Big Data Research ••• (••••) •••–•••

Contents lists available at ScienceDirect

Big Data Research

www.elsevier.com/locate/bdr

FlexAnalytics: A Flexible Data Analytics Framework for Big Data Applications with I/O Performance Improvement ☆

Hongbo Zou [a,*], Yongen Yu [b], Wei Tang [c], Hsuan-Wei Michelle Chen [d]

[a] Queensland University of Technology, Brisbane, Qld 4001, Australia
[b] Illinois Institute of Technology, Chicago, IL 60616, USA
[c] Argonne National Laboratory, Argonne, IL 60439, USA
[d] San Jose State University, San Jose, CA 95192, USA

ARTICLE INFO

Article history:
Available online xxxx

Keywords:
I/O bottlenecks
In-situ analytics
Data preparation
Big data
High-end computing

ABSTRACT

Increasingly larger scale applications are generating an unprecedented amount of data. However, the increasing gap between computation and I/O capacity on High End Computing machines makes a severe bottleneck for data analysis. Instead of moving data from its source to the output storage, in-situ analytics processes output data while simulations are running. However, in-situ data analysis incurs much more computing resource contentions with simulations. Such contentions severely damage the performance of simulation on HPE. Since different data processing strategies have different impact on performance and cost, there is a consequent need for flexibility in the location of data analytics. In this paper, we explore and analyze several potential data-analytics placement strategies along the I/O path. To find out the best strategy to reduce data movement in given situation, we propose a flexible data analytics (FlexAnalytics) framework in this paper. Based on this framework, a FlexAnalytics prototype system is developed for analytics placement. FlexAnalytics system enhances the scalability and flexibility of current I/O stack on HEC platforms and is useful for data pre-processing, runtime data analysis and visualization, as well as for large-scale data transfer. Two use cases – scientific data compression and remote visualization – have been applied in the study to verify the performance of FlexAnalytics. Experimental results demonstrate that FlexAnalytics framework increases data transition bandwidth and improves the application end-to-end transfer performance.

1. Introduction

Big data applications running on High-End Computing (HEC) machines are generating large volumes of data in a single execution, and these data volumes are only expected to increase in future. Since this massive data is key to scientific discovery, the ability to rapidly store, move, analyze, and visualize data is critical for scientists' productivity. Yet the growth of data volume imposes numerous requirements on I/O performance, which results in I/O bottlenecks in current HEC machines. Furthermore, the enlarged gap between computational power and I/O performance further worsens I/O performance. These two folds combined together result in undesirable situations, where I/O bottlenecks devastate the efficiency and scaling of scientific simulations and associated data analyses and visualizations. Scientists are forced to wait a substantial portion of the simulation runtime writing data to the storage [15] or simply forgo writing out data in order to keep total I/O within reasonable bounds. These trends will be speeded up in the planning for exascale computing platforms in which the attainable I/O performance is further exacerbated by increased contention on shared resources [30,31,38] on those platforms.

To improve I/O performance, in-situ data analytics has emerged as an effective way to substitute for the traditional offline data analytics and overcome the increasingly severe I/O bottleneck for scientific applications running at the petascale and beyond. Through processing data before placing it on disk, in-situ analytics can reduce I/O costs (both in time and in power), extract and deliver valuable insights from live simulation output in time, and gain improved end-to-end performance. The utility of in-situ processing is demonstrated by its wide use by leading scientific applications, like the Metagenomics [21], the Cosmology Simulation [26,27], and the Maya [19]. Furthermore and enabled by standard parallel I/O interfaces like ADIOS and MPI-IO [5], there are emerging infrastructures that support online data analytics, including FastBit [28], FlexQuery [35], and others [7,20,33].

☆ This article belongs to Scalable Computing for Big Data.

* Corresponding author.
E-mail address: hongbo.zou@student.qut.edu.au (H. Zou).

http://dx.doi.org/10.1016/j.bdr.2014.07.001
2214-5796/

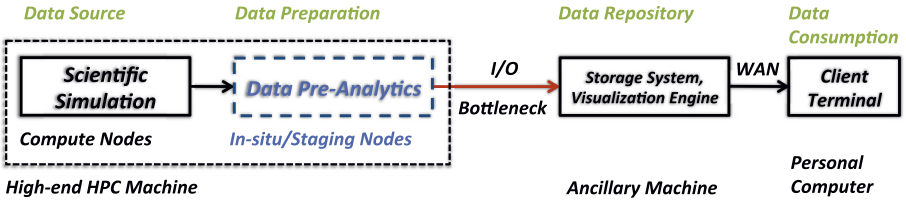

Fig. 1. Scientific data analytics system architecture overview.

Speeding up data processing to keep pace with simulation pushes the in-situ analytics to be separated from simulation and parallelly executed on independent computational resources, such as shared compute cores (helper-core), separate nodes dedicated to analytics (termed 'staging nodes'), or I/O nodes. The separated locations of placing simulation and data analytics, however, introduce the extra data movement along the I/O path. The challenge becomes how to balance the conflict between parallelly using limited and costly computational resources and the extra data movement cost. Fig. 1 depicts a typical scientific data analytics architecture. It is comprised of the scientific simulation and its associated data pre-analytics functions for processing simulation results. The simulation is run on some high end HPC system, and data visualization is performed on the visualization cluster. The output of analysis or visualization directly stores in storage system for science users access through Wide Area Networks (WAN) connection. Our study takes the further step of 'augmenting' the data analytics to any data processing techniques (such as data compression, query) for data reduction purpose.

Data compression has been proven to be a very effective data analytics method in improving end-to-end performance in the wide-area networking domain, in which the large size data are compressed and transferred with the worst networks bandwidth [14,36,39]. In HEC domain, the applicability of data compression has been studied and applied to many [22,34,36]. In addition, exploration and visualization of scientific data often are described as 'visualization queries' posed on simulation outputs. Typical queries include point selection, subsetting, cutting planes, global data summaries, and feature extraction. Such queries often return only subsets of the original data and can significantly reduce the amounts of data that have to be moved and/or further processed for display. Modern visualization systems exploit early data reduction within their visualization engine to optimize visualization performance [35]. Examples include 'contracts' in VisIt [24], 'events' in VTK [23], and 'rendering information' in ParaView [7]. So, data compression and visualization query can be considered as two kinds of effective in-situ data analytics for data reduction.

This paper presents a detailed assessment of using data compression and visualization query to reduce data movement cost along I/O path, especially with flexible placement strategies. Because the data compression and visualization query incur computational resource contention with simulation, such analytics servicing for data reduction should be carefully studied and devised. Understanding when, where, and how to use such data analytics in I/O path so as to improve the end-to-end application performance is the major goal of this paper. In this paper, we make the following contributions: (1) We present a latency based quantitative model to dynamically make a decision whether should reduce data for next step and what data analytics should be selected to apply on data with the comprehensive consideration on the cost of both I/O and computational resource. (2) We analyze the end-to-end latency of four potential analytics placement strategies with the above quantitative model. (3) We propose a flexible analytics framework and implement a prototype system for such framework. (4) We conduct experiments to show that flexibly choosing the best data movement mode in various resource changes can improve total execution time compared to the execution time without this service.

The reminder of the paper is organized as follows. Section 2 presents background information about data compression, visualization query, and in-situ analytics. Section 3 describes the analytics quantitative model and latency analysis of four placement strategies with model. Section 4 is the implementations of flexible analytics placement system with the integration with I/O middleware. Section 5 presents flexible analytics system built for the GKW data and the measurements of performance and cost. Parallel analytics is checked on this section for performance optimization. Section 6 reviews related work, and Section 7 concludes the paper.

2. Background

To develop a flexible data analytics framework, compression methods, data visualization query, and data analytics placement strategies are briefly introduced here. Data compression and visualization query are considered as two common data analytics to reduce data here. Data analytics placement strategies lists four potential placement strategies for the typical scientific data analytics architecture.

2.1. Data compression

Data compression aims to reduce redundant information in data to save the consumption of resources such as storage space or I/O bandwidth, and therefore this technique has very important application in the areas of data storage and transmission [6]. Apart from the space savings, data compression provides another benefit that compressed data could be stored and transmitted faster, thereby improving the performance of I/O intensive workloads. However, compression is a compute-intensive operation and imposes an extra processing cost and resource contention on compute node [14]. Therefore, the design and choosing of data compression algorithms involve trade-offs among various factors, including the degree of compression, the amount of distortion introduced (using in lossy data compression), and the computational resources required to compress and decompress the data [3]. To facilitate the following discussion in the paper, a brief introduction will be given about lossy and lossless compression algorithms.

Lossless data compression algorithms usually exploit statistical redundancy to represent data more concisely without losing information, and can reconstruct the original data exactly from the compressed data [17]. There are two key metrics – compression ratio and (de)compression speed that are used to evaluate the performance of lossless data compression algorithms. Different compression algorithm has different compression ratio. Even the same compression algorithm could have varying compression ratio with different original data format. In general, the compression ratio of text format data is higher than binary format, and the compression ratio of repetitive data is higher than random data. Processor power has direct relationship with compression speed. The compression speed is also affected by compression buffer, but we don't want to extend this discussion at here.

Lossy data compression is contrasted with lossless compression and can reconstruct an approximation of the original data. In these

Please cite this article in press as: H. Zou et al., FlexAnalytics: A Flexible Data Analytics Framework for Big Data Applications with I/O Performance Improvement, Big Data Research (2014), http://dx.doi.org/10.1016/j.bdr.2014.07.001

Table 1
Data reduction queries.

Query	Type	Description
Max, Min, Average, Sum, Stdev	Statistics	Max/Min/Average/Sum/Stdev (var_double_array)
Range	Similarity	Range(variable, dimensions, start_pos, end_pos)
Select, Lineout	Visualization	Select(variable, threshold1, threshold2) Lineout(variable, start_pos, end_pos)
Expression	Combination	Self-defined function, select(lineout(variables, start_pos, end_pos), threshold1, threshold2)

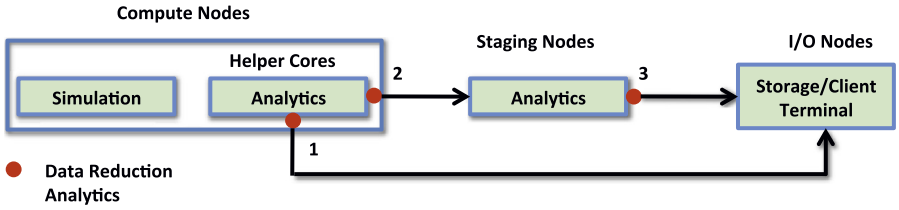

Fig. 2. Potential locations of reduction analytics located in I/O path.

algorithms, some loss of information is acceptable [10]. The lossy data compression is also could be evaluated by compression ratio and (de)compression speed. Compression accuracy is a new metric involved by lossy compression. Accuracy is used to judge the difference between original data and decompressed data. The inherited deficiency of information loss limits the application area of lossy compression on lossless requirement. This paper doesn't explore compression algorithms in depth. We use a lossy compression developed by [11] to evaluate our flexible analytics framework.

2.2. Data visualization query

Exploration and visualization of scientific data may be described as 'data queries' posed on simulation outputs. Typical queries include point selection, subsetting, cutting planes, global data summaries, and feature extraction. Such queries often return only subsets of the original data and can significantly reduce the amounts of data that have to be moved and/or further processed for display. Modern visualization systems exploit early data reduction within their visualization engine to optimize visualization performance [35]. Examples include 'contracts' in VisIt [24], 'events' in VTK [23], and 'rendering information' in ParaView [7].

In this study, adaptive data reduction uses only four types of data reduction queries, as shown in Table 1. Statistics queries calculate the features of a given variable. Similarity queries are used to filter out the values with close features. Visualization queries are similar to the former in terns of implementation, but the specified features are those relevant for visualization. Combination queries provide a simple pipeline method to help users combine basic queries to construct their won data requests. These four query categories have been widely adopted by the visualization community and provide the basic quantitative analysis for data visualization [35].

2.3. Data analytics placement strategies

To explore the possible places on where data analytics can be placed in scientific data analytics architecture, analytics placement strategies is introduced here. In general, data analytics can be simply classified into four types according to execution time and location [32].

(1) Inline Processing: analysis/visualization routines are synchronously performed by the simulation. ParaView [7], VisIt [24] and other in-situ visualization works [35] fall into this category. Inline processing is easy to code and implement by scientists. The sequential execution of simulation and analytics introduces extra data waiting time for both of their execution.
(2) Helper-core Processing: some cores on the compute nodes are dedicated to perform select analysis actions. Examples include Functional Partitioning [12] and Software Accelerator. Helper cores parallelly analyze and reduce data immediately after data produced. It involves some resource contention, such as memory, to simulation; In (1) and (2), data compression and analytics services share compute nodes with simulation. The compressed data is prepared for the further processing or storage saving.
(3) Dedicated-nodes Processing: the analytics are executed on non-simulation nodes, such as staging area and active storage [16]. Remote Processing needs additional computational resource for analytics and incurs the extra data movement cost.
(4) Offline Processing: data is read back for analysis [8] after written to storage. Offline processing can statically provide enough resource and tools to analyze data in details.

Any one of the above four placement strategies can be selected and executed independently. Or, some of placement strategies can be combined together and executed hybrid data processing strategy. The details will be extended in Section 3 according to our flexible data analytics framework design requirements.

3. Methodology

HEC machine is a complex system, which generally consists with three components – compute nodes, analytics nodes, and storage. To reduce I/O overheads on applications, some HEC machines allocate part of compute nodes to conduct staging functionality, called staging nodes. Data analytics can be placed and executed on staging nodes to reduce data before writing data into storage.

3.1. Flexible analytics placements

We introduce data analytics into peta-scale computing to reduce the data movement. To achieve this goal, we investigate where data analytics can be placed among I/O path. In general, a scientific application includes two parts – simulation and analytics, some application including visualization. Simulation and inline analytics is executed on compute node after an application submission. Among application execution, the output data is dumped to I/O nodes for storage saving. As shown in Fig. 2, data analytics can be placed at point 1 to reduce the output dumping to storage.

If data analytics is placed on staging nodes, there are numerous data movements produced between compute nodes to staging

Please cite this article in press as: H. Zou et al., FlexAnalytics: A Flexible Data Analytics Framework for Big Data Applications with I/O Performance Improvement, Big Data Research (2014), http://dx.doi.org/10.1016/j.bdr.2014.07.001

Table 2
Notations in performance quantitative model.

Notation	Description
$\rho^1 = \rho_c$	Available Processor Cycles Ratio (<1, sharing processor with other processes, >1 using multiple processors to parallel execution). ρ_c is available cycles ratio on compression side. And ρ_d is available cycles ratio on decompression side.
ρ_d	
BW	Available Data Transfer Bandwidth
B_{data}	The Size of Data Block
T_c, T_d	Compression/Decompression Throughput
δ	Compression Ratio (compressed/original)
$\varepsilon^1 = \varepsilon_c$	The Impact Factor of Cache. ε_c is impact factor in compression side. ε_d is impact factor in decompression side.
ε_d	
$t_{original}$	Total end-to-end latency to original (uncompressed) data transfer
$t_{compression}$	Time to compress the data (algorithm dependent)
$t_{decompression}$	Time to decompress the data (algorithm dependent)
$t_{transfer}$	Transfer time
t_{total}	Total end-to-end data transfer latency

nodes. Point 2 in Fig. 2 shows the potential position to place analytics to reduce the data size moving from compute nodes to staging nodes. Such case is the primary study case in the paper. After the data analysis, the reduced data needs to be written into storage for saving. In Fig. 2, point 3 shows that the data analytics is used to reduce movement cost between staging nodes and I/O nodes.

The listed three points in Fig. 2 show the potential places for data analytics. Adaptive placement algorithm will find out the best place to apply the analytics to the data and reduce the movement cost.

3.2. Analytics algorithms quantitative model

To evaluate and compare the performance of different analytics method and placement strategies, we quantitate the analytics algorithms in this section. Because data queries are explicitly requested by client terminal with special use case, we only show the compression quantitative model for general data reduction request here. A set of performance and cost metrics are defined for quantitative model description. We then derive a simple performance model to compare the different compression algorithms. The model has been developed and successfully applied in practice to implement adaptive compression choosing.

Table 2 lists the notations used in compression performance quantitative model. Every notation is described and simply explained in the table. However, there are still two notations, ρ and ε, need to be explained here. ρ notates the available processor cycles ratio for compression computation. Because compression is a computation-intensive task, the processor idle cycles ratio indicates how many available processor cycles could be used for compression. $\rho = 1$ when a special core is exclusively used by compression process. Parallel compression is executed on multiple processors if $\rho > 1$. In addition, ε stands for the impact factor related cache. Compression process needs to swap data from memory for compression; therefore the available size of cache space directly impacts compression performance. $\varepsilon = 1$, when there is no other processes share multiple level caches with compression process. To simplify our discussion, the impact factor of cache is set as 1 ($\varepsilon = 1$) in the paper. The following equations calculate the cost (latency) of data transfer from sender to receiver when compression is running.[1]

Eq. (1) gives the end-to-end original data transfer latency on networking with available bandwidth BW. Eqs. (2) and (4) give compression and decompression time on sender and receiver respectively. Eq. (3) gives the transfer time of compressed data from sender to receiver. Eq. (5) shows the total cost.

$$t_{original} = B_{data}/BW; \tag{1}$$

$$t_{compression} = B_{data}/(T_c * \rho * \varepsilon); \tag{2}$$

$$t_{transfer} = (B_{data} * \delta)/BW; \tag{3}$$

$$t_{decompression} = B_{data}/(T_d * \rho * \varepsilon); \tag{4}$$

$$t_{total} = t_{compression} + t_{transfer} + t_{decompression}; \tag{5}$$

With the above quantitative analysis, the data transfer costs can be compared with/without compression under the selected compression algorithm. Therefore, the compression decision can be made by the *If* statement as follows:

```
If (t_original > t_total)
{
    Select the algorithm with Min(t_total);
    Compress with the selected algorithm;
}
Else
    Don't Compress;
```

To implement *If* statement, the quantitative metrics are classified into two categories with different information collecting method:

Profiling Information:
Compression/Decompression Speed (T_c, T_d),
Compression Ratio (δ).
Monitoring Metrics:
Available Processor Cycle Ratio (ρ),
Available Data Transfer Bandwidth (BW),
The Impact Factor of Cache (ε).

3.3. Total end-to-end latency analysis

To explore the resource cost of the analytics, we analyze the total end-to-end latency in all cases. According to the different transfer destination, I/O path are roughly classified into two categories – memory-to-memory transfer and memory-to-storage one. Because the connection bandwidth of two categories is totally different, the analytics cost varies with the available resource. $t_{original} > t_{total}$ gives the condition that compression is selected. The following analysis is based on this condition.

(1) Simulation to Storage (Inline compression): in this case, the data is transferred from computational nodes to the storage. Such case does not include data decompression. Therefore,

$$t_{original} > t_{total} \Rightarrow t_{original} > t_{compression} + t_{transfer}; \quad \text{and,}$$

$$B_{data}/BW > B_{data}/(T_c * \rho * \varepsilon) + (B_{data} * \delta)/BW$$

$$\Rightarrow \quad 1 - \delta > BW/(T_c * \rho * \varepsilon);$$

$$\text{so,} \ T_c * (1 - \delta) * \rho * \varepsilon > BW; \tag{6}$$

In Eq. (6), δ and T_c (compression ratio) are constants if the compression algorithm and input data format don't change. Therefore, keeping ρ and BW in a proper proportion decides the improvement of end-to-end latency. ρ can be greatly increased with the parallel. For data transmission from simulation to storage, although BW is limited by storage, the simulation can provide available processor-cycles for inline compression to offset such limitation. For example, if $BW = 400$ Mb/s,

[1] We give ρ *and* ε in equations to simplify the discussion. And, $\rho_c \rho_d \varepsilon_c$ and ε_d are distinct when they need to be measured.

Please cite this article in press as: H. Zou et al., FlexAnalytics: A Flexible Data Analytics Framework for Big Data Applications with I/O Performance Improvement, Big Data Research (2014), http://dx.doi.org/10.1016/j.bdr.2014.07.001

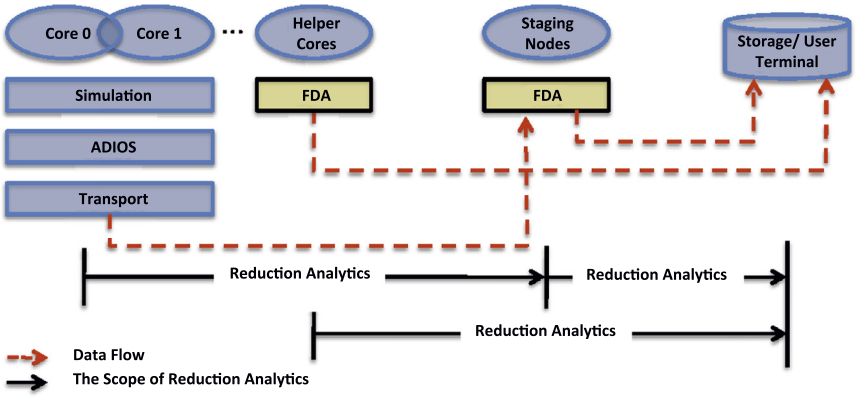

Fig. 3. Flexible data analytics system framework.

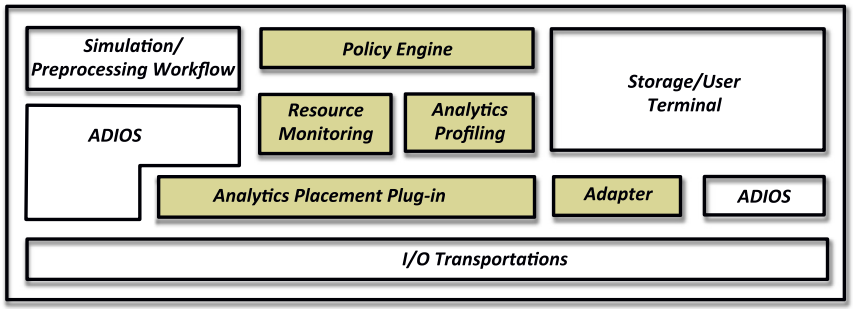

Fig. 4. Flexible data analytics software stack.

$T_c = 100$ Mb/s, $\delta = 50\%$, and $\varepsilon = 1$, ρ should be equal or larger to 4. $BW = 400$ Mb/s is very common configuration in a supercomputer.

(2) Simulations to Staging (Inline compression): in this case, the data is transferred from computational nodes to staging nodes. This case includes data decompression in staging nodes. Therefore Eq. (6) becomes

$$(T_c * T_d)/(T_c + T_d) * (1 - \delta) * \rho * \varepsilon > BW; \tag{7}$$

Actually, $T_c < T_d$ (this happens often in compression algorithms), and compression and decompression are executed in parallel with multiple blocks transfer, therefore ρ_c can be calculated with Eq. (6) and ρ_d in decompression side can be derived by

$$\rho_d = T_c * \rho_c / T_d; \tag{8}$$

Eq. (7) gives us the same conclusion with Eq. (6). If the BW is very large between computational nodes and staging nodes with high-speed networking connection, compression becomes an expensive operation in this case. For example, if $BW = 16$ Gb/s (16 384 Mb/s), $T_c = 100$ Mb/s, $T_d = 200$ Mb/s, $\delta = 50\%$, and $\varepsilon = 1$, ρ should be equal or larger to almost 491.520. $\rho \geq 491.520$ stands for that the computational area needs more than 492 processors for data compression. With Eq. (8), there are at least 246 processors to decompression. The values given in example are very common configuration in HEC.

(3) Helper cores to Staging: this case is same with case (2). Cases (2) and (3) stand for high-speed memory-to-memory data transfer. Such available bandwidth needs fast data compression to catch the small end-to-end latency.

(4) Staging to Storage: this case is same with case (1). Cases (1) and (4) stand for memory-to-storage data transfer. Compression becomes very effective method to release the I/O bottleneck in this case.

4. Design and implementation

4.1. Overview

The quantitative model clears that different use case requests different data analytics to reduce data. Many factors, such as transfer bandwidth, available memory, and data format, etc., impact the end-to-end performance. With this conclusion, a flexible data analytics framework is developed. Fig. 3 shows the flexible data analytics system framework (FlexAnalytics). Such framework can dynamically deploy data analytics on the I/O path to overcome the bandwidth limitations between data source (simulation) and sink (storage/user terminal). FlexAnalytics includes five components: Policy Engine, Resource Monitoring, Analytics Profiling, Analytics Placement Plug-in, and Adapter. Fig. 4 shows the FlexAnalytics software stack. FlexAnalytics clearly separates the simulation, analytics, and storage/user terminal into two parts: the data source, which is the outputs of the simulation performed on the high end HPC system, and the data sinks, which consume the output data with data storing and visualization.

FlexAnalytics leverages the ADIOS [9] parallel I/O library which provides meta-data rich read/write interfaces to simulation and analysis codes. ADIOS has a set of built-in I/O methods under its higher level APIs to support various file I/O methods (such as MPI-IO, HDF5, and NetCDF) as well as data staging methods. ADIOS has been used by several leadership scientific codes and is integrated with popular analysis and visualization tools that include ParaView, VisIt.

4.2. Data compression profiling

We profile end-to-end latency with 222 MB float data transferring between two nodes with three data compression algorithms – lossy [11], bzip2 [1], and gzip [4]. The profiling results can help us to evaluate the selected compression algorithm. Three compression

Please cite this article in press as: H. Zou et al., FlexAnalytics: A Flexible Data Analytics Framework for Big Data Applications with I/O Performance Improvement, Big Data Research (2014), http://dx.doi.org/10.1016/j.bdr.2014.07.001

Table 3
Compression algorithm features profiling for binary data on two nodes measurement.

	lossy	bzip2	gzip
T_c	51.002 MB/s	5.417 MB/s	10.991 MB/s
T_d	242.468 MB/s	8.703 MB/s	23.632 MB/s
δ	23.46%	46.18%	71.75%
ε	Approximate Linear (we generally set $\varepsilon = 1$ as impact factor to eliminate the cache impact with cache exclusively using in this benchmark testing).		

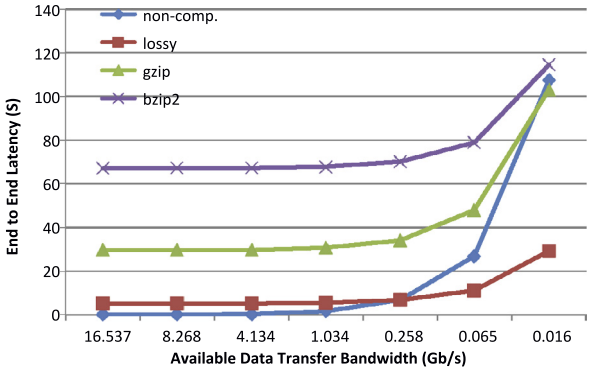

Fig. 5. End-to-end latency comparison with different compression.

algorithms emphasize the different requirements on the compression speed and compression ratio. Bzip2 and gzip are lossless data compression. Bzip2 is an implementation of the Burrows–Wheeler algorithm, which is more effective on data compression ratio. Gzip is based on the DEFLATE algorithm, which provides better compression speed and worse compression ratio than bzip2 [2]. We check the performance of lossy algorithm described in [11]. The lossy compression is better than lossless on speed and ratio. However, information loss constraints its wide use.

The testing nodes are equipped with 2.13 GHz Intel Xeon Processor with 4 MB L3 cache and 12 GB memory. The nodes connected with 16 GB infiniband networks. Table 3 lists the profiling information of three compression algorithms: lossy, bzip2, and gzip. Lossy compression will be introduced in experimental section.

To evaluate the behavior of compression with real-world use cases, we measure end-to-end latency on varying transfer bandwidth. The compression and decompression have been exclusively executed on one single processor. Fig. 5 shows the measurement results. The transfer latency of raw data without compression is shown in figure as baseline.

The results show the (de)compression time of lossless data compression accounts for over 98% of end-to-end latency when the transfer bandwidth is larger than 1.034 Gb/s. For lossy compression, the (de)compression accounts for over 95% of end-to-end latency. If we compress the data in parallel, there are at least 4 processors for lossy compression, 63 processors for gzip, and 74 processors for bzip2 to catch up the data transfer with 1.034 Gb/s bandwidth. When the transfer bandwidth reduces to 0.258 Gb/s, the proportion of (de)compression in latency starts to decrease. The latency of data compression and transfer with gzip is better than non-compression after the available bandwidth less than 0.016 Gb/s. The measured results verify our quantitative model. In our case, all three compressions are not effective method to optimize data transfer when the bandwidth is larger than 264.19 Mb/s. Therefore, the new challenge is how to reduce data (de)compression time to match the high-speed transfer in HEC.

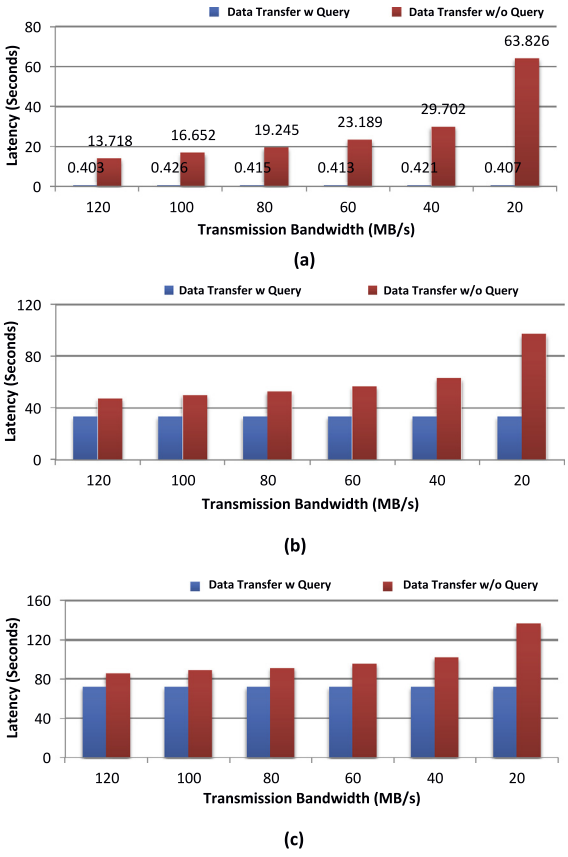

Fig. 6. Data query features profiling with different query types.

4.3. Visualization query profiling

It is similar with compression analytics. We profile end-to-end latency with 222 MB float data transferring between two nodes with three different visualization queries – statistics, similarity, and combination. Three queries have different workload for computing resource requirement. Statistic query(ang_vel) is a lightweight query, which requests the minimal value of angle velocity in the data set. Because the return results are very small data set, the statistic query can significantly increase the transmission bandwidth online. We inject controlled additional network traffic between two nodes variability, resulting in available bandwidth varying from 20 MB/s to 120 MB/s. Fig. 6(a) shows the latency comparison of 222 MB data transfer with and without statistics query. The lightweight query has great data reduction ratio in our profiling. Therefore, the latency is significantly reduced over 150 times when the available bandwidth is less than 20 MB/s.

Figs. 6(b) and (c) show the latency comparison of similarity and combination query on temperature and velocity mesh. These two queries stand for heavy load and joint query, which request much more computing resource for data filtering. So, such queries are good for data reduction with high available transfer bandwidth. 20 MB/s available bandwidth is a special case in the profiling. Although similarity and combination queries are heavy load operations, the low available bandwidth still needs data analytics to improve I/O performance. Because the data query is the operation which is explicitly specified by user terminal, the query cannot be applied on the output data in general cases.

4.4. Flexible analytics placement algorithm

The flexible analytics placement algorithm places the data analytics to the best place, to reduce data movement cost, subject

Please cite this article in press as: H. Zou et al., FlexAnalytics: A Flexible Data Analytics Framework for Big Data Applications with I/O Performance Improvement, Big Data Research (2014), http://dx.doi.org/10.1016/j.bdr.2014.07.001

Table 4
Profiling and monitoring metrics.

Notation	Parameter
Profiling metrics (profiled on idle nodes)	
t_m	Data transfer time per iteration
t_c	Query execution time per iteration
c_p	Processor usage (percentage) for the analytics
m_p	Memory usage (percentage) for the analytics
Monitoring metrics (periodically collected on the candidate place)	
T_c	Data transfer time per iteration
C_p	Query execution time per iteration
M_p	Processor usage (percentage) for the analytics

to resource availability and performance penalties due to the potential need to share the source's computing resources with other simulation.

Placement takes as inputs the potential locations, resource status, and analytics performance profiles. Table 4 shows the profiling and monitoring metrics used as input. Profiling metrics are collected by placing analytics onto some idle node, which provides the policy engine with baseline data about the analytics' running (this should be changed with changed analytics method, especially for data query). For example, if an analytics requires 35% of computing resource for its computation, placement must select a place that is presently utilized less than 65%. Profiling and monitoring are collected on the candidate places for decision-making.

Algorithm 1 operates as follows. When the data is generated by the simulation, an analytics is selected to reduce the data on the idle node, thereby profiling it. Next, the algorithm checks the potential places (as shown in the pseudo-code). Monitoring information $(C_p, M_p)_i$ and the profiling metrics (t_m, t_c, c_p, m_p) are collected by algorithm on all potential places. If the analytics is a compute-intensive task, the algorithm needs to allocate appropriate computational resources to execute the query in parallel (as shown on the algorithm, we currently consider an analytics task to be compute intensive if its processor utilization is 90% or above when it is profiled). A placement decision diverging from the query's initial placement onto an idle node involves analytics relocating. If the analytics is relocated, analytics execution time T_c is again checked on the new place. Placement decisions are re-evaluated at regular time intervals, an example being an analytics relocation from the current (busy) place to the idle place, i.e., if $T_c > t_c + t_m$. The pseudo-code of Algorithm 1 shows the main idle of flexible analytics placement.

4.5. Placement policy engine

The placement policy engine decides where to execute each analytics requested by the terminal adapter. The placement decision is enforced by deploying the analytics code to the best place among the data path. The flexible analytics placement algorithm is performed in the placement policy engine.

5. Experiments and analysis

Experiments with the Gyro-Kinetic Simulation (GKW) [29] application are run on an 80-node Linux cluster. An SGI Altix UV 1000 system is used as ancillary visualization machine for output data analysis. Every node of simulation cluster is equipped with four quad-core 2.0 GHz AMD Opteron processors, 32 GB of memory, a gigabit Ethernet networking connection. The compute node runs CentOS 5.7 OS. GKW is a three-dimensional Particle-In-Cell (PIC) code used to study the microburbulence. SGI Altix system is a shared memory system, which has 1024 2.0 GHz cores (Intel Nehalem EX processors), 4 TB of global shared memory, and

```
Data: Input data per iteration
Input: An analytics
Result: Place the analytics to the best location

  Select the potential locations (N_1, N_2, N_3, ..., N_i)
  Place the analytics on an idle node to profile its performance
    for every iterations do
      collect (C_p, M_p)_i on every potential place N_i;
      Max(CM) = (C_p, M_p)_i; //N_i is the place with maximum resources
      if analytics on the idle node then
        profile (t_m, t_c, c_p, m_p) on the idle node;
        if c_p ≥ 90% || m_p ≥ 90% then
          execute analytics in parallel;
        else
        if Max(CM) > (c_p, m_p) then
          relocate the analytics to the place with Max(CM);
        end
      else
        collect(T_c) on the location where analytics is running;
        if T_c > t_c then
          relocate the analytics to the place with Max(CM);
          collect(T_c) on the new place;
        end
        if T_c > t_c + t_m then
          relocate the analytics back to the previous place;
        end
      end
    end
```

Algorithm 1: Flexible analytics placement.

Table 5
Compression algorithm features profiling for GKW on 80-node cluster.

	lossy	bzip2	gzip
T_c	49.672 MB/s	2.491 MB/s	16.242 MB/s
T_d	819.767 MB/s	6.523 MB/s	108.698 MB/s
δ	24.33%	97.44%	94.08%

8 GPUs in a single system image. The data movement throughput between two machines via shared Lustre file system can reach at most 200 MB/s. GKW simulation outputs particle data containing two particle arrays for zions and electrons respectively. The particle arrays consist of millions of unique double precision values. The values are high degree of randomness and are the most "hard-to-compress" scientific dataset [11].

In our configuration, GKW submits four processes in every node. Every process generates output data for further data processing. Four analytics processes are created to service data transfer in every node. To find out the metrics for analytics selection, compression algorithms are profiled. Table 5 shows the profiled features of compression algorithms in the cluster. All results presented in the paper are averaged with at least three runs.

5.1. Bandwidth analysis from simulation to storage

In the experiment, the compression is deployed on computational side to reduce data size. The available end-to-end bandwidth reaches to 62.74 MB/s with 64 computational nodes. We compare the expected, estimated, and real-tested bandwidth among end-to-end connection. Expected bandwidth is the best one, which considers compress time is very small and negligible. Therefore, $BW_{expected} = BW_{IO}/\delta$. The estimated bandwidth is calculated by Eq. (6) in the paper. Estimated bandwidth shows the maximal optimization of data transfer with parallel compression. Any data transfer bandwidth is smaller than the $\min\{BW_{expected}, BW_{estimated}\}$. The experiment is mapped to two scenarios: simulation to storage and staging to storage. Both of them can be summarized as memory to storage data transfer model. Fig. 7 shows the tested bandwidths with/without compression.

Please cite this article in press as: H. Zou et al., FlexAnalytics: A Flexible Data Analytics Framework for Big Data Applications with I/O Performance Improvement, Big Data Research (2014), http://dx.doi.org/10.1016/j.bdr.2014.07.001

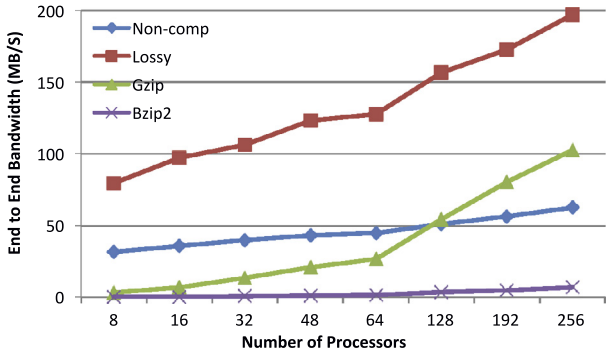

Fig. 7. Data transfer bandwidth comparison without staging.

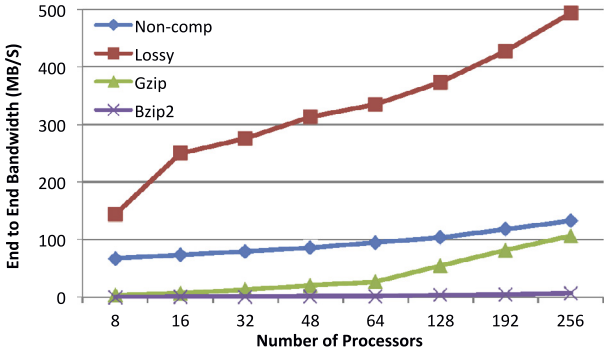

Fig. 8. Data transfer bandwidth comparison with staging.

5.2. Bandwidth analysis from simulation to staging nodes

In the experiment, the output of GKW is written to staging nodes. The compression service is running on computing nodes to reduce the data size. The available end-to-end bandwidth reaches to 132.94 MB/s with 64 computational nodes. We configure 4 staging nodes in the experiment. The experiment simulates two cases: simulation to staging and helper core to staging. Both of them can be summarized as memory-to-memory transfer model. Fig. 8 shows the tested bandwidth with or without compression.

In the experiment, multiple data blocks transfer from simulation nodes to staging nodes. (De)compression and compression are executed on computational nodes and staging nodes separately. Because compression processes are executed on same compute node and share multi-level cache hierarchy (including L3 cache), the cache impact factor should be ($\varepsilon = 1/2$).

5.3. Latency analysis with data query

For data visualization, end users are interested in (i) query latency – the elapsed time between when data is available to when query output appears, and (ii) simulation I/O throughput, which is impeded if data cannot be moved off the high end machine before the application's next output step – due to application blocking on output.

Fig. 9 shows the performance impact of flexible analytics placement on data transfer latency, given variable connection bandwidth between the visualization machine and the simulation. We inject controlled additional network traffic between simulation cluster and ancillary visualization machine, resulting in available bandwidth varying from 20 MB/s to 120 MB/s. The GKW simulation generates 1 GB output data per output step.

We compare two cases: one is to move raw simulation output data to ancillary visualization machine and run query locally on it; the other is to place the query on the simulation cluster to reduce data movement between the two machines. Fig. 9 shows that

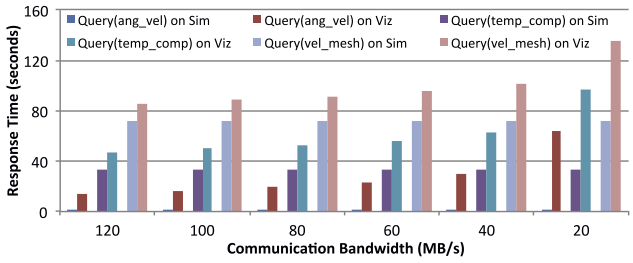

Fig. 9. Data request response time test with/without query.

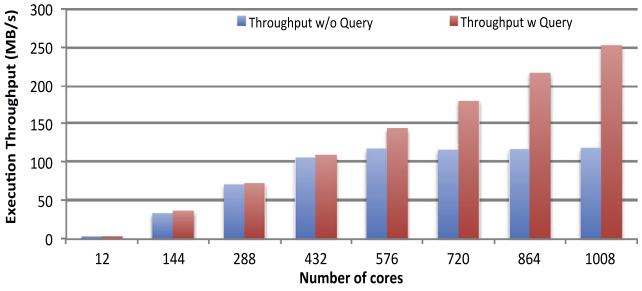

Fig. 10. The improvement of simulation throughput.

the FlexAnalytics system can significantly improve query latencies by reducing data movement volumes for all three tested queries. Even when the available bandwidth between two machines is at its maximum (120 MB/s), there is 23.7% improvement. When available bandwidth decreases to 20 MB/s, the improvement brought by FlexAnalytics can be up to 80%.

5.4. Throughput analysis with data query

Simulation execution throughput (the amount of simulation output over a given interval of time) is impaired when output data cannot be moved off the high end machine between subsequent simulation time steps (i.e., due to blocking on I/O). Fig. 10 shows, under 120 MB/s communication bandwidth, the throughput of the simulation running at different scales. When the simulation scales beyond 432 cores, the data movement cost become significant enough so that placing queries near data source becomes beneficial. At the scale of 1000+ cores, the advantage of FlexAnalytics is as large as 100%.

Results show the FlexAnalytics approach to be capable of placing queries so as to alleviate communication bottlenecks between data sources and analytics processing sites.

6. Related work

In-situ data analytics has been widely used in many data-intensive applications to reduce data size. Data compression and query is useful because it helps minimize storage utilization, reduce networks bandwidth and energy consumption in hardware. However, most of compressions and queries are not transparently operated at the I/O layer, but rather explicitly worked at the application level [25,35,39].

In many recent studies [11,25], compression is applied on big data research to reduce transfer latency and improve response time. All of them concentrate on the study on compression algorithm and data saving on parallel file system. Our work differs from these compression studies, as it does not focus on the study of compression algorithms. The exploration of compression selection is conducted in the paper. We investigate the effects of data compression on the whole I/O path rather the file system to find the best location to place the compression. We show that the compression improves network end-to-end transmission time, network

Please cite this article in press as: H. Zou et al., FlexAnalytics: A Flexible Data Analytics Framework for Big Data Applications with I/O Performance Improvement, Big Data Research (2014), http://dx.doi.org/10.1016/j.bdr.2014.07.001

bandwidth, and consequently the I/O performance with the well-planned placing.

The area of data compression in cloud computing has received many attentions in recent literature. A recent study [3] investigated the effects of compression in MapReduce clusters. This study focused on increasing I/O performance in order to reduce cluster power consumption. Our work focuses on improving I/O performance to achieve better end-to-end application performance. Another data compression studies focus on applying the data compression into grid computing. The highly parallel-distributed data management in grid has the different requirements on data transmission with HEC. Such research is designed for use in Grid computing environments. So, the existing data compression methods cannot be applied on HEC directly.

There are a number of related efforts that provide solutions to data visualization and analysis in the HPC community [1,13,35]. They all use the data sever/client architecture supporting visualization by permitting remote users to acquire images via their visual client from visual servers located on large-scale machines placed 'next' to simulation. The problem of how to move data from the simulation to the visualization machine remains to be unsolved. Out work addresses it. In addition, related work on query-driven visualization [18] considers static data queries rather than ad hoc, dynamic queries enabled by FlexAnalytics. There are also contributions on data indexing, which we have not yet leveraged. Instead, we flexibly place analytics among I/O path to improve both simulation and data access performance. Additional input to out work can be derived from out previous work on adaptive compression and FlexQuery [35,37].

7. Conclusions and future work

In this paper, we studied questions about how to introduce data analytics into HEC to reduce data movement and release severe I/O performance bottleneck. To explore the possible solutions, a quantitative model is built to evaluate the potential analytics algorithms and placement strategies. Based on this model, we propose a flexible analytics framework for I/O performance optimization. This flexible placement strategy combines data compression and visualization query – two algorithms, which can be dynamically selected and switched to achieve the best I/O performance with features profiling and real-time system resource status monitoring.

The experiments with the real application of GKW are conducted on an 80-node 1280-core cluster machine and an SGI visualization machine. The analysis of results shows us that the data reduction ratio and available processors are two most important factors to impact the analytics selection. The experiments investigate these two factors in details. Thus, our future work should focus on the improvement of on these two factors.

Acknowledgements

The work at Argonne is supported in part by the U.S. Department of Energy (DOE), Office of Science, under Contract DE-AC02-06CH11357.

References

[1] J.P. Ahrens, et al., Interactive remote large-scale data visualization via prioritized multi-resolution streaming, in: Proc. of the 2009 Workshop on Ultrascale Visualization, UltraVis'09, 2009.

[2] High-quality data compressor, bzip2, http://www.bzip.org.

[3] Y. Chen, A. Ganapathi, et al., To compress or not to compress compute vs. I/O tradeoffs for MapReduce energy efficiency, in: Proc. of the SIGCOMM Workshop on Green Networking, 2010, pp. 23–28.

[4] Compression utility, gzip, http://www.gzip.org.

[5] MPI Forum, MPI-2: extension to the message-passing standard.

[6] D.A. Lelewer, D.S. Hirschberg, Data compression, in: Proc. ACM Computing Surveys (CSUR), 1987, pp. 261–296.

[7] N. Fabian, K. Moreland, et al., The ParaView coprocessing library: a scalable, general purpose in situ visualization library, in: Proc. IEEE Symp. on Large-Scale Data Analysis and Visualization (LDAV2011), 2011, pp. 89–96.

[8] A. Gerndt, B. Hentschel, M. Wolter, T. Kuhlen, C. Bischof, VIRACOCHA: an efficient parallelization framework for large-scale CFD post-processing in virtual environments, in: Proc. ACM/IEEE Conference on Supercomputing (SC04), 2004, pp. 50–61.

[9] S. Hodson, S. Klasky, Q. Liu, J. Lofstead, N. Podhorszki, F. Zheng, et al., ADIOS 1.3.1 User's Manual, Oak Ridge National Laboratory, 2011, pp. 1–95.

[10] I. Kontoyiannis, Pattern matching and lossy data compression on random fields, in: Proc. IEEE Transactions on Information Theory, 2003, pp. 1047–1051.

[11] S. Laksh, N. Shah, S. Ethier, S. Klasky, R. Latham, R. Ross, N. Samatova, Compressing the incompressible with ISABELA: in-situ reduction of spatio-temporal data, in: Proc. International European Conference on Parallel and Distributed Computing (Euro-Par 2011), 2011, pp. 366–379.

[12] M. Lin, S.S. Vazhkudai, A.R. Butt, F. Meng, X. Ma, Y. Kim, C. Engelmann, G. Shipman, Functional partitioning to optimize end-to-end performance on many-core architectures, in: Proc. ACM/IEEE Conference on Supercomputing (SC10), 2010, pp. 1–12.

[13] K.-L. Ma, A new approach to remote visualization of large volume data, SIGGRAPH Comput. Graph. 44 (3) (2010) 5:1–5:2.

[14] B. Nicolae, High throughput data-compression for cloud storage, in: Proc. of the 3rd International Conference on Data Management in Grid and Peer-to-Peer Systems (Globe'10), 2010, pp. 1–12.

[15] R. Oldfield, S. Arunagiri, P.J. Teller, S.R. Seelam, M.R. Varela, R. Riesen, P.C. Roth, Modeling the impact of checkpoints on next-generation systems, in: Proc. IEEE Conference on Mass Storage System and Technologies (MSST 2007), 2007, pp. 30–46.

[16] J. Piernas, J. Nieplocha, E.J. Felix, Evaluation of active storage strategies for the Lustre parallel file system, in: Proc. ACM/IEEE Conference on Supercomputing (SC07), 2007, pp. 1–10.

[17] K. Sayood, Introduction to Data Compression, 3rd edition, Morgan Kaufmann, 2005.

[18] K. Stockinger, et al., Query-Driven Visualization of Large Data Sets, 2005.

[19] M. Slawinska, T. Bode, H. Zou, et al., A Maya use case: adaptable scientific workflows with ADIOS for general relativistic astrophysics, in: Proc. of the Conference on Extreme Science and Engineering Discovery Environment, 2013.

[20] X.H. Sun, C. Du, H. Zou, V-mcs: a configuration system for virtual machines, in: Proc. IEEE International Conference on Cluster Computing (Cluster'09), 2009.

[21] W. Tang, J. Wilkening, N. Desai, W. Gerlach, A. Wilke, F. Meyer, A scalable data analysis platform for metagenomics, in: Proc. of the IEEE Conference on Big Data, 2013.

[22] W. Tang, Z. Lan, N. Desai, Y. Yu, et al., Reducing fragmentation on torus-connected supercomputers, in: Proc. of the IEEE International Parallel and Distributed Processing Symp. (IPDPS'11), 2011, pp. 828–839.

[23] VTK 6.0.0 documentation, 2013.

[24] Paraview VisIt, OpenSource scientific visualization and graphical analysis tool, 2012.

[25] B. Welton, D. Kimpe, J. Cope, C. Patrick, et al., Improving I/O forwarding throughput with data compression, in: Proc. IEEE International Conference on Cluster Computing (Cluster'11), 2011.

[26] J. Wu, Z. Lan, Y. Yu, et al., Performance emulation of cell-based AMR cosmology simulations, in: Proc. ACM/IEEE Conference on Supercomputing (SC'12), 2012.

[27] J. Wu, Z. Lan, X. Xiong, et al., Hierarchical task mapping of cell-based AMR cosmology simulations, in: Proc. of the International Conference on High Performance Computing, Networking (Cluster'11), 2011.

[28] K. Wu, S. Ahern, et al., FastBit: interactively searching massive data, in: Proc. SciDAC, J. Phys. Conf. Ser. 180 (1) (2009) 1–10.

[29] W.X. Wang, Z. Lin, W.M. Tang, W.W. Lee, S. Ethier, J.L.V. Lewandowski, G. Rewoldt, T.S. Hahm, J. Manickam, Gyro-kinetic simulation of global turbulent transport properties in tokamak experiments, in: Proc. Physics of Plasmas, 2006, pp. 59–64.

[30] Y. Yu, D. Rudd, Z. Lan, J. Wu, et al., Improving parallel IO performance of cell-based AMR cosmology applications, in: Proc. IEEE International Parallel and Distributed Processing Symp. (IPDPS'12), 2012, pp. 933–944.

[31] Y. Yu, J. Wu, Z. Lan, et al., A transparent collective I/O implementation, in: Proc. IEEE International Parallel and Distributed Processing Symp. (IPDPS'13), 2013, pp. 297–307.

[32] F. Zheng, H. Zou, J. Cao, J. Dayal, T. Nugye, G. Eisenhauer, S. Klasky, FlexIO: location-flexible execution of in-situ data analytics for large scale scientific ap-

plications, in: Proc. IEEE International Parallel and Distributed Processing Symp. (IPDPS'13), 2013, pp. 320–331.

[33] H. Zou, H. Jin, Z. Han, et al., A virtual-service-domain based bidding algorithm for resource discovery in computational grid, in: Proc. IEEE/WIC/ACM International Conference on Web Intelligence, 2005, pp. 5–53.

[34] H. Zou, H. Jin, Z. Han, et al., HRTC: hybrid resource information service architecture based on GMA, in: Proc. IEEE International Conference on e-Business, 2005, pp. 541–544.

[35] H. Zou, K. Schwan, M. Slawinska, G. Eisenhauer, F. Zheng, et al., FlexQuery: an online query system for interactive remote visual data exploration at large scale, in: Proc. IEEE International Conference on Cluster Computing (Cluster'13), 2013.

[36] H. Zou, W. Wu, X.H. Sun, et al., An evaluation of parallel optimization for OpenSolaris network stack, in: Proc. IEEE 35th Conference on Local Computer Networks (LCN), 2010, pp. 296–299.

[37] H. Zou, Y. Yu, W. Tang, M. Chen, Improving I/O performance with adaptive data compression for big data applications, in: Proc. IEEE International Parallel and Distributed Processing Symp. (IPDPS'14), 2014.

[38] H. Zou, X.H. Sun, et al., A source-aware interrupt scheduling for modern parallel I/O systems, in: Proc. IEEE International Parallel and Distributed Processing Symp. (IPDPS'12), 2012, pp. 156–166.

[39] H. Zou, F. Zheng, K. Schwan, et al., Quality-aware data management for large scale scientific applications, in: Proc. High Performance Computing, Networking, Storage and Analysis (SC'12), 2012, pp. 816–820.

Please cite this article in press as: H. Zou et al., FlexAnalytics: A Flexible Data Analytics Framework for Big Data Applications with I/O Performance Improvement, Big Data Research (2014), http://dx.doi.org/10.1016/j.bdr.2014.07.001