shows the importance of our system's finer segmentation, which helps in detecting more data overlap.

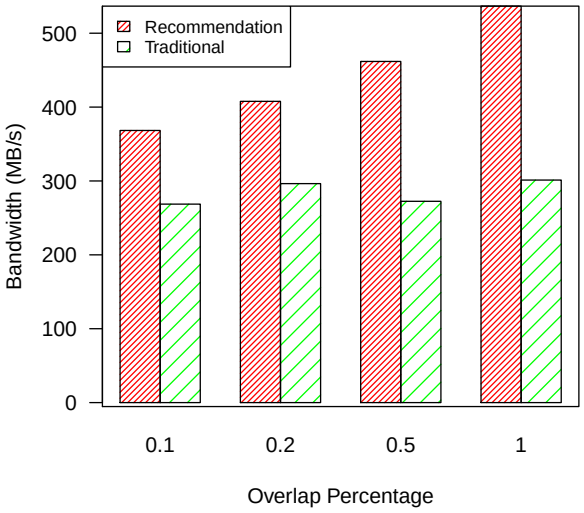

Fig. 9: I/O Bandwidth Comparison with Recommendation

With a 50% hit rate, we plot Figure 9 to show the relation between the I/O bandwidth and the overlap percentage. We also compare our system with the traditional method, in which no recommended analysis are performed and no additional analysis results are cached.

In group two, we evaluated the query efficiency using the distributed in-memory database. Figure 10 shows the performance comparison with and without the in-memory database. This group of tests used one instance memcached and allocated 6GBs memory. We generated different size of analysis results and stored them in file and the database separately. We performed 10,000 times of query and plot the total response time. To be precise, in each test, the results stored in file are all read into memory and the query is performed in memory. On the other hand, the in-memory database is started from the first test and keeps running as a distributed server.

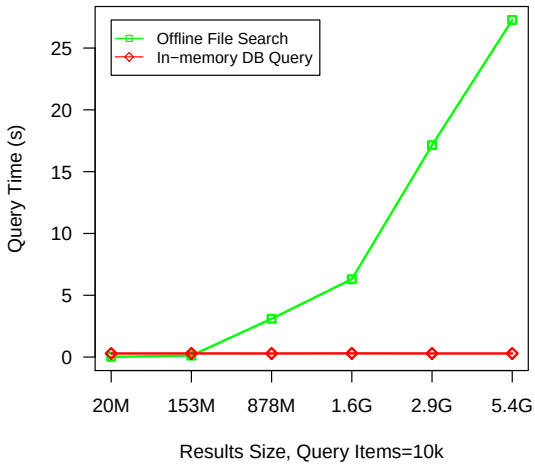

Fig. 10: Results Query Performance Comparison

We can observe that using in-memory database, the query response is constant and is only proportional to the number of query items. While the traditional 'offline file search' keeps increasing the response time as more results accumulated. To utilize the data analytics ahead as a technique to reduce the data movement, we must ensure that the technique will not cause much overhead. We can not afford a query cost as 25 seconds (Figure 10,size=5.4G), to complete each analytic operation. Our evaluation shows that the distributed in-memory database is promising to help reducing the data movement for big data analytic problems.

In the segmented in-advance framework, we have designed a fine-grained segmentation component. Here we change the access pattern and observe the performance with different segmentation granularity. In Figure 11, 'SegA' is the default configuration in our system, which turns on all the segmentation strategy. 'SegD1', 'SegD2' and 'SegD3' are three different segmentation that has granularity of 1D, 2D and 3D separately. The access pattern is ranges from 1D access, 2D plane access and 3D subcube access. We can see from Figure 11, without the default segmentation strategy, none of the other segmentation can always achieve a best performance. For example, 'SegD2' performs badly when the access pattern is on 3D subcube. 'SegD1' is supposed to achieve a good performance when the access pattern is 1D or 2D. The reason that 'SegD1' does not performs good in 2D case, is that there is additional cost for constructing a 2D plane from the 1D results in the cache. This test shows us that a flexible and finer segmentation strategy is necessary to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2541142, IEEE Transactions on Cloud Computing

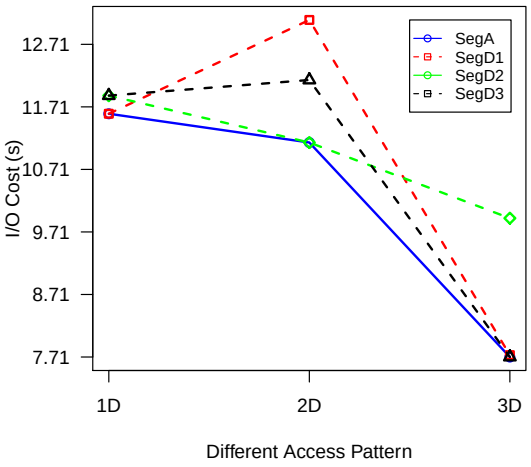

Fig. 11: Fine-grained Segmentation Evaluation

achieve good performance. We will conduct more research on developing an intelligent segmentation strategy, in which the most frequent access pattern can be better served using flexible segmentation.

VIII. Related Work

A. Result Caching in Cloud Computing

Instead of caching the raw data, result caching and re-using is a lightweight yet powerful technique. In cloud computing, caching results is a technique to achieve fault tolerance (RDD [37])cite linkedin and incremental computing [13], [10]. The idea of RDD (or later on Spark) is to keep the partitioned operation and recompute the data using lineage for fast fault tolerance. Twister and Incoop support iterative computing via caching partial results and sharing with other nodes [13], [10]. These works, however, are mainly focusing on systems' reliability and applications' parallelism. Besides, those works target mapreduce jobs. Another example is ReStore [14], where the output of whole MapReduce jobs can be reused for the larger workflow. How to fully utilize the diverse analysis results and efficiently reuse them for reducing data movement in scientific applications are not the major consideration in those systems. Our system is designed for scientific big data applications, in which the analysis pattern and data movement pattern is quite complex. Our Segmented In-Advance system is designed to reuse the analysis results by detecting the computation and I/O overlapping. We have also designed a fine-grained segmentation strategy utilizing various access pattern and the specific formats of scientific datasets.

B. Reducing Data Movement in Scientific Applications

Compression [21], [11], [16], in-situ processing [36], [9], [17], PreDatA [38], Adaptable I/O system [25], [24], [33] and active storage [29], [26], [35] are well-known techniques to reduce the scientific application's data movement. In ISABELA [21], a new compression algorithm is designed for writing HDF5 dataset, and the data movement is largely reduced after the compression. However, the compression and decompression both bring additional cost for the analysis workflow. In-situ processing is a method used during the data generation. When the scientific application or simulation generate data on compute nodes, before sending over the network, the in-situ analysis kernel is integrated into the application code, and certain amount of computation tasks are completed before data are written on storage nodes. The major drawback of this technique is the lack of knowledge of the analysis workflow. Active storage also largely reduces the data movement by moving computing task near to the data location. While the storage may not be able to deliver enough computing resources for the scientific applications. In our previous work, we designed a segmented analysis system to optimize the I/O performance. The idea is to cache the current analysis results for future re-use. However, the performance gain depends on whether the results can be hit. In this new work, we propose to use Segmented In-Advance data analytics to predict the useful operations based on different pattern, and cache additional results for increasing the hit ratio. This work is not only a finer combination of segmentation and in-advance prediction, but also a new framework that utilizes the analysis pattern as well as the data movement pattern in scientific big data analysis, such that big data movement is reduced.

C. Knowledge Discovery in Scientific Visualization

Scientific discovery continues to be a hot topic as the data management and real-time visualization bring challenges to the scalability and performance. Traditional discovery is defined as 'the nontrivial extraction of implicit, previously unknown, and potentially useful information from data' [15]. It focuses on using various machine learning or statistical methods to explore the data for unknown knowledge [30], [18], [27]. Our work focus on the middle layer data movement to support the upper data analysis. Our proposed segmented in-advance system has a function to observe the user's analysis pattern and tries to make a recommendation for scientists, which in some sense benefit the data analysis. Rather than intelligently designing a machine learning algorithm for scientists to discovery the knowledge, we argue that our system provides a systematic lightweight framework (not an algorithm) for scientific applications, with the goal of reducing the huge amount of data movement, and indirectly speeding up the data analysis. Future research may include more intelligent recommendation algorithms in the middle layer.

IX. Conclusion

In this study, we have introduced a new *Segmented In-Advance* (SIA) data analytics method with cloud-based distributed database support, for reducing data movement for big data analysis and big data applications. The proposed SIA data analytics leverages a recommendation system that uses minimal computing resources to generate interesting analysis

results in advance. The proposed SIA data analytics approach also uses a distributed in-memory database for managing the analytic results. When these results are hit by demand analytic operations, there is no need to access the requested data anymore as long as the the results are ready in the distributed database. As data movement dominates the execution time of many big data analysis, and computing is virtually free, the SIA data analytics can be a promising solution that fully leverages data locality and reduces the data movement. The evaluation results show that the SIA data analytics can dramatically reduce the data movement and improve the big data application's I/O performance from 1.2X to 6X speedup, with 50% data overlapping and 10%-100% operation predication hit rate.

The *Segmented In-Advance* data analytics extends our prior work of segmented analysis [22] and in-advance data analytics [23], and provides a thorough study, an integrated framework and a cloud-based distributed data store. It further advances the state of the art of big data analytics and data movement reduction methods. In the future, we plan to further evaluate it with more analytic operations.

X. Acknowledgments

This research is sponsored in part by the National Science Foundation under grant CNS-1338078 and CCF-1409946. We also acknowledge the High Performance Computing Center at Texas Tech University for providing the resources that have contributed to the research results.

References

[1] Climate data operator. https://code.zmaw.de/projects/cdo.

[2] Community climate system model. http://en.wikipedia.org/wiki/General_Circulation_Model.

[3] General circulation model. http://en.wikipedia.org/wiki/Community_Climate_System_Model.

[4] Hrothgar. http://www.hpcc.ttu.edu/.

[5] Libmemcached. http://libmemcached.org/.

[6] List of in-memory database. http://en.wikipedia.org/wiki/List_of_in-memory_databases.

[7] Memcached. http://memcached.org/.

[8] Parallel NetCDF. www.mcs.anl.gov/parallel-netcdf.

[9] H. Abbasi, G. Eisenhauer, M. Wolf, K. Schwan, and S. Klasky. Just in time: adding value to the io pipelines of high performance applications with jitstaging. In *HPDC*, pages 27–36, 2011.

[10] P. Bhatotia, A. Wieder, R. Rodrigues, U. A. Acar, and R. Pasquin. Incoop: Mapreduce for incremental computations. In *Proceedings of the 2Nd ACM Symposium on Cloud Computing*, SOCC '11, pages 7:1–7:14, New York, NY, USA, 2011. ACM.

[11] K. M. Curewitz, P. Krishnan, and J. S. Vitter. Practical prefetching via data compression. In P. Buneman and S. Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 257–266, Washington, D.C., 26–28 May 1993.

[12] T. L. Delworth. Gfdl's cm2 global coupled climate models. *J. Climate*, 19:643674, 2006.

[13] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S. hee Bae, J. Qiu, and G. Fox. Twister: A runtime for iterative mapreduce. In *In The First International Workshop on MapReduce and its Applications*, 2010.

[14] I. Elghandour and A. Aboulnaga. Restore: Reusing results of mapreduce jobs. *Proc. VLDB Endow.*, 5(6):586–597, Feb. 2012.

[15] W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus. Knowledge discovery in databases: An overview. *AI Mag.*, 13(3):57–70, Sept. 1992.

[16] M. Gardner, W. chun Feng, J. Archuleta, H. Lin, and X. Ma. Parallel genomic sequence-searching on an ad-hoc grid: Experiences, lessons learned, and implications. In *SC'2006 Conference*, Tampa, FL, Nov. 2006.

[17] J. Gray, D. T. Liu, M. A. Nieto-Santisteban, A. S. Szalay, G. Heber, and D. DeWitt. Scientific data management in the coming decade. Technical Report MSR-TR-2005-10, Microsoft Research (MSR), Jan. 2005.

[18] A. J. Hey, S. Tansley, K. M. Tolle, et al. *The fourth paradigm: data-intensive scientific discovery*, volume 1. Microsoft Research Redmond, WA, 2009.

[19] S.-L. Kang and B. G. H. A large eddy simulation study of moist convection initiation over heterogeneous surface fluxes. In *Monthly Weather Review*, pages 2901–2917, 2011.

[20] F. Kaspar, U. Schulzweida, and R. Muller. Climate data operators as a user-friendly processing tool for cmsaf's satellite-derived climate monitoring products. In *the Proc. of the EUMETSAT Meteorological Satellite Conference*, 2010.

[21] S. Lakshminarasimhan, J. Jenkins, I. Arkatkar, Z. Gong, H. Kolla, S.-H. Ku, S. Ethier, J. Chen, C.-S. Chang, S. Klasky, R. Latham, R. B. Ross, and N. F. Samatova. ISABELA-QA: query-driven analytics with ISABELA-compressed extreme-scale scientific data. In *SC 2011, Seattle, WA, USA, November 12-18, 2011*, page 31, 2011.

[22] J. Liu, S. Byna, and Y. Chen. Segmented analysis for reducing data movement. In *the Proc. of the IEEE International Conference on Big Data,(Bigdata'13)*, 2013.

[23] J. Liu, Y. Lu, and Y. Chen. In-advance data analytics for reducing time to discovery. In *Big Data (Big Data), 2014 IEEE International Conference on*, pages 329–334, Oct 2014.

[24] J. Lofstead, M. Polte, G. Gibson, S. Klasky, K. Schwan, R. Oldfield, M. Wolf, and Q. Liu. Six degrees of scientific data: Reading patterns for extreme scale science io. In *In Proceedings of High Performance and Distributed Computing*, 2011.

[25] J. Lofstead, F. Zheng, S. Klasky, and K. Schwan. Adaptable, metadata rich io methods for portable high performance io. In *In Proceedings of IPDPS'09, May 25-29, Rome, Italy*, 2009.

[26] X. Ma and A. L. N. Reddy. MVSS: An active storage architecture. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, PDS-14(10):993–1005, Oct. 2003.

[27] L. Magnani, N. Nersessian, and P. Thagard. *Model-based reasoning in scientific discovery*. Springer Science & Business Media, 1999.

[28] A. Parameswaran, N. Polyzotis, and H. Garcia-Molina. Seedb: Visualizing database queries efficiently. Technical report, Stanford University, 2013.

[29] J. Piernas, J. Nieplocha, and E. J. Felix. Evaluation of active storage strategies for the lustre parallel file system. In *SC'07*. ACM/IEEE, Reno, NV, Nov. 2007.

[30] K. Popper. *The logic of scientific discovery*. Routledge, 2005.

[31] R. Ross, R. Latham, M. Unangst, and B. Welch. Parallel I/O in practice, tutorial notes. In *SC'08*. ACM/IEEE, Austin, TX, Nov. 2008.

[32] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 4.4 edition, 2014.

[33] Y. Tian, S. Klasky, H. Abbasi, J. Lofstead, R. Grout, N. Podhorszki, Q. Liu, Y. Wang, and W. Yu. Edo: Improving read performance for scientific applications through elastic data organization. In *In Proceedings of IEEE Cluster 2011*, 2011.

[34] D. Wang, C. Zender, and S. Jenks. Efficient clustered server-side data analysis workflows using swamp. *Earth Science Informatics*, 2(3):141–155, 2009.

[35] R. Wickremesinghe, J. S. Chase, and J. S. Vitter. Distributed computing with load-managed active, storage. In *Proc. 11th IEEE International Symposium on High Performance Distributed Computing (11th HPDC'02)*, pages 13–23, July 2002.

[36] H. Yu, C. Wang, R. W. Grout, J. H. Chen, and K.-L. Ma. In situ visualization for large-scale combustion simulations. *IEEE Computer Graphics and Applications*, 30(3):45–57, May/June 2010.

[37] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. Technical Report UCB/EECS-2011-82, EECS Department, University of California, Berkeley, Jul 2011.

[38] F. Zheng, H. Abbasi, C. Docan, J. Lofstead, S. Klasky, Q. Liu, M. Parashar, N. Podhorszki, K. Schwan, and M. Wolf. Predata - preparatory data analytics on peta-scale machines. In *In Proceedings of 24th IEEE International Parallel and Distributed Processing Symposium, April, Atlanta, Georgia*, 2010.