2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems

Bi-Cluster：A High-performance Data Query Framework for Large-scale Scientific Data

Yixian Shen
National Supercomputer Center in Guangzhou, School of Data and Computer Science, Sun Yat-sen University
Guangzhou,China
shenyx9@mail2.sysu.edu.cn

Cheng Peng
College of Computer, National University of Defense Technology
National Supercomputer Center in Guangzhou
Changsha,China
peng_cheng_13@163.com

Yunfei Du✉
National Supercomputer Center in Guangzhou, School of Data and Computer Science, Sun Yat-sen University
Guangzhou,China
duyunfei@mail.sysu.edu.cn

Yutong Lu
National Supercomputer Center in Guangzhou, School of Data and Computer Science, Sun Yat-sen University
Guangzhou,China
yutong.lu@nscc-gz.cn

***Abstract*—Emerging scientific computing generates massive amounts of scientific data by relying on high-performance computer systems, challenging data management and analysis. State-of-the-art query technology such as FastQuery supports directly indexing of scientific data but does not support in-situ index construction, build index extremely slow and generate huge volume index files. Block index technology coarsely retrieves a large amount of redundant data increasing the filtering overhead. Therefore, this paper proposes a high-performance query data framework for scientific data. In terms of index generation, a two-tier index data structure is designed to build the index in parallel to reduce the size of index and speed up index generation. Meanwhile, an in-situ kernel index parallel strategy is proposed to build the index for online real-time generated data. In terms of data retrieval, a two-tier parallel query mechanism is designed to efficiently read data, and a dynamic union read strategy and an adaptive scheduling strategy are used to optimize the data retrieval process. Finally, the Bi-Cluster Framework is evaluated on scientific datasets, which proves that our design achieves good performance. The size of index and index generation time is much smaller than FastQuery. In terms of retrieving data, data retrieval performance has improved a lot. The scalability of Bi-Cluster is pretty good by evaluating on 12288 cores.**

Keywords—High-performance Computing, Bitmap Index, Min-Max Block Index, Data Retrieval, Load Blancing

I. INTRODUCTION

Scientific data comes from scientific applications, scientific observations, and large-scale simulation experiments, such as turbulent combustion simulation, plasma physics simulation [1], and cosmological observations [2]. There are three main reasons for the massive scientific data generation. Firstly, large-scale particle accelerators and telescopes, high-throughput gene sequencers and other large scientific devices are widely used for exploring science which produces data [3,6] all day long. Secondly, hardware computing power such as CPU and GPU, the network bandwidth and the storage capacity are rapid-growing. Supercomputers[7,8] can provide superior computing power for analyzing data. Finally, the deep learning and machine learning surges, massive observations and simulation data have brought scientific research into an unprecedented era of scientific big data, and the patterns of scientific discovery will undergo major changes. "Data-intensive science" has become the fourth new model of scientific discovery [4].

As the exascale computing era approaches, large-scale data analysis on high-performance platforms is becoming increasingly important[15]. Traditional supercomputers focus on computing-intensive task optimization. With the increasing massive scientific data, supercomputers are required to equip with strong analytical capabilities [5]. The rapid growth of high-performance computing systems has injected new impetus into emerging scientific discovery and engineering innovation.

Tianhe-2A is a group of heterogeneous supercomputer located in National Supercomputer Center in Guangzhou with capable of a peak performance of 100.67 PFlops and sustained performance of 61.4 PFlops after replacing the original Intel Xeon Phi accelerator with the domestic accelerator Matrix 2000. At present, a large number of scientific calculations are carried out on the Tianhe-2A. The fields of application mainly include atmospheric marine environment, astronomical geophysics, industrial design and manufacturing, bio-health medical care, new energy and new materials, and smart cities. PB level data was generated every day. An effective data query framework is urgently needed on Tianhe-2A.

Large-scale scientific data is equipped with low value density. The time overhead of scanning all data to locate critical information is unacceptable. Indexing plays an important role in the field of data science. In the traditional database management system, the task is OLTP (Online Transaction Processing) [9] which read load and write load are quite balanced, while the scientific data input and output exists in the form of files. Importing scientific data into the database will generate time-comsuming data migration overhead, besides the scientific data is OLAP (On-Line Analytical Processing) [10], which is often the feature of reading multiple times. Therefore, this paper proposes to establish high-performance data query framework for scientific data on the Supercomputing computing platform. The main contribution of Bi-Cluster is as follows:

- We design a two-tier query mechanism and a two-layer index data structure which reduce index size and speed up the process of query.

978-1-7281-2058-4/19/$31.00 ©2019 IEEE
DOI 10.1109/HPCC/SmartCity/DSS.2019.00130

- We propose in-situ kernel parallel index generation mechanism which build index for real-time generated data, removing the need of reading data from the disk.
- We design an adaptive scheduling algorithm to optimize the load imbalance caused by inconsistent computing tasks in the process of data query.
- We design dynamic joint reading strategy to avoid excessive redundant data reading into the system and also to prevent excessive I/O access overhead.

We evaluate index generation method and data retrieval at three scientific datasets. In terms of index generation, the index size and generation speed outweigh FastQuery. In terms of data retrieval, compared with FastQuery, the retrieval speedup can be up to 6.9 times. Compared with block index technology, the retrieval speedup can be up to 13 times. Besides, we evaluate the performance between online and offline data index generation method using PiBase simulation software which demonstrate in-situ index generation method is over of the offline method. Finally, we test the scalability of Bi-Cluster which proves that our design support sublinear expansion ability.

This paper is organized as follows. Section II briefly introduces and discusses other frameworks for data query. Section III shows the architecture design, key components and implementation of Bi-Cluster. Section IV describes experimental results, demonstrates and compares with other systems to draw conclusions in the real environment. We summarize the paper in Section V.

II. Related Work

A. SciDB Scientific Database

SciDB[11] is a new scientific database system for astronomy, particle physics, nuclear fusion, remote sensing, oceanography and biology. By analyzing the traditional SQL language does not meet the needs of scientific analysis, SciDB explores an array data model to satisfy user needs. SciDB supports nested, multi-dimensional data models that can be applied to the analysis of scientific data. However, during the process of analysis, the original data needs to be imported into the SciDB database. When the amount of data is small, this overhead is acceptable, but as the dataset continues to increase, unacceptable conversion overhead is generated, which seriously affects the efficiency of scientific data analysis.

B. FastQuery Parallel Index Framework

FastQuery[2] is a parallel file index framework for the massive scientific data designed by the Lawrence Berkeley Lab in the United States, which speeds up data queries by directly buiding index in the original scientific dataset. FastQuery uses Bitmap[13,14] as the index data structure and maps the array data model to the relational data model. FastBit[17] is applied to compress, bin, and encode bitmap indexes to reduce bitmap size which shows that accelerating query processing by at least an order of magnitude in many different applications[18]. But in some cases, the size of index generation is still large which is even as same as the size of the original data. Moreover, FastQuery generates indexes at a slower speed, and the time overhead is significant when generating indexes for large-scale data.

C. In-Situ Index Technology

The in-situ data processing method [19,22] aims to bypass disk access as much as possible. The basic idea is that the data is executed in memory while performing a series of predetermined operations in memory. Currently, many in-situ processing methods are only for parallel between nodes, and do not consider kernel parallelism.

D. Block Index Combined the ADIOS Technology

Reading the continuous data at a time is the motivation for block index[16]. Therefore, the read time of a single record can be as long as the read time of the data block. In addition, reading one block at a time can greatly reduce the number of I/O requests compared to random access on a single data record.

ADIOS is a middleware for high-performance I/O that enables the implementation of the I/O layer away from application scientists [21] and provides users with the flexibility to work between different I/O implementations via XML configuration files. Block index technology combined ADIOS[20] can accelerate building index. Good performance can be achieved when retrieving large amounts of data. However, when a tiny fraction of the data need to be retrieved, the process of data retrieval is very low and relatively time-consuming.

III. Architecture And Design

Bi-Cluster is divided into two parts: Index Generation and Data Retrieval. For index generation, we take two different subsystems processing offline data and real-time generated data. We adopt offline data parallel indexing subsystem (OFPI) for offline data, and adopt in-situ kernel parallel index generation subsystem (ISPI) for real-time generated data. In terms of data retrieval, we use high-performance parallel retrieval subsystem (HPR) to efficiently query data. Based on these, the system architecture is shown as Fig.1.

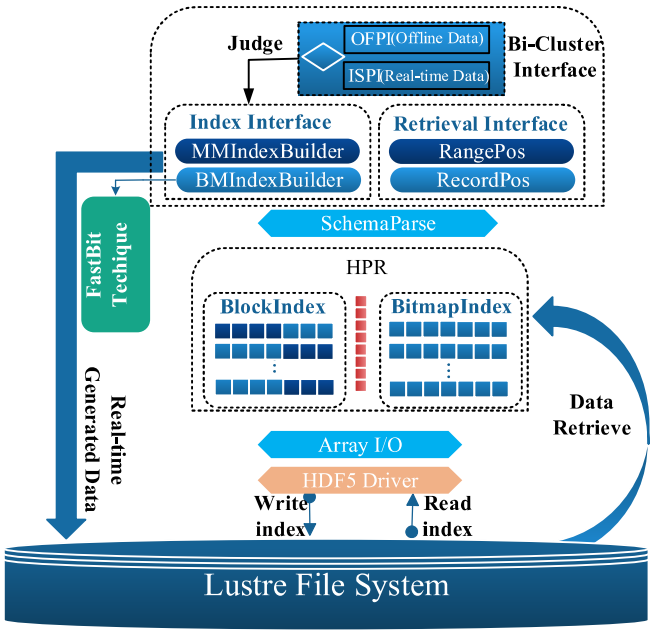


Fig.1 Bi-Cluster Architecture

A. OFPI

OFPI is a subsystem for generating indexes for offline data. The offline data volume is usually very large which cannot be read into the memory to build index at one time. Therefore, we need to partition the data and utilize muti-cores technology to read them in parallel. A two-tier index structure and write load balancing strategy are designed in this section. The OFPI strategy is designed as Fig.2:

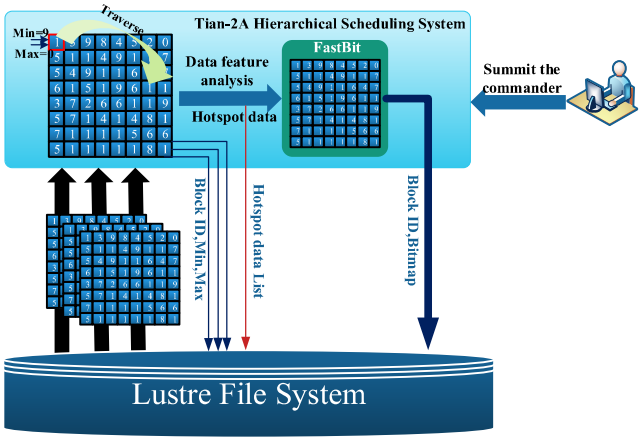


Figure 2 Offline Data Parallel Indexing Strategy Design

A Two-tier Index Structure Design: We divide the dataset into data blocks of fixed size such as 64MB. Each data block has a unique identifier. Then we read the data blocks into memory in parallel. At first, utilizing the Min and Max pointers to traverse all data in a data block, we record the blockID, maximum, minimum as BlockIndex. Due to the feature data in scientific data needs to be visited frequently, data characteristics can usually be marked by variance, average, Top-max vector, Top-min vector. When building block index, data feature constants are also calculated synchronously. According to actual condition, we set a certain portion of data as hotspot data. Then we build the BitmapIndex for hotspot data. We build the index using the fastBit technology to reduce the BimapIndex size. The fastBit can set the bining coefficient to adjust index size level.

During the index generation, we take use of muti-cores to build index in parallel. Take the meteorological data collected by Guangdong Meteorological Bureau as an example. The temperature variable (Temp) can be divided into blocks, which can be divided into 24 subarrays for parallelization. Each sub-array has a unique ID, which is respectively Temp1[1:100], Temp2[101:200] ..., Temp24[2301: 2400]. The Tianhe node has 24 cores which each core opens a process to computer these subarrays and establishes the data index for these sub-blocks independently. Both coarse-grained BlockIndex and fine-grained BitmapIndex can be built in parallel.

Write Load Balancing Strategy: The BlockIndex files can be written to disk in parallel owing to the fixed offset value including blockID, maximum and minimum. However, the BitmapIndex files is not the same. Every data block has the same size and different cardinality, the Bitmap size is only related to cardinality. So the offset of BitmapIndex is different. A synchronization strategy on the write bitmap index is carried out to prevent the bitmap index from being overwritten.

B. ISPI

The essence of in-situ indexing is to build index for generated data directly in real time. No need to write data to disk first, read it again. Tianhe node allocation in an exclusive manner which we use the Tianhe-2A Hierarchical scheduling subsystem (Teno) to fine-grain control the computing resources. We allocate a part of cores as scientific computing task. The remaining cores are used to build index. Real-time generated data needs to accumulate to a certain volume before building index. During the scientific computing process, some tasks generate a lot of data. In order to avoid memory limitations, we adapt In-situ computing adaptive mechanism as follows.

In-situ Computing Adaptive Mechanism
1, Initialize Teno and Configure the n_{task} and n_{build}
2, Count the number of nodes in the Computing network
3, Set memory utilization threshold M= M_{occpy}/M_{total}
4, Broadcast polling memory usage per 1 second
5, Add up the M > 75% node and mark it busy node
6, Calculate the portion of busy nodes to the entire compute node, L=N_{busy}/N_{total}
7, If L≥60%, expand computing resources by a factor of two and reassign computing tasks
8, Continue to monitor and Repeat step 3

The method of constructing in-situ index and writing the index file are the same as those in OPFI. During the in-situ build process, the generated data can be directly written into the file independently in parallel. With this design, we reduce time-consuming disk access overhead and write collision domains.

C. HPR

Scientific data index stored on the Lustre parallel file system. We use multi-core advantages to read required data in parallel and efficiently perform data retrieval. The design of HPR is shown as Fig.3.

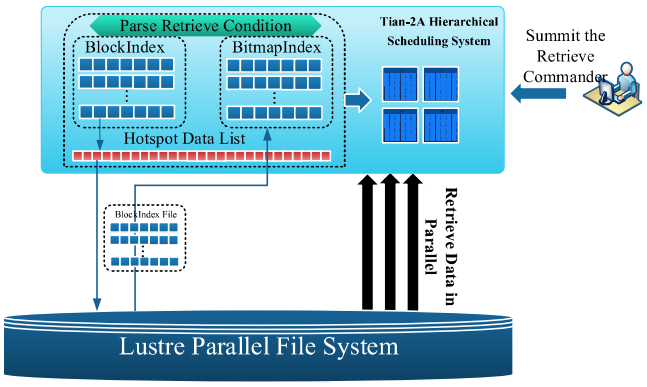


Figure 3 High-performance Parallel Retrieval Strategy Design

In the case of initializing computing resources, the data retrieval process is completed in five steps:

(a) Load the two-tier index file into the memory in parallel.
(b) Check the index boundary according to the BlockIndex
(c) Create a candidate list to collect candidate data block ID determining the data range.
(d) Check the candidate list to find out overlapped data, read hotspot data list and match the BitmapIndex.

(e) Determine the fine-grained retrieval scope according the matching and a little remaining candidate data BlockID was scanned to position the record in a data block.

Dynamics Joint Read Strategy: When indexing massive amounts of scientific data, it is difficult to predict the optimal index size. Meanwhile, when designing the index generation size, it is hard to accurately predict which data will be accessed. We design the dynamic joint read strategy to optimize the data retrieval. The size of data chunk is usually smaller than the physical block. During the data retrieval, we joint read the data surrounding the hotspot as shown below. We set different joint scopes by coefficient, but we can't exceed the memory limit.

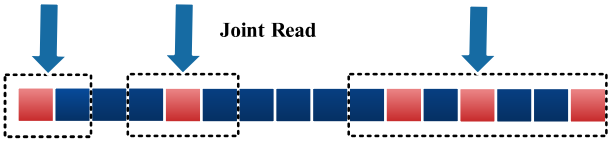


Figure 4 Dynamics Joint Read Strategy

Adaptive Scheduling Strategy: Retrieve tasks allocated to computer node is not evenly owing to BitmapSize. The previous section pointed out that bitmap size is only related to cardinality. High cardinality need to be retrieved for longer. During the static allocation, all tasks are equally distributed on each node which may cause the fast-calculated nodes to wait for the slow nodes. We design adaptive scheduling strategy to reduce waiting problem as shown below.

Adaptive Scheduling Strategy
1, Assign a part of tasks evenly to the nodes in the cluster
2, Monitor node memory usage per 1s, M= M_{occpy}/M_{total}
3, If M < 40%, double tasks allocated to node
4, Broadcast polling memory usage per 1 second
5, If M > 90%, kill nodes and resign tasks to other node.
6, Continue to monitor and Repeat step 2

IV. Expriments and Analysis

We conduct multiple groups of comparative experiments to evaluate Bi-Cluster performance. Firstly, we test the OFPI index generation compared with FastQuery and block index technique using the dataset ClimateGZ, VPDS, COSM. And then we conduct ISPI index generation experiments. And then we take the comparative experiments to analyze the HPR performance. Finally, we conduct scalability experiment evaluating the OFPI, ISPI and HPR.

A. OFPI Index Generation Experiments

The scientific dataset used is in HDF5 format. The HDF5 format is commonly used in scientific data to store running logs, metadata, and data in a file, while HDF5 files support parallel access. Three scientific datasets were used in our evaluation as described below.

ClimateGZ dataset is a meteorological data from Guangdong. It has been collected since 2000. Each file is about 150GB and the entire dataset size is 3TB. The data for each year is an HDF5 file that includes information on precipitation, temperature, and radar maps for 119 zones every 30 minutes. This data is used to record weather changes in Guangdong. It is used to predict weather based on historical data and extreme weather such as typhoons. Take the temperature for examples, If we analyze whether the extreme weather exists in Guangzhou during a certain time, the temperature variance of the dataset is an important indicator.

VPDS dataset was generated by plasma physics simulation software. The raw dataset contains a record of 10 billion particles, each associated with seven one-dimensional variables describing its position and energy. In our experiments, we use a subset dataset of 24 billion particles, about 900GB. In the assessment, we query the particles with the highest energy levels.

COSM is a dataset generated by simulation cosmics oftware. All observed planetary energy is aggregated into a single variable, containing a one-dimensional dataset of 1.74 billion records, and the entire file size is 974 GB. The query we evaluated was to find the planet with the darkest planet and locate the darkest planet by searching for records with the lowest amplitude values. The query we evaluated was to find the darkest planet and locate the darkest planet by searching for records with the lowest amplitude values.

Index Generation Experiment in a Single Node: We measure the time from reading data, retrieving index, and writing index file for FastQuery, block index and OFPI in a single node. The experimental dataset uses ClimateGZ, which divides the dataset into four different datasets, respectively 3GB small dataset (Small), 30GB medium dataset (Medium), 300GB big dataset (Large) and 3TB massive dataset (Huge). The time to build index under the four datasets of different sizes is shown in Fig.5 It can be analyzed from the experimental results that the block index has the most fastest time of index generation due to traversing all data just record the maximum and minimum, FastQuery appears to be slowest. In Huge data, FastQuery takes more than 12 hours to build index which is too long.

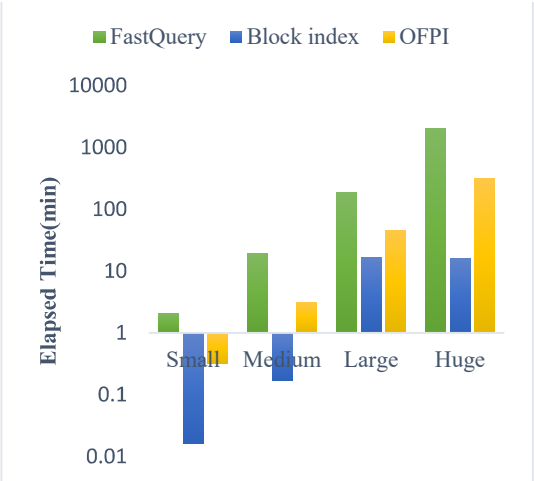
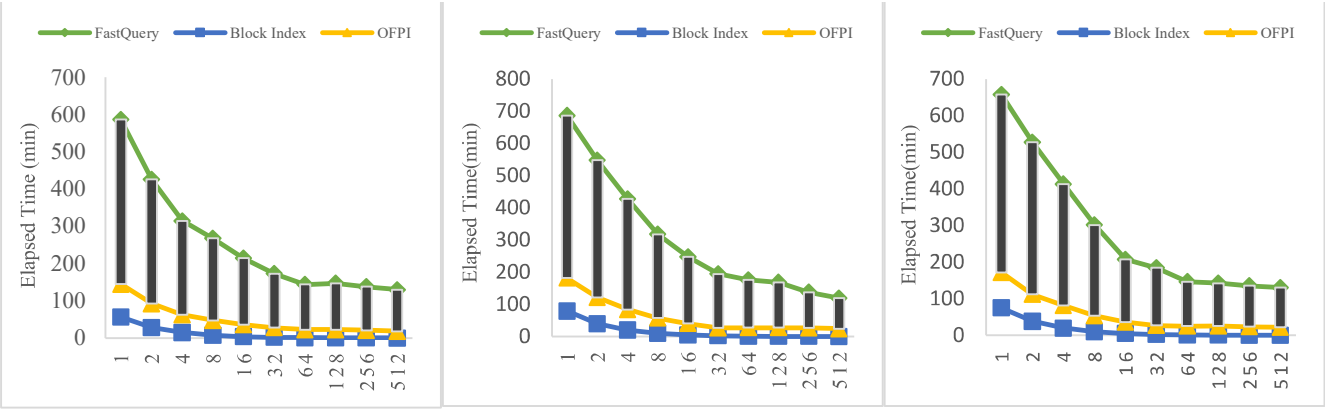


Figure 5 The time of index generation

Index Generation Experiment in multi Nodes: We scale up to 512 nodes for testing. As can be seen from Fig.6, OPFI index creation time is far less than FastQuery. Although the Block index is extremely short, it is a coarse-grained index. With the degree of parallelism increasing, index generation time of OFPI can be significantly reduced.



(a) ClimateGZ (b)VPDS (c) COSM

Figure 6 OFPI, FastQuery and Block index for index generation time comparison on three different datasets.

Index size is also an significant important indicator of index generation method. According to the variance, we set 5%, 15%, 25% data to build BitmapIndex. Table 1 below shows the index generation size of the three index generation methods under four different datasets. The size of the block index is the smallest, FastQuery is the largest.

Table 1 The size of index generation

	Small	Medium	Large	Huge
FastQuery	236MB	2.1GB	17.6GB	169.7GB
Block Index	0.57KB	5.6KB	57.2KB	579KB
OFPI (5%)	12.41MB	105.75MB	985.34MB	8.49GB
OFPI (15%)	35.7MB	317.3MB	290.67MB	26.17GB
OFPI (25%)	59.72MB	526.4MB	4.14GB	42.5GB

In short, OFPI is better than FastQuery in aspect of index generation speed and index size in a single node. Building massive index tasks in a single node takes too long. We need a multi-node, multi-core technology to accelerate the build index process. OPFI builds fine-grained bitmap index for 15% of data. We evaluate the time of index generation about FastQuery, Blockindex and OPFI methods on three different datasets. The results show that the OPFI method is much better than FastQuery. our performance is relatively stable. FastQuery builds bitmap index for all data. When traversing all data, the cardinality in the bitmap is relatively high. OPFI only builds a Bitmap index for a small amount of data, and establishes a bitmap index in a small data range with a relatively low cardinality. Block index creation only needs to traverse all data, record the maximum value and minimum value, so the generation speed is faster than OFPI, and the generation size is smaller than OFPI. However, Block index only support coarse-grained query.

B. ISPI Index Generation Experiments

In-situ Kernel and Node parallel Experiment: This section evaluates the in-situ index kernel generation method and the in-situ index node generation method. We use PiBase simulation software to generate data in real time. The software can control the amount of data generated. This experiment was tested on 48 nodes and experimental data use 3G, 8G, 32G, 128G, 512G and 2.048TB three-component particle data. The experimental results are shown in the Fig.7

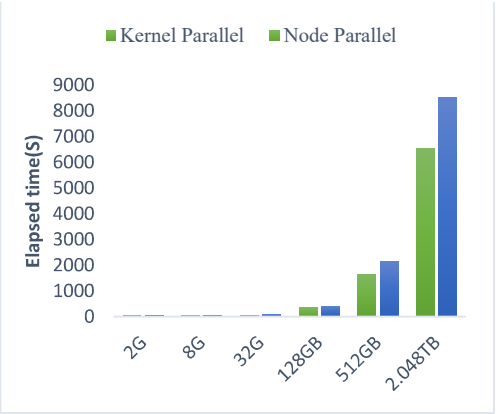
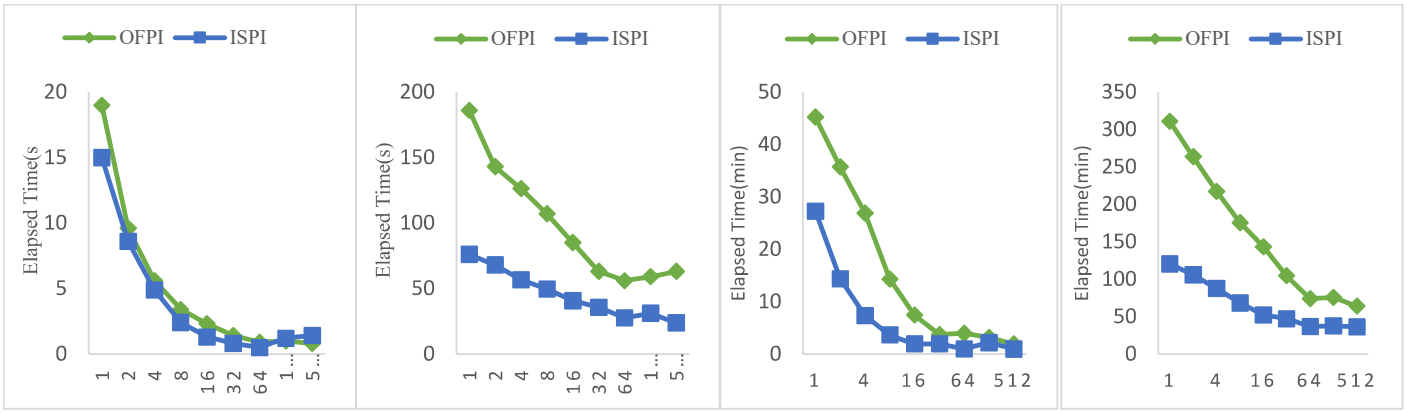


Figure 7 Kernel and Node parallel Comparisons

When the dataset is small, the time of the kernel and node parallelism are not much different. However, as the dataset size increases, kernel parallel method is much faster than node parallel. Because traditional in-situ index generation parallel between nodes. After a node generates data, it needs to send data to other nodes for index generation through MPI. When the amount of data is small, the Tianhe-express inline network can transmit at high speed, so communication overhead has little effect. With the increasing datasets, the impact of communication overhead is getting severe. Besides, The process of synchronous data writing is also getting worse. Kernel parallel method only has 8 cores for indexing but node parallel method has 24 cores for indexing. The probability of write conflict in the kernel parallel method is relatively small and the write conflict field greatly reduced.

ISPI and OSPI Contrast Experiment: In-situ indexing method is extreme important in data query technology. Next we conduct experimental evaluation of in-situ index generation method and offline index generation method. In order to maintain the unity of the dataset, we use Pixie simulation data. We also generate four sizes of datasets, which are 3GB small



(a) Small (b) Medium (c) Large (d) Huge

Figure 8 The time of index generation of OFPI and ISPI

dataset (Small), 30GB medium dataset (Medium), 300GB large dataset (Large), and 3TB massive dataset (Huge). In the comparative experiment, we ensure that the number of index calculation cores of ISPI is the same as the number of cores in the OFPI. The test results are shown as Fig.8.

When the amount of data is small, there is little difference between OFPI and ISPI performance. Because Tianhe-2A equipped with strong computing power and its peak-to-peak bandwidth can reach 500GB/s. In the OFPI, the real-time generated data must be accumulated to the smallest data block(64MB) before index construction in parallel. Meanwhile, the 3GB data can be read into memory at a time. The time of read is short. With the increasing data, datasets need to be read into the memory multiple times in batches. I/O overhead for reading offline data is gradually increasing. Taking the Huge dataset as an example, a single data read requires 67 I/Os. In the process of node expansion, multiple nodes can read data in parallel on the Lustre parallel file system. But when writing data indexes, fine-grained Bitmap indexes will become bottlenecks. In the ISPI, a part of the cores in each node are used as calculations, and a part of the cores are used as calculation indexes and write indexes. Even if the size of the dataset increases, the calculation time of the scientific task can be paralleled with the index calculation, and the write index waiting time will also decrease. Therefore, the ISPI is superior to OFPI.

In conclusion, In-situ kernel parallel method is better than node parallel owing to less data movement overhead and less write synchronization wait time. For real-time generated data, we should give priority to ISPI because it can reduce I/O access overhead.

C. *HPR Comparative Experiments*

Data retrieval performance is an considerable important indicator for evaluating query performance. This section mainly conducts throughput comparison experiments and retrieval time comparison experiments. We evaluate FastQuery, block index, and HPR. We analyze three retrieval methods I/O throughput and query data time by searching a certain portion of data. This experiment was conducted on 64 nodes. The size of the dataset is 900GB. The throughput of the three retrieval methods is shown in Table 2.

Table 2 Retrieve I/O throughput comparison

Datasets	FastQuery	Block Index	HPR
ClimateGZ	103.4MB/s	14.5GB/s	7.6GB/s
VPDS	5.9GB/s	432.7GB/s	216.5GB/s
COSM	431.7MB/s	84.3GB/s	49.3GB/s

From the throughput of the three scientific datasets, the throughput of VPDS is nearly 7 times that of ClimateGZ, and the throughput of ClimateGZ is the smallest. Because each particle is associated with seven one-dimensional variables during data retrieval in VPDS. However, only the temperature in the ClimateGZ is queried, and the rest of the data in the scientific dataset does not need to be read into the memory. The throughput of the block index and HPR is large because the block index is continuously read in units of data blocks each time.

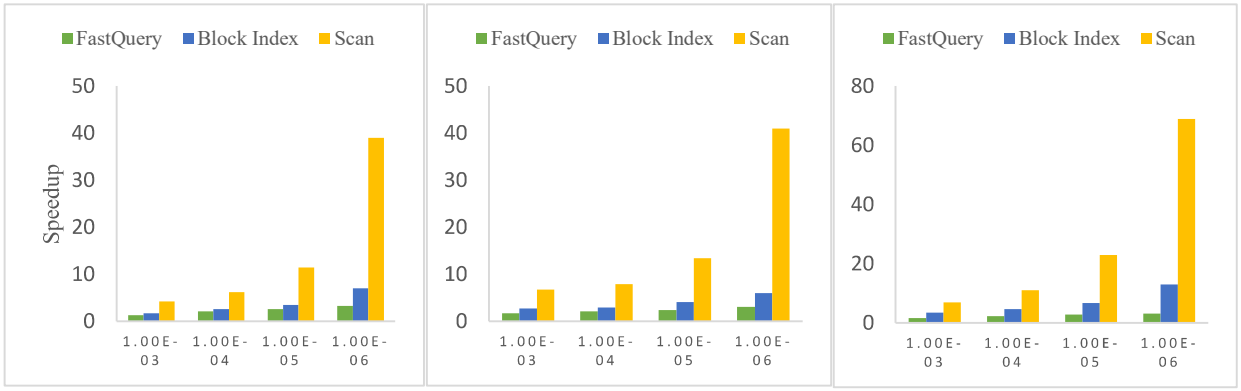
According to the retrieval constraints, we set the retrieval data proportion as 10^{-3},10^{-4},10^{-5},10^{-6}. The metric measured by the retrieval time is I/O throughput divided by the retrieval time. The more I/O throughput, the shorter the retrieval time, the more efficient the retrieval. We name the metric absolute throughput, expressed by the symbol R_{out} , $Q_{I/O}$ is I/O throughput and T_{query} is the retrieval time. The formula is as follows.

$$R_{out} = Q_{I/O} / T_{query} \quad (1)$$

As can be seen from Fig.8, the performance of HPR is the best. Compared to the block index retrieval method, the time spent on coarse-grained queries is the same. But when the data is read into memory, the block index needs to scan all the records to get the data within the retrieval constraint. HPR can return records directly based on the fine-grained Bitmap index constraints. The smaller the portion of data selected, the better performance than block index. For FastQuery, its absolute throughput increases significantly as the proportion of query selection decreases. When the query data ratio is 10^{-6}, under the ClimateGZ dataset, the absolute throughput can approach or even exceed of the block index, but still slower than the high performance parallel retrieval method.

(a) VPDS (b) COMS (c) ClimateGZ

Figure 8 The absolute throughput comparison of FastQuery, Block block technology and HPR



(a) VPDS (b) COMS (c) ClimateGZ

Figure 9 The speedup comparison of FastQuery, Block index technology and HPR

We analyze the performance of HPR by speedup. As can be seen from Fig.9. HPR can achieve an acceleration ratio of more than 3 times when the data of 10^{-6} is retrieved in three different scientific data sets. Compared with block technology, the highest acceleration can be achieved by 13 times, which can be increased by an order of magnitude. Compared to FastQuery, a maximum of 3.3x acceleration can be achieved. An acceleration ratio of 4.2 to 69 times can be achieved with respect to the scanning load.

D. The Scalability Experiments among OFPI,ISPI and HPR

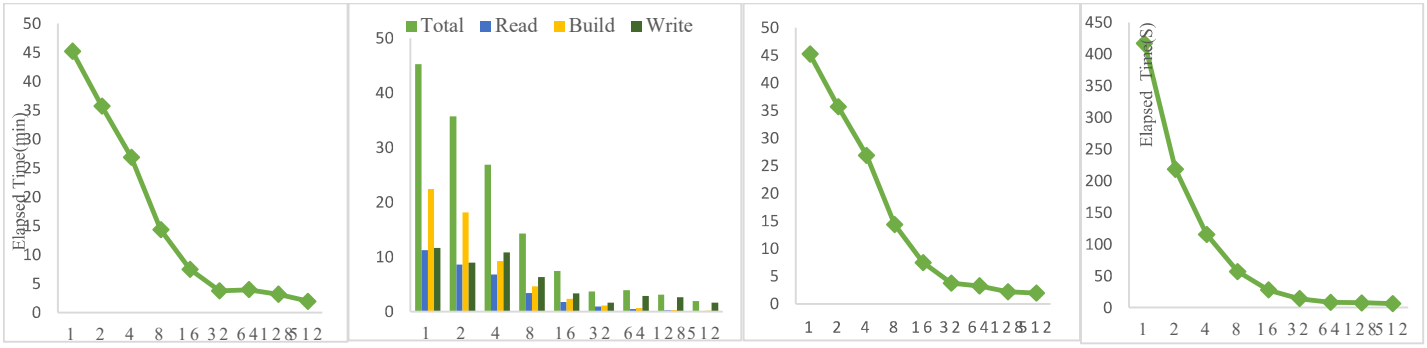
Scalability is an important indicator to measure the robustness of a framework. The OFPI, ISPI and HPR method are key methods of Bi-Cluster query framework. So we conduct the experiments to analyze the scalability performance.

OFPI Scalability Experiment: The strong scalability experiment used the ClimateGZ scale of 300G, the maximum expansion of 512 nodes, testing on 12288 cores. The result are shown in Fig.10(a).

As can be seen from the figure, before the 64 nodes, as the number of nodes increases, the index construction is getting shorter and shorter. When the node reaches 64, the time spent is increased compared to 32 nodes. As the nodes continue to increase, the build time is hardly reduced. In order to analyze the reason, we break down the time of index generation. The total time is decomposed to read data, build index, and write index. This result is shown in the Fig.10(b).

Both read data time and build index time support linear expansion. With the increasing nodes, the elapsed time is becoming shorter and shorter. However, the process of write fine-grained BitmapIndex take a long time at 64 nodes. As nodes continue to increase, the proportion of write indexes to the entire time is increasing. Because the write index needs to be synchronized. When a bitmap index block with high cardinality is being written, the already calculated node must be queued for it to complete the write task. As the number of cores increases, the probability of write waiting increases.

ISPI Scalability Experiment: We need to analyze the data generated in real time, so we use PiBase Simulation software. Generate 300GB of data through software, test extension to 512 nodes. The experiment result as Fig.10(c). From the experimental results, as the number of nodes increases, the index construction time decreases. When the number of nodes reaches 64, the write index process also needs to wait. However, in the in-situ indexing process, the number of cores for each build index is fixed at 8, and the collision domain waiting for the write index and the number of write waiting is relatively small. Although the number of nodes increases, the time to complete the index will be reduced. Therefore, the scalability of ISPI is better than OFPI.



(a) OFPI (b) Time breakdown of OFPI (c) ISPI (d) HPR

Figure 10 The scalability experiments about OFPI, ISPI and HPR

HPR Scalability Experiment: We retrieve the 10^{-6} data of ClimateGZ as the load. As can be seen from the Fig10(d), HPR supports strong scalability. Each node in the process of retrieving data is independent, so as the number of nodes increases, the retrieval time will be shorter and shorter.

V. Conclustion

We propose a high-performance query data framework for scientific data. In terms of index generation, a two-tier index data structure is designed to build index in parallel. Meanwhile, an in-situ kernel parallel index strategy is proposed to build index for online real-time generated data. With regard to data retrieval, a two-tier parallel query mechanism is designed to efficiently read data, and a dynamic union read strategy and an adaptive scheduling strategy are used to optimize the data retrieval process.

The Bi-Cluster Framework is evaluated on scientific datasets, which proves that our design achieves good performance. The size of index generation is much smaller than FastQuery. HPR is superior to the FastQuery and block index technique. Compared with the scan method, the maximum speedup of 69 can be achieved. Compared with the block index technique, the maximum speedup can be achieved by 13 times, and the FastQuery can achieve a maximum speedup of 3.3 times. But for now, our method is only efficient under certain rules. In the future research, we will analyze the characteristics of the data and realize a general high-performance data query framework.

Acknowledgment

Supported by National Key R&D Program of China 2017YFB0202201, Program for Guangdong Introducing Innovative and Enterpreneurial Teams under Grant NO. 2016ZT06D211 and the Natural Science Foundation of China under Grant NO.U1811464

Reference

[1] Bowers K J , Albright B J , Yin L , et al. Ultrahigh performance three-dimensional electromagnetic relativistic kinetic plasma simulation[J]. Physics of Plasmas, 2008, 15(5):199-434.

[2] Stephens Z D , Lee S Y , Faghri F , et al. Big data: Astronomical or genomical?[J]. Plos Biology, 2015, 13(7):e1002195.

[3] A. Shoshani and D. Rotem, editors. Scientific Data Management: Challeng es, Technology, and Deployment, chapter 6. Chapman & Hall/CRC Press, 2010.

[4] Hey T . The Fourth Paradigm – Data-Intensive Scientific Discovery[M]//E Science and Information Management. Springer Berlin Heidelberg, 2012.

[5] Jha S , Qiu J , Luckow A , et al. A Tale of Two Data-Intensive Paradigms: Applications, Abstractions, and Architectures[J]. 2014.

[6] T. Hey, S. Tansley, and K. Tolle, editors. The Fourth Paradigm: DataIntensive Scientific Discovery. Microsoft, Oct. 2009.

[7] Liao X , Xiao L , Yang C , et al. MilkyWay-2 supercomputer: system and application[J]. Frontiers of Computer Science, 2014, 8(3):345-356.

[8] Fu H , Liao J , Yang J , et al. The Sunway TaihuLight supercomputer: system and applications[J]. Science China Information Sciences, 2016, 59(7).

[9] Mozafari B, Curino C, Jindal A, et al. Performance and resource modeling in highly-concurrent OLTP workloads[C]. international conference on management of data, 2013: 301-312.

[10] Aguilera M K, Golab W M, Shah M A, et al. A practical scalable distribu ted B-tree[J]. very large data bases, 2008, 1(1): 598-609.

[11] Michael Stonebraker, Jacek Becla, David Dewitt, et al. Requirements for science data bases and scidb. In Conference on Innovative Data Systems Research, 2009.

[12] Chou J , Wu K , Prabhat. FastQuery: A Parallel Indexing System for Scientific Data[C]// 2011 IEEE International Conference on Cluster Computing. IEEE Computer Society, 2011.

[13] Wu K , Shoshani A , Stockinger K . Analyses of Multi-Level and Multi-Component Compressed Bitmap Indexes[M]. ACM, 2010.

[14] Wu K, Stockinger K, Shoshani A, et al. Breaking the Curse of Cardinality on Bitmap Indexes[J]. statistical and scientific database management, 2008: 348-365.

[15] Cheng L , Wang Y , Pei Y , et al. A Coflow-Based Co-Optimization Framework for High-Performance Data Analytics[C]// Proc. 46th International Conference on Parallel Processing. IEEE, 2017..

[16] Wu T , Shyng H , Chou J , et al. Indexing Blocks to Reduce Space and Time Requirements for Searching Large Data Files[C]// IEEE/ACM Inte rnational Symposium on Cluster. IEEE, 2016.

[17] Wu K, Ahern S, Bethel E W, et al. FastBit: interactively searching massive data[J]. Lawrence Berkeley National Laboratory, 2009, 180(1).

[18] Wu K, Otoo E J, Shoshani A, et al. Optimizing bitmap indices with efficient compression[J]. ACM Transactions on Database Systems, 2006, 31(1): 1-38.

[19] S. Klasky, H. Abbasi, et al. In Situ Data Processing for Extreme-Scale Computing. In SciDAC, July 2011.

[20] Wu T , Chou J , Podhorszki N , et al. Apply Block Index Technique to Scientific Data Analysis and I/O Systems[C]// IEEE/ACM International Symposium on Cluster. IEEE, 2017.

[21] B. Behzad, H. V. T. Luu, J. Huchete, S. Byna, Prabhat, R. Aydt, Q. Koziol, and M. Snir. Taming parallel i/o complexity with auto-tuning. In SC, pages 68:1–68:12, 2013

[22] S. Lakshminarasimhan, N. Shah, S. Ethier, S. Klasky, R. Latham, R. Ross, and N. F. Samatova. Compressing the Incompressible with ISABELA: In-situ Reduction of Spatio-temporal Data. In Euro-Par, pages 366–379, 2011.