ANTELOPE: A Semantic-Aware Data Cube Scheme for Cloud Data Center Networks

Yu Hua, *Senior Member*, *IEEE*, Xue Liu, *Member*, *IEEE*, and Hong Jiang, *Senior Member*, *IEEE*

Abstract—Today's cloud data centers contain more than millions of servers and offer high bandwidth. A fundamental problem is how to significantly improve the large-scale system's scalability to interconnect a large number of servers and meanwhile support various online services in cloud computing. One way is to deal with the challenge of potential mismatching between the network architecture and the data placement. To address this challenge, we present ANTELOPE, a scalable distributed data-centric scheme in cloud data centers, in which we systematically take into account both the property of network architecture and the optimization of data placement. The basic idea behind ANTELOPE is to leverage precomputation based data cube to support online cloud services. Since the construction of data cube suffers from the high costs of full materialization, we use a semantic-aware partial materialization solution to significantly reduce the operation and space overheads. Extensive experiments on real system implementations demonstrate the efficacy and efficiency of our proposed scheme.

Index Terms—Cloud computing, data center networks, semantic awareness, data cube

1 Introduction

Cloud data centers are facing the problem of data deluge. The volume of digital content maintained in cloud data centers is growing at an ever increasing pace. According to a recent International Data Corporation (IDC) study, 800 Exabytes of data were created in 2009 [1]. Facebook reports, in June 2010, there exists 21PB raw storage capacity in the internal data warehouse, and moreover, 12TB compressed new data are added every day [2]. In a foreseeable future, this already staggering volume of data is projected to increase. Unfortunately, until now, we are not ready to handle the data deluge. For example, from 1700 responses to a *Science* poll [3], about 20% respondents often use more than 100 GB datasets (wherein 7% over 1TB), more than 63% have asked colleagues for data sharing, and about half of those polled store the data only in their own labs due to lack of funding to support archiving, let alone real-time data analysis (e.g., online queries). In order to efficiently handle big data analytics, cloud platforms have emerged, such as MapReduce [4], Hadoop [5], Dryad [6], Pig [7] and Hive [8], which demonstrate the ability to scale to thousands of nodes, and support fault tolerance, high availability and automatic management. Moreover, users routinely pose queries across hundreds of Gigabytes of data stored on data centers [9]. A cost-effective scheme in real-world applications hence becomes more important to efficiently satisfy users' requests and significantly improve system performance.

In recent years, the data centers for real-world applications have been built to provide various services, such as information retrievals, E-mails, instant messages and Web services. Major players like Amazon, Google, Microsoft, IBM, Facebook, Apple, Intel and Yahoo! are constructing mega-data centers for cloud computing [10]–[12] to compete for such service-oriented markets, by moving the computation, storage and operations to the cloud computing platform. Data centers are themselves a networking infrastructure that connects a large number of servers via high-speed links, routers and switches. Providing online services in a scalable cloud computing environment has become a main concern for the IT industry. This has become an emerging and important research topic in cloud computing.

The essence of online cloud service in data centers is to provide real-time response when carrying out various operations, such as query services and system configuration. For example, an online query service in data centers can identify "hot spot" data that are frequently visited by measuring the maximum of I/O accesses. In order to obtain load balance and alleviate performance bottlenecks, we need to carry out data migration [13] or replica control [14] in advance. Furthermore, users are often interested in the "hot spot" data that can satisfy most query requests with high accuracy. A prefetching or caching scheme can be further used to decrease query latency. Therefore, providing online cloud services demonstrates the benefits of quick response, system optimization and cost savings, which are critical and important to enhance the scalability of large-scale cloud data centers.

Most large-scale cloud computing applications essentially require the online services that cloud data-center networks support to be scalable and highly efficient. In order to improve the scalability and efficiency in data centers, researchers have recently proposed several data-center network architectures, such as Portland [15], Ficonn [16], VL2 [17], DCell [18], BCube

- *Y. Hua is with the Wuhan National Laboratory for Optoelectronics, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China. E-mail: csyhua@hust.edu.cn.*
- *X. Liu is with the School of Computer Science, McGill University, Montreal, Quebec H3A 0E9, Canada. E-mail: xueliu@cs.mcgill.ca.*
- *H. Jiang is with the Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, NE 68588-0150 USA. E-mail: jiang@cse.unl.edu.*

Manuscript received 28 May 2012; revised 05 Feb. 2013; accepted 24 Apr. 2013. Date of publication 05 May 2013; date of current version 07 Aug. 2014. Recommended for acceptance by J. Weissman.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2013.110

[19], Spain [20] and fat tree [21]. Although the newly proposed architectures work well for their own design purposes, such as throughput increments, reliability enhancements and cost savings. They do not consider the online applications running on top of the architectures. The key issue is the lack of the comprehensive considerations in terms of the data placement, which is tightly associated with access patterns in online cloud services. Therefore, we need to take into account both the properties of network architecture and data placement to provide scalable online cloud service. Specifically, we need to handle three main challenges.and implementation.

Weak scalability: Data center networks mainly use switch-based tree structure to interconnect the increasing number of servers and do not scale well. The hierarchical tree based physical configurations require expensive and high-speed switches to sustain the exponential growth of servers. Hence, core and rack switches in this tree often pose as the bandwidth bottleneck. Furthermore, the source-destination link can be shared by many other host pairs. Traffic congestion often arises, in particular near the higher hierarchy (e.g., root switch of the tree). One observation is that the aggregate throughput becomes much lower than the sum of network interface card throughputs [22]–[24].

Limited inter-server bandwidth capacity: Data centers require bandwidth-intensive communication supports for IT infrastructure services such as GFS [25], BigTable [26], MapReduce [4] and Dryad [6]. Limited ports in the high-cost switches unfortunately decrease aggregate bandwidth moving up the hierarchy and result in large oversubscription [17]. The over-subscription severely limits the overall performance due to preventing overloaded services from being assigned to idle servers and meanwhile requiring high-cost hardware to support more ports for interconnection. Hence, if all communications need to go through limited high-level core switches, data centers will decrease the overall performance.

Low link utilization: Most existing designs for data center networks pay little attentions to the link bandwidth utilization for network transmission. Ideally, a server can obtain queried results from its own or adjacent servers, rather than remote ones. The reduction of path length allows us to obtain the fast query response and the increment of link utilization. In order to improve link utilization, a key issue is how to carry out (near)-optimal data placement among millions of servers in the cloud data-center networks.

In order to address the above challenges and support online cloud services, we propose ANTELOPE, a scalable distributed data management scheme, which can bridge the gap between network architecture and data placement in large-scale cloud data centers. The basic idea behind ANTELOPE is to leverage off-line precomputation to improve online query performance. The precomputation model in ANTELOPE is data cube [27]. The rationale comes from a proper understanding between the data cube and the semantic-based data management in cloud data centers. On one hand, using the data cube can provide the online service and support rich dimension queries. The data cube supports not only conventional queries for original data, but also the queries for statistic based measures, such as *Max* and *Min*. The statistic based queries meet the needs of decreasing data migration, offering real-time response and supporting queries for derived dimensions that are the computation results of source data. Cloud data centers can hence significantly reduce the amounts of transmitted data since in many cases, what users are really concerned with is the statistic results, rather than the source data. On the other hand, the construction of a data cube is a data-intensive task that incurs large amounts of computation and storage overheads. The potential semantic correlation from the access patterns can significantly reduce the operation overheads. ANTELOPE optimizes the construction of data cube by identifying semantic correlation and efficiently supports online cloud services in data centers. We make the following contributions.

First, a data cube [27] can accurately and efficiently satisfy on-line aggregate query requests by using precomputed statistics. The data cube consists of multi-dimensional aggregates that come from a fact table with a measure attribute and a set of dimensional attributes. However, performing the construction of a data cube is non-trivial due to the problem of full materialization [28]. The full materialization is to precomputes all possible aggregates and unfortunately incur very high computation and storage costs. In order to address this problem, we use a *semantic-aware partial materialization* as a suitable tradeoff between the construction efficiency and the query accuracy by precomputing the related, rather than all, aggregates.

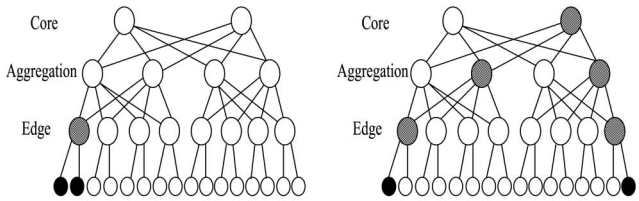
Second, given the real-life fact of high-cost and limited-ports switches, what we can do is to optimize the data placement to improve the entire throughput, especially among the low-level servers. ANTELOPE achieves this by aggregating data with strong locality into the same or adjacent servers. Specifically, by exploring access patterns, ANTELOPE places data close to their locality-aware servers with the aid of Locality-Sensitive Hashing (LSH) [29], [30]. We thus significantly enhance network bandwidth utilization through core switches. The high-level links in the hierarchy will not become the performance bottleneck. When new servers are added, existing running servers have little influence. In particular, our data-centric placement design serves as virtual layer well and is suitable for arbitrary low-level network topology.

Third, in order to carry out (near)-optimal data placement in large-scale cloud data centers, ANTELOPE explores the locality residing in the access patterns such that data with strong locality can be aggregated and placed in the same or adjacent servers. We improve the bandwidth utilization. The path length for completing query operations is significantly reduced. Performing the fast identification of data locality generally requires heavy computation and space overheads. LSH [29] can efficiently identify data locality with acceptable complexity.

The rest of the paper is organized as follows. Section 2 shows the research backgrounds especially in traffic patterns analysis, data cube and locality sensitive hashing. Section 3 describes the design principles of ANTELOPE. Section 4 shows the implementation details. We study extensive experiments in real system implementations in Section 5. We present the related work in Section 6. Finally, we conclude our paper in Section 7.

2 Backgrounds

This section shows the research backgrounds of ANTELOPE design. Observations from access pattern analysis motivate our research work that makes use of data cube as precomputation model.



(a) LR=3/30. (b) LR=7/30.

Fig. 1. Locality rates for multi-rooted hierarchy with random routing.

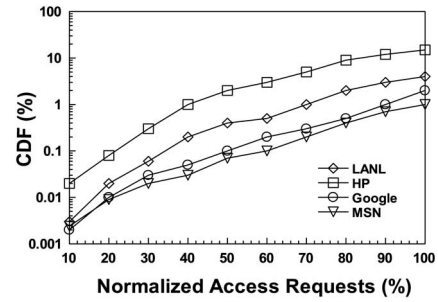


Fig. 2. Cumulative distribution function of the locality rates of the real-world traces.

2.1 Locality-Aware Analysis

Currently, since no large-scale data center traffic traces are publicly available, we generate patterns along the lines of traffic distributions in published work and open system traces to emulate typical data center workloads. The measured traces are listed as follows.

- **LANL**: Los Alamos National Laboratory (LANL) recently released multiple sets of data [37]. These metadata show the information about files. This data set is about 19 GB and consists of roughly 112 million lines of archive data and roughly 9 million lines of home/project space data. The attributes of these data include unique ID, file sizes (in bytes), creation time, modification time, block sizes (in bytes) and the paths to files.
- **HP**: HP file system provides a 10-day 500 GB trace [33] that records the accesses from 236 users. The trace records multiple operations, such as READ, WRITE, LOOKUP, OPEN, and CLOSE, on the accessed files with file names and device numbers.
- **MSN**: MSN trace [34] maintains metadata information and correlated users within a 6-hour period and has been divided into 10-minute intervals. This trace contains 1.25 million files and records 3.3 million "READ" and 1.17 million "WRITE" operations. The queried objects are the files that exhibit multi-dimensional attributes, including access time, the amounts of READ, the amounts of WRITE, operational sequence IDs and file size within an examined interval.
- **Google Cluster**: Google recently releases anonymized log data from their clusters [35]. This is a collected trace in a 7-hour period. The workload in the trace consists of a set of tasks and each task runs on a single machine. Tasks consume memory and one or more cores (in fractional units). Each task belongs to a single job. One job may have multiple tasks (e.g. mappers and reducers). The trace has totally 3,535,029 observations, 9218 unique jobs and 176,580 unique tasks.

To measure the locality of access patterns in data centers, we make use of a metric, called *locality rate*, which is defined as the percentage of switches/routers that recursively contain the visited nodes as shown in Fig. 1. A lower value of this rate means much stronger locality.

We illustrate the Cumulative Distribution Function (CDF) of locality rates for above traces in Fig. 2. We observe that real-world applications usually exhibit strong locality by accessing close and correlated data. The average value of four traces' locality rates is 3.25% and the maximum is 11.2%, which means that most access requests can be completed within adjacent low-level servers. Based on the observations, we argue that an optimized data placement can improve scalability and improve link utilization. The observations also motivate our ANTELOPE design for supporting online cloud service.

2.2 Data Cube for Online Query

A data cube [27] uses multi-dimensional aggregates based on off-line precomputation to obtain fast on-line runtime performance. A cube consists of a lattice of *cuboids*. Each cuboid is associated with an aggregate of measure attributes according to a *group-by* operation. This operation uses a subset of the dimensional attributes. Data cube hence precomputes the aggregation of all possible combination of dimensions to facilitate the queries.

Definition 1 (Data Cube). *Consider a relation $R(A, S)$ with p-dimensional attributes $A_1, A_2, \cdots, A_p$ and non-negative scores, S, to represent the attributes A^S ordered by S. The data cube D_R is the set of aggregates that are the precomputed results through executing group-by operations on R.*

When the cardinality of p attributes are $L_1, L_2, \cdots, L_p$, it becomes $\prod_(L_i + 1)$. A *cell* c over cube measure can be represented as a group-by cell in a d-dimensional cuboid when $d(d \leq p)$ values from $A_1, A_2, \cdots, A_p$ exist.

Ranking operations in ANTELOPE need to rank the multi-dimensional aggregates in each cuboid in ascending or descending order. The ranked aggregates allow explicit representation of stored data to efficiently answer aggregate queries. The ranked results can be ordered by score defined in Definition 1. The score comes from the computation of ranking functions on the measures in each cuboid.

Definition 2 (Ranking Function). *A ranking function F^S sorts each aggregate based on measure values represented as score S in descending or ascending order. The ranked aggregates are formulated by group-by attributes, $A_1^S, A_2^S, \cdots, A_d^S (d \leq p)$.*

Typical ranking functions execute precomputation by using aggregation measures, such as *sum*,*ave*, *max* and *stddev*, and in this paper, as an example, we use *max* to compute the maximum of the aggregates that are then ranked by their scores. According to the ranking function, we can further obtain the ranking cuboids, in which the aggregates are ordered by their scores in multi-dimensional analysis.

Definition 3 (Cuboids). *Given group-by attributes $A^S = \{A_1^S, A_2^S, \cdots, A_d^S\} (A^S \subseteq A, d \leq p)$ and ranking function F^S, the ranking cuboids C^S are defined as the subset of all traffic messages and each cuboid contains d cells, $\{r^1, r^2, \cdots, r^d\}$, which come from the aggregation over S by using ranking measures from $F_1^S, F_2^S, \cdots, F_d^S$ respectively.*

TABLE 1
Typical Schemes for Cube Compression and Aggregation

Basic Ideas	Examples	Design Purpose		Performance Metrics			
		Compression	Aggregation	Query Accuracy	Space Efficiency	Time Efficiency	Scalability
Sampling	Dynamic Selection [40]	√		Approximate	Y	N	Y
Factorization	Multi-way Array [41]	√		Approximate	Y	N	N (clustering)
Probability Density	Compressed cube [42]	√		Approximate	Y	N	N (clustering)
Loglinear	Quasi cube [43]	√		Approximate	Y	N	N (clustering)
Ranking	ARCube [44]		√	Approximate	N	Y	Y
	Ranking Cube [45]		√	Approximate	N	Y	Y
Signature	P-Cube [46]	√		Exact	N	Y	Y
Tuples Aggr.	Condensed Cube [47]		√	Exact	Y	N	N (fully computed)
Correlation	Range cube [48]	√		Exact	Y	Y	N (careful partition)
Semantics	ANTELOPE		√	Approximate	Y	Y	Y

A series of ranking cuboids construct ANTELOPE that takes into account ranking measures to facilitate top-k aggregate queries.

Definition 4 (Ranking (Top-k) Aggregate Queries). *A ranking aggregate query can obtain k cells $\{c_1, c_2, \cdots, c_k\}$ through involving the group-by $R(A^S)$ to satisfy that any other cell $c^* \in R(A^S)$, $F^S(c^*) \leq \min(c_i | c_i \in R(A^S))$.*

Data cube answers top-k aggregate queries by ranking the group-by results from precomputed aggregate values and further obtaining the top-k groups. The ranking aggregates in the above example actually utilize full materialization approach to first precompute all possible combinations of multi-dimensional attributes and then rank them in the descending order. Storing these multi-dimensional precomputed results often consumes too much storage space, in particular with the growth of the number of dimensions and the size of associated hierarchy.

Performing the materialization on a data cube is to precompute the ranked multi-dimensional aggregates for each cuboid to facilitate ranking-based aggregate queries while requiring certain storage space to store and maintain generated results. In practice, there are two baseline choices for cube materialization. One is to use *no materialization* method that fully depends on on-line computation and does not precompute any of "non-base" cuboids. Since there is no precomputed results, no materialization approach essentially requires expensive costs for on-line computation and thus gives extremely slow responses to query requests. In contrast, the other approach, i.e., *full materialization*, precomputes all possible combinations of aggregates. Although the full materialization can quickly provide query response, it obviously occupies huge amounts of storage space that is often much larger than the available capacity of local memory. Therefore, a tradeoff between storage space and response time is more interesting and important to efficiently organize and store the precomputed results.

A data cube usually suffers from the high space overhead in practical applications and the main solutions to decrease cube sizes can be classified into compression and aggregation as shown in Table 1. Compared with existing work, ANTELOPE uses typical information retrieval tool to exploit the semantic correlation among received messages and only precomputes correlated aggregates to carry out partial materialization with the benefits of space savings.

In practice, the ANTELOPE needs to carefully select precomputed aggregates that consist of some subsets of entire dataset. These selected subsets are represented as some cuboids to satisfy user-specified requests. Since user requests usually produce the checking on some correlated subsets of entire cube structure, it is naturally unnecessary to precompute all possible cuboids.

2.3 An Example

A data cube in cloud data centers can provide online aggregate queries by using precomputed results. A typical service as a case study is to identify potential performance bottleneck, e.g., "hot spot" data. Table 2 shows an example of I/O access behaviors according to the dimensions *Position*, *I/O Behavior* and *Period* by considering *Access Times* as a numeric *measure*. The measure value representing a numerical function aggregates the data belonging to a given cuboid defined by dimension-value pairs in the data cube space. For instance, this table shows the numbers of I/O access going through 4 servers (A, B, C and D), at 3 periods (Morning, Afternoon and Evening) and in 2 I/O behaviors (Read and Write). According to the fact table, we further construct a 3-dimensional data cube as shown in Fig. 3 to illustrate the data cube structure. Note that the data cubes are multi-dimensional, not limited to 3-D, and any n-D data tables can be displayed as a series of $(n-1)$-D cubes. Due to space limitation, here we do not display higher dimensional cubes.

Given a set of dimensions, a data cube consists of a series of cuboids. Each cuboid is correlated with a subset of the given dimensions. Fig. 4 shows the data representation at different

TABLE 2
An Example of I/O Access Behaviors

		*Server.*A	*Server.*B	*Server.*C	*Server.*D
Morning	Read	56	206	127	82
	Write	28	372	165	55
Afternoon	Read	57	196	188	152
	Write	35	107	162	67
Evening	Read	10	22	6	8
	Write	5	17	11	9

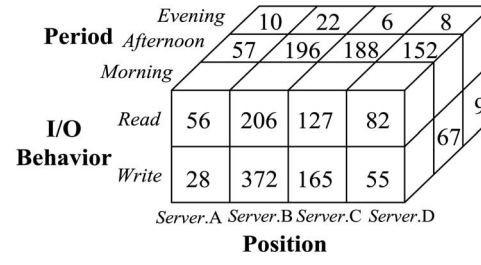


Fig. 3. A 3-D representation of the data in Table 2.

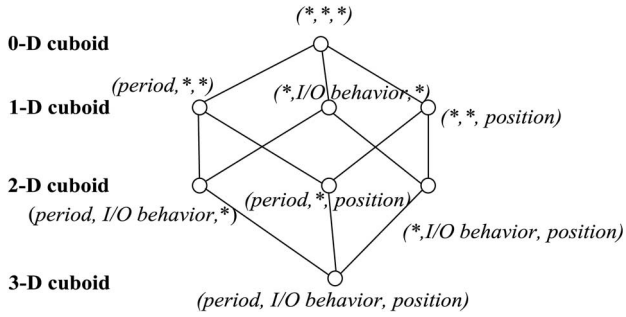


Fig. 4. A data cube consisting of a lattice of cuboids for the dimensions *position*, *I/O behavior* and *period*.

levels of aggregation. Each dimension in the 0-D (**apex**) cuboid is not specified and thus the cuboid aggregates all information in entire data cube. In contrast, when each dimension in the 4-D (**base**) cuboid is specified, the cuboid only displays one cell in the data cube. Thus, the data cube can answer aggregate query by checking multiple cuboids, each of which represents a different level of summarization over a set of cells.

A data cube can support aggregate queries that are correlated with multiple dimensions in an *ad-hoc* manner by verifying a subset of all precomputed combinations of multi-dimensional attributes. Query requests receive answers by checking partial cuboids as shown in Fig. 4. For example, "*How many accesses are there onServer_B*?" can be answered by checking a 1-D cuboid, and "*How many accesses are there by reading operation in the morning*?" can be answered by checking a 2-D cuboid. On the other hand, these examples potentially indicate another critical problem of ranking aggregates, since each cuboid possibly contains multiple aggregates and the nearest results should be the answers. In addition, the concept of hierarchy means a sequence of concept mapping among different levels. For example, considering the dimension *Position*, the *Server* values can be mapped to *disk* or *directory* which it belongs to.

ANTELOPE uses partial materialization to obtain significant space savings and provide fast query response. For example, considering the I/O access records in Table 2, "hot spot" data in *Server.A* possibly introduce further queries on alternative servers for load balance, which may consider I/O access history for performance prediction and decision making. We thus need to locate the cuboid that is correlated with the event and then check the precomputed subsets in the cuboid to obtain query answers. As shown in Fig. 4, when *Server.A* becomes performance bottleneck, the 3 rd point $(*, *, position)$ in the 1-D cuboid is correlated with the event and then the 2-D and 3-D cuboids in the lower levels connecting with this point will be also precomputed to answer query requests. The "hot spot" data drive the precomputation on partial, not all, cuboids.

In essence, ANTELOPE offers approximate query accuracy in the context of the big data era. The potential applications demonstrate some common characteristics, such as massive data, distributed deployment, heterogeneous forms and on-line processing, which introduce the great challenge of data processing. In order to address this challenge, ANTELOPE leverages a proper tradeoff between a very small query inaccuracy and significant performance improvements. In general, this tradeoff can be accepted by many typical real-world applications, such as on-line image query and processing [45], [46], keyword-based search in documents [47], [48], social network analysis [49] and industrial product optimization [50], [51].

3 Design of Antelope

In this section, we present the design principles of ANTELOPE. To significantly decrease the computation complexity, we leverage cost-effective partial materialization, rather than full or no materialization, to build the cube structure that stores and maintains semantic-aware data. Semantic vectors are further used to accurately represent the semantic-aware data. Moreover, in order to handle the curse of dimensionality [29], we make use of the locality sensitive hashing (LSH) to fast and efficiently identify the semantic-aware data.

3.1 Section Partial Materialization

Performing the materialization on a data cube is to precompute the ranked multi-dimensional aggregates for each cuboid to facilitate ranking-based aggregate queries. The data cube follows a principle to simplify computation costs.

Principle 1. *If a given cell does not satisfy minimum support, then no descendant of the cell will satisfy minimum support.*

Based on this principle, there are two baseline choices for cube materialization. One is *no materialization* method that fully depends on on-line computation and does not precompute any of "non-base" cuboids. Since there is no precomputed results, this approach essentially demands expensive costs for on-line computation and thus gives extremely slow responses to query requests. In contrast, the other approach, i.e., *full materialization*, precomputes all possible combinations of aggregates. Although the full materialization can quickly provide query response, it obviously requires large storage capacity. The used storage capacity is often much larger than the available memory size. Therefore, a suitable tradeoff between storage space and response time, i.e., partial materialization, is more important to efficiently execute the precomputation for online services.

ANTELOPE needs to select which aggregates can be precomputed based on the access patterns upon the locality-aware data. These aggregates actually come from the subsets of entire dataset and are represented as the cuboids to meet the needs of system optimization from an administrator. An administrator may be concerned with the servers with "hot spot" data, which is described as the maximum of I/O accesses as show in Fig. 7. We can compute the cuboids that have I/O accesses more than a threshold, e.g., 150 times. Thus, according to the Principle 1, since $(evening, *, *)$ and $(*, *, Server\_A)$ have the values that are smaller than 150, their descendants will not contain the larger value. It is unnecessary to compute the descendant cuboids. Therefore, we partially materialize the data cube model and obtain the computation and space savings. The thresholds in multiple levels determine the sizes of precomputed results and depend upon the available memory sizes.

In order to efficiently support the operations of partial materialization, we need to accurately represent and carefully identify correlated data. These correlated data can facilitate the cost-effective construction of cube structure.

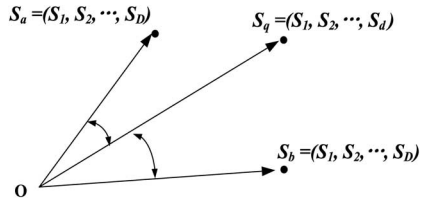


Fig. 5. Measure of semantic vectors.

3.2 Correlation-Based Semantic Representation

Information retrieval tools, such as vector space model (VSM) [52], K-means [53] and latent semantic indexing (LSI) [54], play a role in supporting correlation analysis. VSM heavily suffers from synonyms and noise in representing correlated documents since there are false positives from word substrings match and false negatives from documents with similar context but different term vocabulary while overlooking the order of terms appearing in the document. The results from K-means approach may be not satisfactory due to the inappropriate choice of K input and the distribution of the initial set of clusters.

We use Latent Semantic Indexing (LSI) [54] as an analysis tool to measure semantic correlation of stored data [55], [56]. The data are essentially represented as base cuboids that are further extracted by LSI to identify which are correlated with each other. Specifically, LSI leverages the Singular Value Decomposition (SVD) [57] to measure semantic similarity. SVD reduces a high-dimensional vector into a low-dimensional one by projecting the large vector into a semantic subspace. Specifically, SVD decomposes an attribute-file matrix A, whose rank is r, into the product of three matrices $A = U\Sigma V^T$, where $U = (u_1, \ldots, u_r) \in R^{t\times r}$ and $V = (v_1, \ldots, v_r) \in R^{d\times r}$ are orthogonal, $\Sigma = diag(\sigma_1, \ldots, \sigma_r) \in R^{r\times r}$ is diagonal, and σ_i is the i-th singular value of A. V^T is the transpose of matrix V. LSI utilizes an approximate solution by representing A with a rank-p matrix to delete all but p largest singular values, $A_p = U_p\Sigma_p V_p^T$.

Definition 5. *Each data item a with D-dimensional attributes can be represented as* ***semantic vector*** $S_a = [S_1, S_2, \cdots, S_D]$. *Similarly, query q can be also represented as* $S_q = [S_1, S_2, \cdots, S_d](1 \le d \le D)$.

Fig. 5 shows an example to measure semantic vectors. We observe that vector a is more correlated with vector q than b since the former exhibits smaller angle by computing cosine similarities in the multi-dimensional space.

In this way, LSI projects a query vector $q \in R^{d\times 1}$ into the p-dimensional semantic space in the form of $\hat{q} = U_p^T q$ or $\hat{q} = \Sigma_p^{-1} U_p^T q$. The latter, i.e., inverse of the singular value, is used to scale the vector. The similarity between semantic vectors is measured by their inner product.

LSI tool is able to identify correlated data in real-world applications [58], [59]. However, it is difficult to directly apply LSI into large-scale data centers due to frequent dynamic configuration and potential skewed distribution with bursts in the stored data, thus making challenging the operations of semantic analysis. We hence present an improved LSI to reduce analysis complexity by using the popularity of multi-dimension attributes of data.

In essence, LSI uses the SVD to derive low-dimensional representation of semantic space and in practice, low-rank matrix is an approximate representation of high-rank matrix. Unfortunately, SVD in LSI is not scalable with respect to storage space and computation costs to execute matrix-based computation. The main reason is that conventional LSI equally treats each value in the matrix, which often becomes sparse, and overlooks their popularity, which comes from the spatial, temporal and content localities of multi-dimensional attributes.

Example 1 (Popularity Awareness)

- *Spatial Popularity: Adjacent files are often visited together.*
- *Temporal Popularity: A visited file is possible to be frequently accessed.*
- *Content Popularity: Similar files are usually prefetched.*

Our Popularity-aware LSI (PLSI) exploits the popularity of data objects and further transforms the original sparse matrix into a *block matrix* without losing any information. The block matrix allows the divided parts of entire matrix to be processed in parallel, thus decreasing computation delays.

Definition 6 (Popularity-Based Block Matrix). *Given a matrix $R = [b_1 b_2 \cdots b_d] \in \mathbb{R}^{t\times d}$, it is a popularity-based block matrix when each submatrix $sub(R) = [b_i \cdots b_j], 1 \le i < j \le d$ is zero or has less than $(j - i + 1)$ different matrix eigenvalues $(\lambda_i \cdots \lambda_j)$.*

Popularity-based block matrix contains multiple zero submatrixes that can be simplified in matrix computation. The popularity-aware non-zero submatrixes exhibit the correlation in the stored data, thus supporting efficient aggregation.

Theorem 1 (Submatrix Correlation). *Submatrix $sub(R) = [b_i \cdots b_j]$, $1 \le i < j \le d$, is linearly correlated if the number of different matrix eigenvalues is less than $(j - i + 1)$.*

Proof. According to the Definition 6, the submatrix in the popularity-based block matrix has less than $(j - i + 1)$ different matrix eigenvalues. We first consider the proof by contradiction, i.e., if the submatrix has $(j - i + 1)$ different matrix eigenvalues, it must be linearly independent. The conclusion can be further proved by mathematical induction.

Assuming $(j - i)$ different eigenvalues in $sub(R)$ correspond to $(j - i)$ characteristic vectors $[b_i \cdots b_{j-1}]$ that are linearly independent, i.e.,

$$\varepsilon_i b_i + \cdots + \varepsilon_{j-1} b_{j-1} = 0. \quad (1)$$

We then need to prove that for $(j - i + 1)$ different eigenvalues $(\lambda_i \cdots \lambda_j)$, the corresponding vectors are also linearly independent. Assume $sub(R)b = \lambda b$ and

$$\varepsilon_i b_i + \cdots + \varepsilon_j b_j = 0. \quad (2)$$

Thus we obtain $\sum_{v=i}^{j} \varepsilon_v \lambda_v b_v = 0$. Further combining Equation 2 produces $\varepsilon_{i-j+1}(\lambda_v - \lambda_{i-j+1}) = 0$. On the other hand, since Equation 1 shows the $(\lambda_v - \lambda_{i-j+1}) \neq 0$, ε_{i-j+1} must be zero. Thus, the submatrix with $(j - i + 1)$ different matrix eigenvalues must be linearly independent. Since block matrix has less than $(j - i + 1)$ different matrix eigenvalues, $sub(R) = [b_i \cdots b_j]$, $1 \le i < j \le d$, is linearly correlated. □

We can transform a large and sparse matrix to become small and dense block matrix while decreasing processing delays due to parallel computation. The matrix decomposition only aggregates approximate data to accelerate the computation without information loss.

Theorem 2 (Lossless Block Matrix). *The block matrix with multiple submatrixes is equivalent to the original matrix.*

Proof. The rank of block matrix comes from its submatrixes in which we count the number of non-zero rows/columes. The submatrixes are linearly correlated and distinguished from zero submatrixes. Furthermore, the division on the original matrix only needs to carry out the transformation of matrix row and columns, thus keeping its rank unchanged. Since block and original matrixes have the same rank, they are equivalent. □

3.3 Identification of Locality-Aware Data

ANTELOPE uses locality sensitive hashing (LSH) [29] to identify locality-aware data. Specifically, data points a and b that have d-dimensional attributes can be represented as vectors $\vec{a}_d$ and $\vec{b}_d$. If the distance between vectors $\vec{a}_d$ and $\vec{b}_d$ is smaller than a pre-defined threshold, they are considered to be similar [65]. We then say that these similar data are locality-aware.

LSH maps similar items into the same hash buckets with a high probability to serve main memory algorithms for similarity search. For a given request for similarity search query, we need to hash query point q into buckets in multiple hash tables, and furthermore union all items in those chosen buckets by ranking them according to their distances to the query point q. We hence can select the closest items to a queried one. LSH function family has the property that items that are close to each other will have a higher probability of colliding than items that are far apart. We define S to be the domain of items. Distance functions $||*||$ correspond to different LSH families of l_s norms based on s-stable distribution to allow each hash function $h_{a,b} : R^d \rightarrow Z$ to map a d-dimensional vector v onto a set of integers.

Definition 7. *LSH function family, i.e., $\mathbb{H} = \{h : S \rightarrow U\}$ is called (R, cR, P_1, P_2)-sensitive for distance function $||*||$ if for any $p, q \in S$*

- *If $||p, q|| \leq R$ then $Pr_{\mathbb{H}}[h(p) = h(q)] \geq P_1$,*
- *If $||p, q|| > cR$ then $Pr_{\mathbb{H}}[h(p) = h(q)] \leq P_2$.*

The settings of $c > 1$ and $P_1 > P_2$ support similarity search. Multiple hash functions can further increase the gap between P_1 and P_2. The hash function in $\mathbb{H}$ is $h_{a,b}(v) = \lfloor \frac{a.v+b}{\omega} \rfloor$, where a is a d-dimensional random vector with chosen entries following an s-stable distribution, b is a real number chosen uniformly from the range $[0, \omega)$ and ω is a large constant.

Fig. 6 shows an example of LSH working scheme in terms of measured distance. Specifically, LSH can determine the proximate locality between two points by examining their distance in a metric space. If the circle centered at q with radius R covers at least one point, e.g. p_1, as shown in Fig. 6(a), LSH can provide a point with no more than cR distance to q as a query result. We can observe that there is an uncertain space in LSH from R to cR distance and the query q will obtain a reply of either point p_1 or p_2, since both points locate within distance cR, i.e. $||p_1, q|| < cR$ and $||p_2, q|| < cR$. On the other hand,

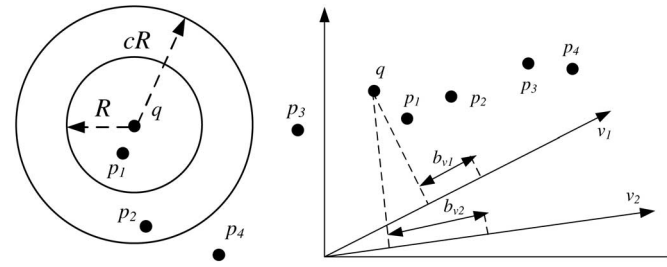


Fig. 6. An example of LSH working scheme.

points p_3 and p_4 are not close to the queried q due to more than cR distance. Fig. 6(b) shows the geometry hashing results from two vectors v_1 and v_2.

In practice, we use hash tables as temporary storage space to maintain the locality-aware data that are then mapped into an R-tree. A server can contain one or more leaf nodes and branches of this R-tree, i.e., a subtree, depending on its own capacity.

4 System Implementations

This Section discusses the implementation issues of ANTELOPE in cloud data centers. In order to offer efficient management of computation and storage models, we first describe the mapping scheme between precomputed cube and storage structure R-tree [61]. We also present the operations, including insertion, deletion, aggregate queries and incremental updates, in ANTELOPE.

4.1 Structure Mapping

We use R-tree [61] to maintain the locality-aware data. The benefits are twofold. One is to support online query service and the other is to facilitate partial materialization. We respectively discuss them.

An R-tree structure is a dynamic and height-balanced index structure. The R-tree height, i.e., the path length from the root to any leaf node, is identical. Minimum Bounding Rectangles (MBR) represent the data that has multi-dimensional attributes. MBR in each dimension denotes an interval of the enclosed data with a lower and an upper bound. MBR in fact partitions data into different groups in the multi-dimensional space. An R-tree allows multi-dimensional queries by aggregating attribute values into corresponding ranges. We can build an R-tree in an iterative way. R-tree provides efficient query service via accessing only a small amount of nodes.

ANTELOPE maps locality-aware data in hash tables to the nodes of R-tree, which correspond to the servers of cloud data center network. Specifically, we map the precomputed cuboids of ANTELOPE to corresponding R-tree nodes that further support aggregate queries and other dynamic operations, such as insertion, deletion and update. Fig. 7 shows an example to illustrate the mapping between the cuboids of ANTELOPE and R-tree nodes. Due to space limitation, we only display the mapping for 0-D and 1-D cuboids as examples and higher-dimension cuboids follow the same way. In each cuboid, the aggregated data are ranked according to the ranking function and here we use the *max* operation in the descending order. All data groups in each cuboid are stored by sorted lists.

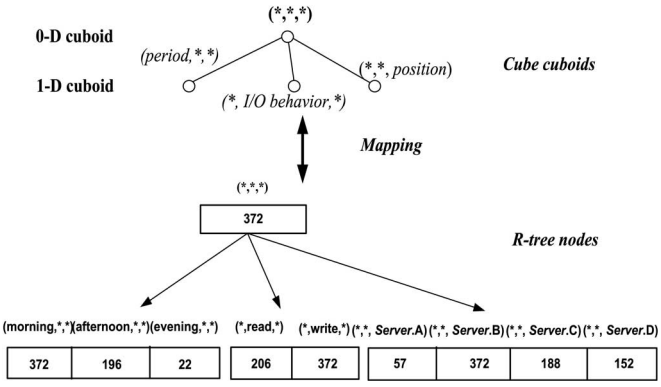


Fig. 7. Implementation mapping from cube cuboids to ANTELOPE nodes for 0-D and 1-D cuboids using MAX operation.

On-line_Aggregate Top-k Query

Input: Top-k query Q for d_Q-dimensional attributes $(d_Q \leq p)$, A^S, Cuboids $C^S(A_1), C^S(A_1), \cdots, C^S(A_p)$

Output: Top-k aggregate cells

1: For each materialized cuboid $C^S(A_i)(i = 1, \cdots, d_Q)$, create a sorted list;
2: Scan all cuboids into memory to initialize sorted lists;
3: $Result := \emptyset$ for storing found results;
4: **while** $Nonempty(All\_sorted\_lists) \cap (Number(Result) < k)$ **do**
5: Select the ranking cell c_i with measure maximum $F^S(c_i)$ from each cuboid $C^S(A_i)$;
6: $\vec{c}$=Max{all selected cells c_i};
7: Insert $\vec{c}$ into *Result*;
8: Delete $\vec{c}$ from its sorted list;
9: **end while**
10: Return *Result*

Fig. 8. Top-k aggregate query algorithm.

4.2 Grouping Procedures

We calculate the correlation among the groups, each of which is represented by a leave node of R-tree. Given multiple nodes storing high-dimensional metadata, a semantic vector with d attributes $(1 \leq d \leq D)$ is constructed to represent each of the N metadata nodes. By using the semantic vectors of these N nodes as the input, we obtain the semantic correlation value between two nodes, x and y, among N nodes.

We need to further build the parent nodes in the R-tree. Nodes x and y are aggregated into a new group if their correlation value is larger than a predefined threshold ε_1. When a node has multiple correlation values that are larger than ε_1, the node with the largest correlation value will be chosen. These groups are recursively aggregated until all of them form a single one that is the root of R-tree. Each tree node uses minimum bounding rectangles to represent all metadata that can be accessed through its children nodes.

The above procedures aggregate all data into an R-tree. For queries, the query traffic is very likely bounded within one or a small number of tree nodes due to metadata semantic correlations and similarities. If each tree node is stored on a single metadata server, such query traffic is then bounded within one or a small number of metadata servers. Therefore, ANTELOPE can avoid or minimize the linear searches.

While there exist other available grouping tools, such as K-means [53] and Vector Space Model (VSM) [62], ANTELOPE leverages Latent Semantic Indexing (LSI) [54], [56] to aggregate semantically-correlated files due to its high efficiency and ease of implementation. K-means algorithm exploits multi-dimensional attributes of n items to cluster them into $K(K \leq n)$ partitions. While the iterative refinement minimizes the total intra-cluster variance that is assumed to approximately measure the cluster, the final results heavily depend on the distribution of the initial set of clusters. VSM is an algebraic model for representing document objects as vectors of identifiers. The grouping depends upon the assigned weights. VSM suffers from the scalability problem for long documents and fails to efficiently deal with the potential problems of synonymy and polysemy. The LSI tool overcomes these problems by using statistically derived concepts instead of terms for retrieval.

4.3 Insertion

When a data object is inserted into a group, the R-tree is adaptively adjusted to balance the workload among all storage nodes within this group. An insertion operation involves two steps: group location and threshold adjustment. Both steps only access a small fraction of the R-tree in order to avoid message flooding in the entire system.

When inserting a data object as a leaf node of the R-tree, we need to first identify a group that is the most closely related to this unit. Semantic correlation value between this new node and a randomly chosen group is computed by using LSI analysis over their semantic vectors. If the value is larger than admission threshold, the group accepts the data as a new member. Otherwise, the new data will be forwarded to adjacent groups for admission checking. After a data object is inserted into a group, MBR will be updated to cover the new node.

The admission threshold is one of the key design parameter to balance load among multiple storage nodes within a group. It directly determines the semantic correlation, membership, and size of a semantic-aware group. The initial value of this threshold is determined by sampling analysis. After inserting a new data object into a semantic group, the threshold is dynamically adjusted to keep the semantic-aware R-tree balanced.

4.4 Deletion

The deletion operation in the semantic-aware R-tree is similar to a deletion in a conventional R-tree. Deleting a given node entails adjusting the semantic correlation of that group, including the value of group vector and the multi-dimensional MBR of each group node. If a group contains too few storage nodes, the remaining nodes of this group are merged into its sibling group. When a group becomes a child node of its former grandparent in the semantic-aware R-tree as a result of becoming the only child of its father due to group merging, its height adjustment is executed upwardly.

4.5 Aggregate Queries

In order to efficiently support aggregate queries, we simplify the representation of the stored data and only keep partial precomputation results, i.e., semantically correlated subset. Correlated data often have higher probability to satisfy the query requests from adjacent data center nodes since they potentially keep approximate spatial and temporal localities. A query request with d-dimensional attributes can be

transformed into a vector q. The vector consists of a series of binary numbers, each of which denotes a dimension. We first hash the vector q into the LSH and obtain all data in the hit buckets. These data are much correlated and further organized into an R-tree structure [61]. We hence only need to select the data correlated with q by using LSH to build a lightweight and adaptive cube for aggregate queries.

We present an on-line ranking-based (top-k) aggregate query algorithm as shown in Fig. 8, which can return top-k query results. The algorithm first creates d_Q sorted lists for ranking cuboids (Line 1) and then initializes these sorted lists by scanning materialized cuboids into memory (Line 2). When all sorted lists are nonempty and the number of found results are smaller than k, ANTELOPE selects the cell with the largest aggregate value, represented as $\vec{c}$, according to the ranking function F^S. The cell $\vec{c}$ is then inserted into $Result$ and deleted from its sorted list as shown From Line 3 to Line 9. The cube finally returns top-k query results in Line 10.

4.6 Incremental Updates

A data center node usually maintains the messages to execute updates on the stale data. The messages contain new changes of system status. However, performing the updates potentially introduces extra re-computation overhead on ANTELOPE to guarantee query accuracy. ANTELOPE leverages the locality property within the received messages to carry out incremental updates. The main benefits are to provide quick response and reduce I/O costs. The basic idea behind incremental updates is to leverage multi-version based method and aggregate some amounts of received messages that exhibit the locality. The preprocess operations combine the same or similar messages that report approximate information into one update.

The multi-version based scheme in real system implementations of ANTELOPE offers cost-effective incremental update and supports the precomputation based queries. Specifically, at the beginning, the time is set to t_0. When the updated data arrive, ANTELOPE creates the versions for corresponding groups. From the times t_{i-1} to t_i, updates are aggregated into the i-th version. These updates contain the operations of insertion, deletion and modification of data, which are labeled in the versions.

In practice, the versioning scheme may incur extra overheads due to checking the attached versions, besides the original information when executing a query. However, since the versions only maintain the changes that essentially require small storage overheads, the extra latency of searching is usually small. A query operation needs to check the original data and its versions from t_i backward to t_0. The advantage of checking backward is to fast obtain the most recent changes since version t_i generally maintains the newer information than the version t_{i-1}.

ANTELOPE removes the attached versions when reconfiguring the original grouped data. The frequency of reconfiguration depends on the user requirements and environment constraints. Removing versions needs to apply the changes of the versions into its original data. To adapt to the system changes, ANTELOPE allows the groups to have different numbers and sizes of the attached versions.

5 Performance Evaluation

In this Section, we evaluate the performance of ANTELOPE in terms of its effectiveness and efficiency, including aggregate throughput in multiple communication patterns (one to one, one to many, many to one and all to all), initialization time, and online query service quality (query delay, query accuracy and update time).

5.1 Experimental Setup

The application scenario of ANTELOPE is in the large-scale data centers to support online services. We make use of real-world online query application datasets for our experiments. Due to space limitation, we mainly report the experimental results from HP [33] and MSN [34] traces, which represent the bounds as shown in Fig. 1 in Section 2.1.

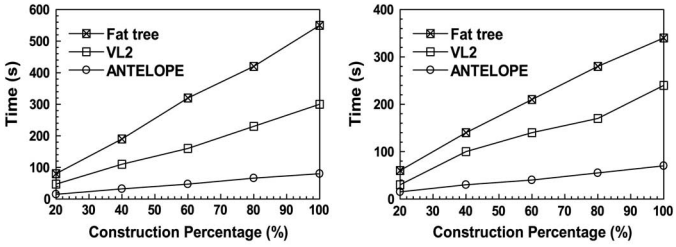
We have designed and implemented ANTELOPE prototype in the Linux environment. We built the testbed consisting of 30 servers, each of which is 2.0 GHz dualcore CPU, 2 GB DRAM, 250 GB disk and 1000PT quad-port Ethernet NIC. All servers are connected with 5 24-port Gigabit Ethernet switches. We use the parameters, $L = 8$, $\omega = 0.9$, $M = 12$ to support online aggregate queries.

In order to support the use of ANTELOPE, we leverage semantic grouping to aggregate correlated data together and store the computation results in the R-tree with the aid of structure mapping. Furthermore, the results of cube computation in the R-tree are distributed in multiple network nodes by using typical schemes of subtree partitioning [58], [63]. The basic idea of subtree partitioning is to allow each node to maintain one or multiple adjacent branches of entire tree. Moreover, in order to support efficient updates and fast queries, each node also stores the information of root node to locate the queried data. Hence, an update operation will incur local computation for a small branch of R-tree. If the new result leads to the modification of root node, it will be transmitted to other nodes for updates. The modification on the root node occurs with small probability in practice.

ANTELOPE stores the precomputed results of partial materialization to facilitate ranking-based (top-k) aggregate queries by setting specified I/O interfaces. When a server receives a query request, the request is then issued to cube interfaces to allow aggregate query operations in ANTELOPE. On the other hand, we set up to 2000 query requests for at most top-20 query results.

We compare ANTELOPE with state-of-the-art schemes, i.e., fat tree [21], VL2 [17] and pSearch [64]. Specifically, fat-tree is a switch-centric structure, motivated by reducing oversubscription ratio and removing single-failure points. Since switches are concatenated, the effective port number for scaling out is half (except the root layer). VL2 uses flat addressing to allow service instances to be placed anywhere in the network and further leverages the end-system based address resolution to scale to large server pools, without introducing the complexity to the network control plane. pSearch is a decentralized non-flooding P2P information retrieval system. By using Latent Semantic Indexing (LSI) to generate semantics, pSearch is able to distribute the indices through the P2P network. Due to aggregating semantically correlated items, the query costs can be reduced significantly.

ANTELOPE shares the similar design goals with fat tree, VL2 and pSearch. To facilitate the comparisons, we



(a) HP trace. (b) MSN trace.

Fig. 9. Construction time.

implement the components and functionalities in fat tree and VL2, not including their fault tolerance and fairness models, which are not the main concerns in this paper. Moreover, a pSearch prototype is constructed based on the guidance [64]. Since pSearch mainly aims to offer scalable and efficient query services, we compare ANTELOPE with pSearch in terms of online query quality as shown in Section 5.2.2. We also compute the VSM and SVD results in the system implementations [66], [67], [68].

It is worth noting that our comparison does not imply that other structures are not suitable for their original design scenarios. Instead, we intend to show that ANTELOPE is a better scheme for data centers that need to understand data to optimize network architecture design.

5.2 Experimental Results

We examine the performance of ANTELOPE by using several metrics and analyze the evaluation results.

5.2.1 Initialization

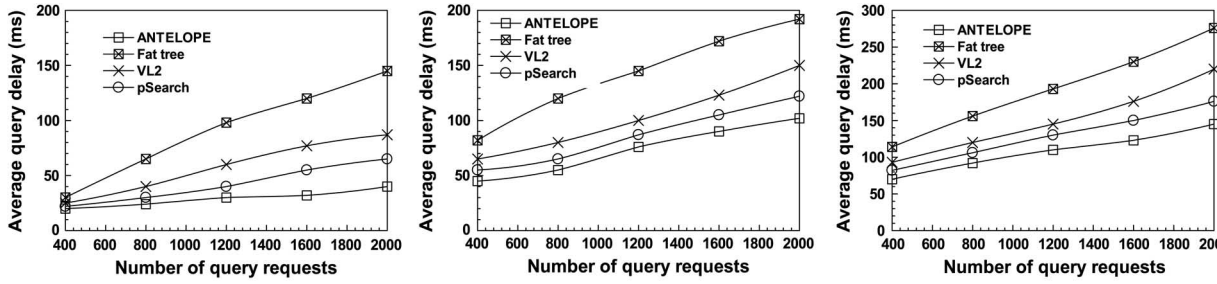
Cube-based designs need to initially carry out the precomputation on the combinations of multi-dimensional attributes, indicated by construction time as shown in Fig. 9.

We first examine the initial construction time as a metric to examine fat tree, VL2 and ANTELOPE schemes. The cube construction needs to first precompute the selected cuboids and then carry out the mapping operations from precomputed cube results to physical storage space. Fig. 9 shows that ANTELOPE using HP and MSN traces can save time on average 71.8% and 52.6% respectively over fat tree and VL2. The main reason is that ANTELOPE makes use of LSH to aggregate correlated data together. These correlated data serve for building cube with high probability and thus reduce potential data migration among multiple servers. In addition, we also observe that VL2 obtains better performance than fat tree due to the flat addressing in the former, which allows flexible data placement.

5.2.2 Online Query Service Quality

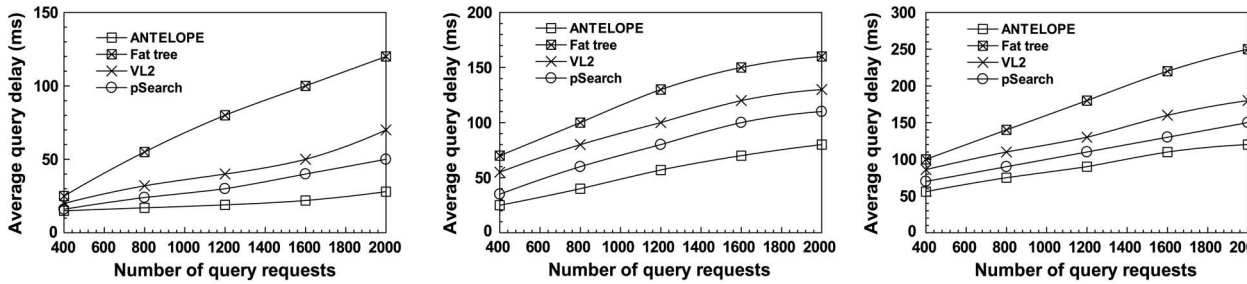
Query delay refers to the time interval from initiating the query request to receiving the results. Figs. 10 and 11 show the average query delays to respectively answer top-5, 10 and 20 queries in HP and MSN traces. We observe that ANTELOPE requires shorter latency, respectively by 35.6%, 47.2% and 65.8% in three queries, than pSearch, VL2 and fat tree. ANTELOPE uses LSH computation to quickly and accurately identify correlated data that can be indexed with high probability, which hence significantly decrease the searching space. In addition, pSearch leverages latent semantic indexing to generate the semantics that help narrow the searching scope and hence reduce query latency. Two typical traces exhibit the similar observations and conclusions.

Since ANTELOPE uses partial materialization to decrease the construction time and space overhead, it may potentially



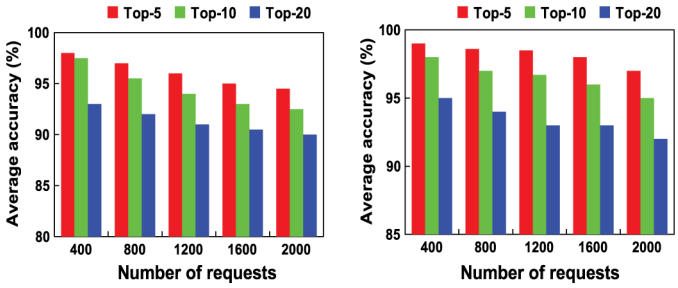
(a) Top-5 query. (b) Top-10 query. (c) Top-20 query.

Fig. 10. Average query delays for answering top-k queries in HP trace.



(a) Top-5 query. (b) Top-10 query. (c) Top-20 query.

Fig. 11. Average query delays for answering top-k queries in MSN trace.



(a) HP trace. (b) MSN trace.

Fig. 12. The accuracy of queries.

introduce some query inaccuracy. We evaluate the query accuracy by examining different top-k queries. Fig. 12(a) and 12(b) respectively show query accuracy of HP and MSN traces. We observe that more than 92.6% queries even considering 2000 requests for top-20 requests can obtain accurate results, meaning that there are one or two inaccurate items from 20 found ones, which can satisfy the query requirements. While considering the significant space savings and the decrease of construction time, the query accuracy in ANTELOPE can be acceptable by most online query applications.

We evaluate the dynamic update operations in ANTELOPE and VL2 as shown in Section 4.6 by examining the metric of incremental update time. The fat tree structure uses a central scheduler to collect flows whose last update is older than a given time and this reservation-based method may introduce more update latency. For updating stale data, VL2 sends the new data to a single directory server to determine the updated position, which introduces extra latency. Fig. 13 shows the execution time of updating different percentages of cuboids in ANTELOPE that leverages the locality characteristics. ANTELOPE in two traces requires on average 57.2% and 38.6% smaller update time than fat tree and VL2, efficiently supporting the operation of updating stale information.

5.2.3 Throughput

We perform experiments to demonstrate ANTELOPE's support for one-to-one, one-to-several, one-to-all and all-to-all patterns. The transmission data from source to destination servers are set to 20 GB. We further compare ANTELOPE with fat tree and VL2. The source server in the one-to-several pattern is to send data to three other servers through TCP connections. Table 3 shows the per-server throughput under different traffic patterns. We observe that ANTELOPE obtains around 2 times faster than fat tree and averagely 35.9% than VL2.

TABLE 3
Per-Server Throughput (Gb/s)

Patterns (HP/MSN)	Fat tree	VL2	ANTELOPE
One-to-one	1.26/1.31	1.65/1.67	2.17/2.21
One-to-several	0.98/1.12	1.43/1.55	1.82/1.96
One-to-all	0.61/0.73	1.15/1.22	1.53/1.68
All-to-all	0.37/0.49	0.62/0.78	1.06/1.26

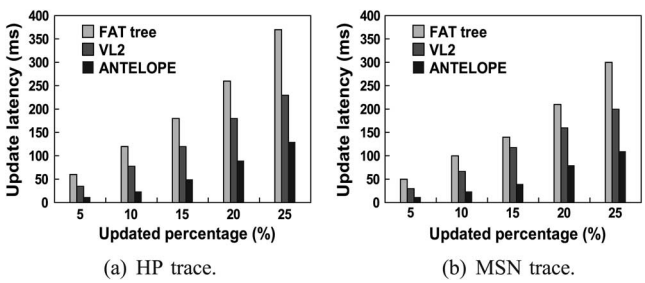
ANTELOPE is a scalable network scheme and we examine its per-server throughput by adding new servers. Fig. 14 shows the experimental results. We observe that ANTELOPE exhibits better scalability than VL2 and fat tree, respectively improving the throughput by averagely 0.86 and 2.17 times. ANTELOPE also shows it can obtain near-linear increase of throughput when adding new servers. The benefit mainly comes from the locality-aware design that allows most operations to be completed in the partial, not entire, networks.

6 Related Work

Data center architecture in the cloud is important to system performance. Existing network architectures focus on the study of scalability and fault tolerance, modular forms and inexpensive design.

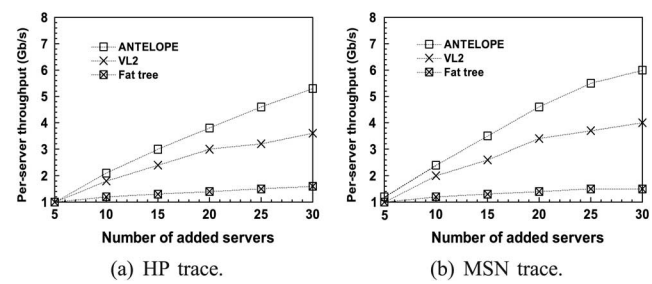
Scalability and fault tolerance. A data center should be scalable and fault tolerance. PortLand [15] is a scalable, fault tolerant layer 2 routing and forwarding protocol for data center environments. Ficonn [16] is a server interconnection network structure that uses dual-port configuration in data center servers. VL2 [17] leverages the programmability of servers and achieves hot-spot-free routing and scalable layer-2 semantics. DCell [18] proposes a dense interconnection network built by adding multiple network interfaces to servers that can forward packets. Since a high-level DCell is constructed from many low-level DCells, DCell can be recursively defined.

Modular design. The recently proposed shipping container data centers use a modular scheme to reduce the costs of cooling, powering and administration in a container. BCube [19] uses switches for faster processing and active probing for load-spreading. BCube supports various bandwidth-intensive applications by speeding up one-to-one, one-to-several, and one-to-all traffic patterns. To construct the inter-container structure and reduce the cabling complexity, MDCube [70] uses high-speed up-link interfaces of the commodity switches in BCube containers. Ripcord [71] is a platform for rapidly prototyping, testing, and comparing different data center networks. Ripcord offers a common infrastructure, and a set of libraries to allow quick prototyping of new schemes.



(a) HP trace. (b) MSN trace.

Fig. 13. Incremental update delay.



(a) HP trace. (b) MSN trace.

Fig. 14. Per-server throughput.

Furthermore, in order to build mega data center from heterogeneous containers, uFix [72] interconnects heterogeneous data center containers and can flexibly scale to large-scale servers. Moreover, a multi-class Bloom filter (MBF) [73] is proposed to support scalable data center multicast and considers element uncertainty. MBF determines the number of hash functions by considering the probability that a group is inserted into the Bloom filter.

Optimized costs. A high end router with more capacity is generally used to scale out the intermediate devices to a large number, thus requiring enormous costs. In order to improve the design of high bandwidth and multi-path data-center networks, Perseus [74] can optimize parameter choices from bandwidth, latency, reliability, parts cost, and other real-world details. SPAIN [20] provides multipath forwarding using inexpensive, commodity off-the-shelf (COTS) Ethernet switches. By exploiting the redundancy in a given network topology, SPAIN precomputes a set of paths and further merges them into a set of trees. Fat tree [21] presents a data center communication architecture that leverages commodity Ethernet switches to deliver scalable bandwidth for large-scale clusters.

7 Conclusion

Network architecture design is important in cloud data centers networks. In order to improve system efficiency and scalability, we need to study the network architecture and data placement, and bridge the gap between them. We present the design and implementation of ANTELOPE, a novel data-centric network scheme, for large-scale data centers. ANTELOPE explores and exploits the access patterns to identify locality-aware data with the aid of LSH that has constant-scale complexity. We further make use of an application, i.e., the precomputation-based data cube, to implement scalable distributed data placement and examine the real performance of ANTELOPE. ANTELOPE implements the partial materialization by leveraging the LSH computation, in which the correlated data are identified to build data cube. Extensive experimental results show the efficiency and scalability of our proposed ANTELOPE scheme.

Acknowledgments

This work was supported in part by National Natural Science Foundation of China (NSFC) under Grant 61173043; National Basic Research 973 Program of China under Grant 2011CB302301; NSFC under Grant 61025008, 61232004; Fundamental Research Funds for the central universities, HUST, under grant 2012QN098; the NSERC Discovery Grant 341823; US NSF under Grants NSF-IIS- 0916859, NSF-CCF-0937993, NSF-CNS-1016609 and NSF-CNS-1116606. The authors greatly appreciate anonymous reviewers for constructive comments.

References

[1] I.D.C. (IDC). (2013). "2010 Digital Universe Study: A Digital Universe Decade - Are You Ready" [Online]. Available: http://gigaom.files.wordpress.com/ 2010/05/2010-digital-universe-iview

[2] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. Sen Sarma, R. Murthy, and H. Liu, "Data warehousing and analytics infrastructure at Facebook," in *Proc. Special Interest Group Manag. Data (SIGMOD)*, 2010, pp. 1013–1020.

[3] Science Staff, "Dealing with data—Challenges and opportunities," *Science*, vol. 331, no. 6018, pp. 692–693, 2011.

[4] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[5] Hadoop. [Online]. Available: http://hadoop.apache.org/

[6] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," in *Proc. ACM Eur. Conf. Comput. Syst. (SIGOPS/EuroSys)*, 2007, pp. 59–72.

[7] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig Latin: A not-so-foreign language for data processing," in *Proc. ACM Special Interest Group Manag. Data (SIGMOD)*, 2008, pp. 1099–1110.

[8] A. Thusoo, J. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murthy, "Hive-a petabyte scale data warehouse using hadoop," in *Proc. Int. Conf. Data Eng. (ICDE)*, 2010, pp. 996–1005.

[9] G. Bell, T. Hey, and A. Szalay, "Beyond the data deluge," *Science*, vol. 323, no. 5919, pp. 1297–1298, 2009.

[10] J. Dai, J. Huang, S. Huang, B. Huang, and Y. Liu, "Hitune: Dataflow-based performance analysis for big data cloud," in *Proc. USENIX Annu. Tech. Conf.*, 2011, pp. 87–100.

[11] R. Katz, "Tech titans building boom," *IEEE Spectr.*, vol. 46, no. 2, pp. 40–54, Feb. 2009.

[12] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs, "Cutting the electric bill for internet-scale systems," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 123–134, 2009.

[13] C. Lu, G. Alvarez, and J. Wilkes, "Aqueduct: Online data migration with performance guarantees," in *Proc. Conf. File Storage Technol. (FAST)*, 2002, pp. 219–230.

[14] C. Pu and A. Leff, "Replica control in distributed systems: As asynchronous approach," *ACM SIGMOD Record*, vol. 20, no. 2, pp. 377–386, 1991.

[15] N. Mysore et al., "PortLand: A scalable fault-tolerant layer 2 data center network fabric," in *Proc. ACM Special Interest Group Data Commun. (SIGCOMM)*, 2009, pp. 39–50.

[16] D. Li, C. Guo, H. Wu, K. Tan, Y. Zhang, and S. Lu, "Ficonn: Using backup port for server interconnection in data centers," in *Proc. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, 2009, pp. 2276–2285.

[17] A. Greenberg, J. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, and S. Sengupta, "VL2: A scalable and flexible data center network," in *Proc. ACM Special Interest Group Data Commun. (SIGCOMM)*, 2009, pp. 51–62.

[18] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "DCell: A scalable and fault-tolerant network structure for data centers," in *Proc. ACM Special Interest Group Data Commun. (SIGCOMM)*, 2008, pp. 75–86.

[19] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "Bcube: A high performance, server-centric network architecture for modular data centers," in *Proc. ACM Special Interest Group Data Commun. (SIGCOMM)*, 2009, pp. 63–74.

[20] J. Mudigonda, P. Yalagandula, M. Al-Fares, and J. Mogul, "Spain: COTS data-center ethernet for multipathing over arbitrary topologies," in *Proc. USENIX Symp. Netw. Syst. Des. Implement. (NSDI)*, 2010, pp. 265–280.

[21] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. ACM Special Interest Group Data Commun. (SIGCOMM)*, 2008, pp. 63–74.

[22] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha, "Sharing the data center network," in *Proc. USENIX Symp. Netw. Syst. Des. Implement. (NSDI)*, 2011, pp. 309–322.

[23] K. Chen, C. Guo, H. Wu, J. Yuan, Z. Feng, Y. Chen, S. Lu, and W. Wu, "Generic and automatic address configuration for data center networks," in *Proc. ACM Special Interest Group Data Commun. (SIGCOMM)*, 2010, pp. 39–50.

[24] A. Viswanathan, A. Hussain, J. Mirkovic, S. Schwab, and J. Wroclawski, "A semantic framework for data analysis in networked systems," in *Proc. USENIX Symp. Netw. Syst. Des. Implement. (NSDI)*, 2011, pp. 127–140.

[25] S. Ghemawat, H. Gobioff, and S. Leung, "The Google file system," *ACM SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, p. 43, 2003.

[26] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber, "Bigtable: A distributed storage system for structured data," in *Proc. Symp. Oper. Syst. Des. Implement. (OSDI)*, 2006, pp. 205–218.

[27] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh, "Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals," *Data Mining Knowl. Discov.*, vol. 1, no. 1, pp. 29–53, 1997.

[28] R. Weber, H. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces," in *Proc. Int. Conf. Very Large Databases (VLDB)*, 1998, pp. 194–205.

[29] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proc. ACM Symp. Theory Comput.*, 1998, pp. 604–613.

[30] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," *Commun. ACM*, vol. 51, no. 1, pp. 117–122, 2008.

[31] Y. Hua, B. Xiao, D. Feng, and B. Yu, "Bounded LSH for similarity search in peer-to-peer file systems," in *Proc. Int. Conf. Parallel Process. (ICPP)*, 2008, pp. 644–651.

[32] Los Alamos National Lab (LANL). File System Data [Online]. Available: http://institute.lanl.gov/data/archive-data/

[33] E. Riedel, M. Kallahalla, and R. Swaminathan, "A framework for evaluating storage system security," in *Proc. Conf. File Storage Technol. (FAST)*, 2002, pp. 15–30.

[34] S. Kavalanekar, B. Worthington, Q. Zhang, and V. Sharda, "Characterization of storage workload traces from production windows servers," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, 2008, pp. 119–128.

[35] J. L. Hellerstein. (2010, Jan.). Google Cluster Data [Online]. Available: http://googleresearch.blogspot.com/2010/01/google-cluster-data.html

[36] B. Babcock, S. Chaudhuri, and G. Das, "Dynamic sample selection for approximate query processing," in *Proc. ACM Special Interest Group Manag. Data (SIGMOD)*, 2003, pp. 539–550.

[37] R. Missaoui, C. Goutte, A. Choupo, and A. Boujenoui, "A probabilistic model for data cube compression and query approximation," in *Proc. ACM Data Warehousing On-Line Anal. Process. (OLAP)*, 2007, pp. 33–40.

[38] J. Shanmugasundaram, U. Fayyad, and P. Bradley, "Compressed data cubes for OLAP aggregate query approximation on continuous dimensions," in *Proc. ACM Conf. Knowl. Discov. Data Mining (SIGKDD)*, 1999, pp. 223–232.

[39] D. Barbara and X. Wu, "Loglinear-based quasi cubes," *J. Intell. Inf. Syst.*, vol. 16, no. 3, pp. 255–276, 2001.

[40] T. Wu, D. Xin, and J. Han, "ARCube: Supporting ranking aggregate queries in partially materialized data cubes," in *Proc. ACM Special Interest Group Manag. Data (SIGMOD)*, 2008, pp. 79–92.

[41] D. Xin, J. Han, H. Cheng, and X. Li, "Answering top-k queries with multi-dimensional selections: The ranking cube approach," in *Proc. Int. Conf. Very Large Databases (VLDB)*, 2006, pp. 463–474.

[42] M. Riedewald, D. Agrawal, and A. El Abbadi, "pCube: Update-efficient online aggregation with progressive feedback and error bounds," in *Proc. Int. Conf. Sci. Stat. Database Manag. (SSDBM)*, 2000, pp. 95–108.

[43] W. Lu and J. Yu, "Condensed cube: An effective approach to reducing data cube size," in *Proc. Int. Conf. Data Eng. (ICDE)*, 2002, pp. 155–165.

[44] Y. Feng, D. Agrawal, A. El Abbadi, and A. Metwally, "Range cube: Efficient cube computation by exploiting data correlation," in *Proc. Int. Conf. Data Eng. (ICDE)*, 2004, pp. 658–669.

[45] X. Jin, J. Han, L. Cao, J. Luo, B. Ding, and C. Lin, "Visual cube and on-line analytical processing of images," in *Proc. 19th ACM Int. Conf. Inf. Knowl. Manag.*, 2010, pp. 849–858.

[46] P. Zhao, X. Li, D. Xin, and J. Han, "Graph cube: On warehousing and OLAP multidimensional networks," in *Proc. Special Interest Group Manag. Data (SIGMOD)*, 2011, pp. 853–864.

[47] B. Ding, B. Zhao, C. Lin, J. Han, and C. Zhai, "Topcells: Keyword-based search of top-k aggregated documents in text cube," in *Proc. Int. Conf. Data Eng. (ICDE)*, pp. 381–384, 2010.

[48] Y. Yu, C. Lin, Y. Sun, C. Chen, J. Han, B. Liao, T. Wu, C. Zhai, D. Zhang, and B. Zhao, "Inextcube: Information network-enhanced text cube," in *Proc. Int. Conf. Very Large Databases (VLDB)*, 2009, pp. 1622–1625.

[49] B. Bi, S. Lee, B. Kao, and R. Cheng, "Cubelsi: An effective and efficient method for searching resources in social tagging systems," in *Proc. Int. Conf. Data Eng. (ICDE)*, 2011, pp. 27–38.

[50] M. Liu, E. Rundensteiner, K. Greenfield, C. Gupta, S. Wang, I. Ari, and A. Mehta, "E-cube: Multi-dimensional event sequence processing using concept and pattern hierarchies," in *Proc. Int. Conf. Data Eng. (ICDE)*, 2010, pp. 1097–1100.

[51] J. Lee, S. Hwang, Z. Nie, and J. Wen, "Product entitycube: A recommendation and navigation system for product search," in *Proc. Demonstrations Int. Conf. Data Eng. (ICDE)*, 2010.

[52] G. Salton, A. Wong, and C. Yang, "A vector space model for automatic indexing," *Commun. ACM*, vol. 18, no. 11, pp. 613–620, 1975.

[53] J. Hartigan and M. Wong, "Algorithm AS 136: A k-means clustering algorithm," *Appl. Stat.*, pp. 100–108, 1979.

[54] S. Deerwester, S. Dumas, G. Furnas, T. Landauer, and R. Harsman, "Indexing by latent semantic analysis," *J. Amer. Soc. Inf. Sci.*, vol. 41, pp. 391–407, 1990.

[55] M. W. Berry, S. Dumas, and G. OBrien, "Using linear algebra for intelligent information retrieval," *SIAM Rev.*, vol. 37, pp. 573–595, 1995.

[56] C. Papadimitriou, P. Raghavan, H. Tamaki, and S. Vempala, "Latent semantic indexing: A probabilistic analysis," *J. Comput. Syst. Sci.*, vol. 61, no. 2, pp. 217–235, 2000.

[57] G. Golub and C. Van Loan, *Matrix Computations*. Baltimore, MD: Johns Hopkins Univ. Press, 1996.

[58] Y. Hua, H. Jiang, Y. Zhu, D. Feng, and L. Tian, "Semantic-aware metadata organization paradigm in next-generation file systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 2, pp. 337–344, Feb. 2012.

[59] C. Tang, S. Dwarkadas, and Z. Xu, "On scaling latent semantic indexing for large peer-to-peer systems," in *Proc. ACM SIGIR Conf. Res. Develop. Inf. Retriev.*, 2004, pp. 112–121.

[60] S. Lee, S. Chun, D. Kim, J. Lee, and C. Chung, "Similarity search for multidimensional data sequences," in *Proc. Int. Conf. Data Eng. (ICDE)*, 2000, pp. 599-608.

[61] A. Guttman, "R-Trees: A dynamic index structure for spatial searching," in *Proc. ACM Special Interest Group Manag. Data (SIGMOD)*, pp. 47–57, 1984.

[62] G. Salton, A. Wong, and C. Yang, "A vector space model for information retrieval," *J. Amer. Soc. Inf. Retriev.*, vol. 18, pp. 613–620, 1975.

[63] S. Weil, S.A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proc. Symp. Oper. Syst. Des. Implement. (OSDI)*, 2006, pp. 307–320.

[64] C. Tang, Z. Xu, and S. Dwarkadas, "Peer-to-peer information retrieval using self-organizing semantic overlay networks," in *Proc. ACM Special Interest Group on Data Commun. (SIGCOMM)*, 2003, pp. 175–186.

[65] Z. Xu, C. Tang, and Z. Zhang, "Building topology-aware overlays using global soft-state," in *Proc. Int. Conf. Distrib. Comput. Syst. (ICDCS)*, 2003, pp. 500–508.

[66] C. Buckley, "Implementation of the smart information retrieval system," Dept. Comput. Sci., Cornell Univ., Tech. Rep. TR 85-686, 1985.

[67] M. W. Berry, "Large-scale sparse singular value computations," *Int. J. Supercomput. Appl.*, vol. 6, no. 1, pp. 13–49, 1992.

[68] G. H. Golub and C. Reinsch, "Singular value decomposition and least squares solutions," *Numer. Math.*, vol. 14, no. 5, pp. 403–420, 1970.

[69] L. De Lathauwer, B. De Moor, and J. Vandewalle, "A multilinear singular value decomposition," *SIAM J. Matrix Anal. Appl.*, vol. 21, no. 4, pp. 1253–1278, 2000.

[70] H. Wu, G. Lu, D. Li, C. Guo, and Y. Zhang, "MDCube: A high performance network structure for modular data center interconnection," in *Proc Int. Conf. emerg. Netw. Exp. and Technol. (CoNEXT)*, 2009, pp. 25–36.

[71] M. Casado, D. Erickson, I. A. Ganichev, R. Griffith, B. Heller, N. Mckeown, D. Moon, T. Koponen, S. Shenker, and K. Zarifis, "Ripcord: A modular platform for data center networking," EECS Dept., Univ. of California, San Diego, CA, Tech. Rep. UCB/EECS-2010-93, 2010.

[72] D. Li, M. Xu, H. Zhao, and X. Fu, "Building mega data center from heterogeneous containers," in *Proc. IEEE Int. Conf. Netw. Protocols (ICNP)*, 2011, pp. 256–265.

[73] D. Li, H. Cui, Y. Hu, Y. Xia, and X. Wang, "Scalable data center multicast using multi-class bloom filter," in *Proc. IEEE Int. Conf. Netw. Protocols (ICNP)*, 2011, pp. 266-275.

[74] J. Mudigonda, P. Yalagandula, and J. C. Mogul, "Taming the flying cable monster: A topology design and optimization framework for data-center networks," in *Proc. USENIX Annu. Tech. Conf.*, 2011.

Yu Hua received the BE and PhD degrees in computer science from the Wuhan University, China, in 2001 and 2005, respectively. He is an associate professor at the Huazhong University of Science and Technology, China. His research interests include computer architecture, cloud computing, and network storage. He has more than 50 papers to his credit in major journals and international conferences including *IEEE Transactions on Computers* (TC), *IEEE Transactions on Parallel and Distributed Systems* (TPDS), USENIX ATC, INFOCOM, SC, ICDCS, ICPP, and MASCOTS. He has been on the program committees of multiple international conferences, including INFOCOM, ICPP and IWQoS. He is a senior member of the IEEE, and a Member of ACM and USENIX.

Xue Liu received the BS degree in mathematics and the MS degree in automatic control both from Tsinghua University, Beijing, China. He received the PhD degree in computer science from the University of Illinois at Urbana-Champaign in 2006. He is an associate professor in the School of Computer Science at McGill University, Montreal, Quebec, Canada. His research interests are in computer networks and communications, smart grid, real-time and embedded systems, cyber-physical systems, data centers, and software reliability. His work has received the Year 2008 Best Paper Award from *IEEE Transactions on Industrial Informatics*, and the First Place Best Paper Award of the ACM Conference on Wireless Network Security (WiSec 2011). He serves as an associate editor of the *IEEE Transactions on Parallel and Distributed Systems* and editor of the *IEEE Communications Surveys & Tutorials*. He is a member of the ACM and IEEE.

Hong Jiang received the BSc degree from the Huazhong University of Science & Technology (HUST), Wuhan, China, in 1982, the MASc degree from the University of Toronto, Ontoria, Canada, in 1987, and the PhD degree from the Texas A&M University, College Station, in 1991. He is Willa Cather Professor at the University of Nebraska-Lincoln. His research interests include computer architecture, computer storage systems and parallel/distributed computing. He serves as an associate editor of the *IEEE Transactions on Parallel and Distributed Systems*. He has more than 200 publications in major journals and international conferences in these areas, including IEEE-TPDS, IEEE-TC, USENIX-ATC, ISCA, MICRO, FAST, ICDCS, IPDPS, SC, ICS, HPDC, etc. He is a senior member of the IEEE and a member of the ACM and ACM SIGARCH.