2017 IEEE 33rd International Conference on Data Engineering

Feisu: Fast Query Execution over Heterogeneous Data Sources on Large-Scale Clusters

An Qin*, Yuan Yuan†, Dai Tan*, Pengyu Sun*, Xiang Zhang*, Hao Cao*, Rubao Lee†, Xiaodong Zhang†
*Baidu, Inc, {qinan, tandai02, sunpengyu, zhangxiang02, caohao}@baidu.com
†The Ohio State University, {yuanyu, liru, zhang}@cse.ohio-state.edu

***Abstract*—Fast data analytics at an increasingly large scale has become a critical task in any Internet service company. For example, in Baidu, the major search engine company in China, large volumes of Web and business data in PB-scale are timely and constantly acquired and analyzed for the purposes of evaluating product revenue, tracking product demanding activities on market, predicting user behavior, upgrading product rankings, and diagnosing spam cases, and many others. Response time for queries of various data analytics not only affects user experiences, but also has a serious impact on productivity of business operations.**

In this paper, to meet the challenge of fast data analytics, we present Feisu (meaning fast in Chinese), a data integration system over heterogeneous storage systems, which has been widely used in Baidu's critical and daily business analytics applications after our R&D efforts. Feisu is designed and implemented to co-work together with several heterogeneous storage systems, and exploit the query similarity embedded in complex query workloads. Our experiments using real world workloads show that Feisu can significantly improve query performance in Baidu. Feisu has been in production use in Baidu for two years to effectively manage over dozens of petabytes of data for various applications.

I. Introduction

In the big data era, it is a challenging and timely task to support various interactive data analytics jobs over increasingly large scale data sets that are archived and managed by different types of storage systems. The search engine of Baidu [1], the leading Chinese online Web service provider serving billions of users, is such a system platform to address these challenges in its daily operations. A large number of Baidu services, such as Web search, map and navigation, cloud service and online encyclopedia, require fast accesses over huge amounts of data sets. In practice, strategy engineers need to quickly model new product ideas, diagnose and optimize their models based on information sources collected from multiple systems; data engineers periodically query dozens of terabytes of user business log data to produce statistical reports; system engineers have to figure out malfunctions hidden in the search engine in petabytes-scale data sets. Processing various types of interactive data analysis jobs in such a complex environment in an efficient way has become increasingly demanding.

In the Baidu search engine, data sets are generated from a variety of sources and managed by different business-specific storage systems (e.g., local files system, HDFS [2] and cold-data distributed filesystem [3]). An interactive data analytics task may access data sources from all these systems, but cannot affect the service quality of any business application running on top of these systems. For this reason, it is unrealistic for Baidu to adopt commonly used systems of general purposes, such as Hadoop [4], Apache Hive [5], Apache Spark [6], and Apache Impala [7] to process these tasks.

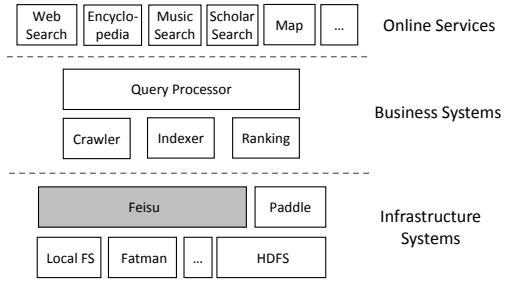


Fig. 1: Baidu's software stack

To make interactive data analytics tasks highly efficient in large scale and heterogeneous cluster systems, we have designed and implemented Feisu, a columnar data processing system that is optimized for geo-distributed storage systems. Feisu handles the geographical distribution via the cross-domain mechanism to share the data schema and access rights. It organizes data sets into partitions using a compression-friendly columnar format. To speedup ad-hoc queries, Feisu introduces an effective adaptive indexing mechanism, called *SmartIndex*, which utilizes query similarity behind our internal user's behaviors. SmartIndex can effectively reduce data retrieval time without affecting the service quality of different storage systems.

Feisu has been in production use in Baidu for two years, supporting more than one hundred products. Figure 1 illustrates Feisu's position in the Baidu's software stack. From rapid product prototyping to revenue reporting, from deep learning model training to system malfunction debugging, Feisu has significantly improved the efficiency of data exploration and supports a large number of products. Now more than fifteen thousand of Feisu instances have been deployed across six data centers, serving five thousands of queries on average every day. The total size of data managed by Feisu has reached dozens of petabytes.

In general, we have made the following contributions in this paper:

- We show the necessity to build an independent data processing system over heterogeneous data sources, such as Feisu.
- We identify query locality and similarity based on analysis of query characteristics in Baidu, which can be used to significantly improve query performance.

2375-026X/17 $31.00 © 2017 IEEE
DOI 10.1109/ICDE.2017.162

- We present the detailed design and implementation of Feisu, a parallel columnar data processing system with an adaptive indexing mechanism over heterogeneous storage systems.
- We evaluate Feisu with real-world workloads and illustrate our experiences in building Feisu, which would shed some light on data management systems on large clusters.

The rest of the paper is organized as follows. Section II explains Baidu's data heterogeneity and the motivation of Feisu. Section III presents the overall design of Feisu. Section IV describes Feisu's optimizations based on our analysis of user logs. After describing implementation issues in building Feisu in Section V, We evaluate the performance of Feisu in Section VI. Section VIII introduces the related work and Section IX concludes the work.

II. Baidu's Requirements in Distributed Data Integration

In Baidu, data are generated from a variety of sources (see the pipeline of web data processing in Figure 2), which can be categorized into the following three types.

- Log data, which include user activity logs and system running logs, are generated on tens of thousands of online service machines.
- Business data, which include web links, web pages, and indices (web page's digest, index and inverted index), are generated by offline processes cross different clusters.
- Labeled data, generated by engineers for model training or other tasks, such as ranking enhancement or bad case intervention.

These data are usually stored on different storage systems in Baidu. Log data are stored in the local file systems of the online machines; business data are mainly stored in the global file systems (e.g., HDFS); and labeled data can be stored in key-value stores or Baidu's internal distributed file system Fatman [3]. Many data analytics tasks need to access data from these storage systems. We use the following three application cases in Baidu to illustrate the characteristics of these tasks and motivate our work.

Case 1: Debugging search engine.

Debugging search engine is a common but challenging task for system engineers to find and solve problems (e.g., inefficient designs, data inconsistency issues and etc.) in Baidu's system stack. Usually engineers need to access data that may be placed in multiple data sources such as local filesystems and standalone distributed filesystems to find the the root cause of a problem. The process is expensive and error-prone because engineers must learn and understand the complex data flows involving different storage systems. Data inconsistency and schema mismatching can happen across storage systems, which significantly delays the engineers from giving insights.

Case 2: Rapid product prototyping.

Rapid product prototyping is important for testing new ideas and launching new products. Before Feisu, a large portion of product engineers' time is spent on preparing data sets from several application storage systems. They have to learn the interfaces of these storage systems and collaborate with system engineers to figure out the necessary data sets to use for product prototyping. One-round of the data preparation process would cost almost one week. Product prototyping usually takes multiple rounds of data preparation and evaluation no matter whether the new product is finally proved to be launched. For example, in our design of voice search product, we need to evaluate the benefited user domain so we have to extract the user behavior data again and again to demarcate the specific user set. Moreover, data freshness is very important for new product research and development. The delay of product prototyping caused by the complexity of different storage systems can result in business losses.

Case 3: Product Analysis.

Product analysis is critical to figure out the user and business trends for our products. Some analysis tasks need to process past one-or-two years of data to analyze the industry tendency or to analyze specific product activities. In this case, historical data stored in cold storage will be loaded together with the latest data for computation. It is also common for engineers to periodically analyze sampled hot data to check the indicators in a revenue report. These tasks access different storage systems and have different performance requirements.

A straightforward approach to solve the problems is to deploy one global file system (e.g., HDFS) to manage these data or loading the data from different storage systems into one storage system and then run standard data processing engines such as Hadoop and Spark for data analytics. However, this approach doesn't work for Baidu.

Deploying one global file system (e.g., HDFS) to manage all data is impossible in Baidu because of the different service requirements for different applications. For example, log data are stored on online service machines, which run legacy retrieval service for Baidu search engine. The retrieval service is critical to the search engine and thus other heavy services such as global file system cannot co-run on the same nodes.

It is also impossible to load the data from these different storage systems to one global file system for data analytics due to the high cost of storage and network data transferring, which is caused by two facts: (1) Internet search engine is usually geo-distributed; and (2) data are generated in a high speed in Baidu (e.g., 2.3 GB log data per hour per node). Collecting the data into one storage system will consume large amounts of network resources, which can severely affect the business service traffic in hot regions. This approach is also inefficient because analytic tasks cannot access the data in a timely manner to generate product insights, which may block fast product development [8].

As a result, we need to develop a new data processing system that efficiently manage and query all the data to meet Baidu's needs. There are two major requirements for the new system. First, the system needs to consider data heterogeneity and different service level requirements when accessing the data on different storage systems. Each storage system works in an independent domain. Data on different systems have different storage layouts, and cannot be shared among systems. The new system must efficiently track the data information

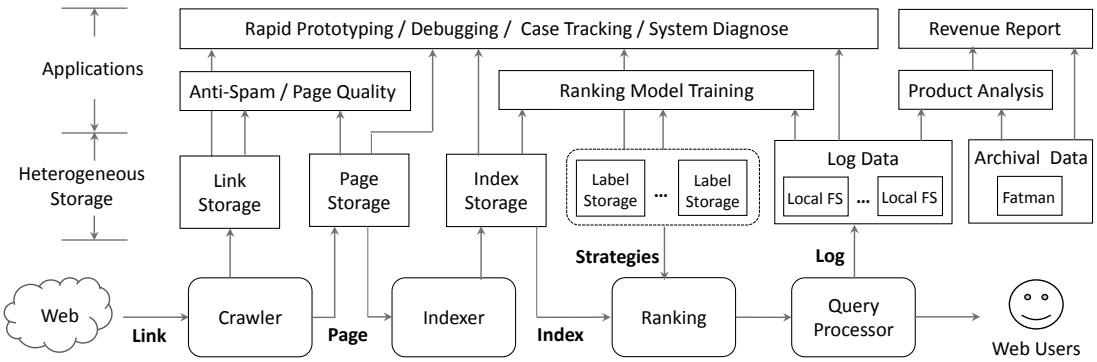


Fig. 2: Data pipeline in Baidu's search engine

across different storage systems. Second, the system should execute queries fast enough to meet the business requirements. We observe that users in Baidu access different types of data differently, which can help design the data access method on different systems. For example, for log data, many analytic applications only need partial information hidden in the data. In this case, we can design a light weight process running on these nodes for information extraction.

In the following sections, we will describe our design of Feisu, a columnar data processing system optimized for heterogeneous storage systems.

III. Feisu Design Overview

A. Data Model and Query Model

Feisu stores data in columnar format, similar to other high performance data processing engines (e.g., [9], [10]). The reason is that data in Baidu usually contain hundreds of attributes but only a small subset of them are actually queried. Storing data in columnar format can effectively reduce the I/O cost. Feisu also supports nested data format such as *json*, which will be flatten into columns when the data are processed.

Currently Feisu supports star schema like queries in the following format.

```
SELECT expr1 [[AS] expr_alias1] [...]
  [aggr_func(expr3) WITHIN expr4]
  FROM table1 [, table2, ...]
  [[INNER|[RIGHT|LEFT] OUTER|CROSS] JOIN
                  table3 [[AS] table_alias3]
    ON join_cond_1 [AND join_cond_N ...]]
  [WHERE cond]
  [GROUP BY (field1 | alias1) [...]]
  [HAVING cond]
  [ORDER BY field1|alias1 [DESC|ASC] [...]]
  [LIMIT n]
  ;
```

B. System Architecture

Figure 3 shows the architecture of Feisu. Feisu's design is motivated by the architecture of modern web search engine. As can be seen, Feisu organizes servers in the cluster using the tree structure. There are three types of servers in the tree, the master server, the stem server and the leaf server. The master server is the root of the tree, which accepts ad-hoc queries from clients and generates optimized query execution plans using a cost-based approach. It dissects a query plan into sub-plans based on the information of available stem servers and dispatch the sub-plans to them. The stem server is the internal node in the tree. It accepts a sub-query plan from the master and further dissects the plan to the leaf servers. The leaf server is the leaf in the tree, which actually executes the query plan and computes the results. After the results are generated, they are summarized in a bottom-up way and sent back to the clients.

Feisu's tree-structured organization can efficiently utilize available computing resources. To maximize the performance, Feisu schedules a query based on data location, the cluster's network structure, and the load statistics on the leaf servers. Feisu always schedules a task to the leaf server that contains the data if the server are available. If the leaf server is not available, Feisu will either schedule the task to the available leaf server that contains the data replica or to an available server that has a low network transfer overhead based on the network structure.

To support heterogeneous storage systems, each storage node in a specific storage system is deployed a light-weight process, which monitors the storage for newly generated data (e.g., log data) and converts the data into Feisu in columnar format when new data arrive. Each storage node acts as a leaf server in Feisu. To guarantee that Feisu doesn't affect the business critical applications on top of the storage system, Feisu controls the amount of resources that can be used for query processing on the leaf server. The resources will always be granted to business-critical applications first.

C. System Implementation

Master. The Feisu's master is a key service and is built with the following main components:

- Job manager. Job manager maintains the running information of user query jobs. When users submit a query, job manager will analyze query execution semantics and verify accessed right of specific data set. If the request is legal, job manager will create an execution plan based on data partition information and cluster utilizations. After that, a new job with dozens of tasks is created. Before the new job is put into a candidate job queue, job manager tries to reuse other running job's task result if tasks are identical.

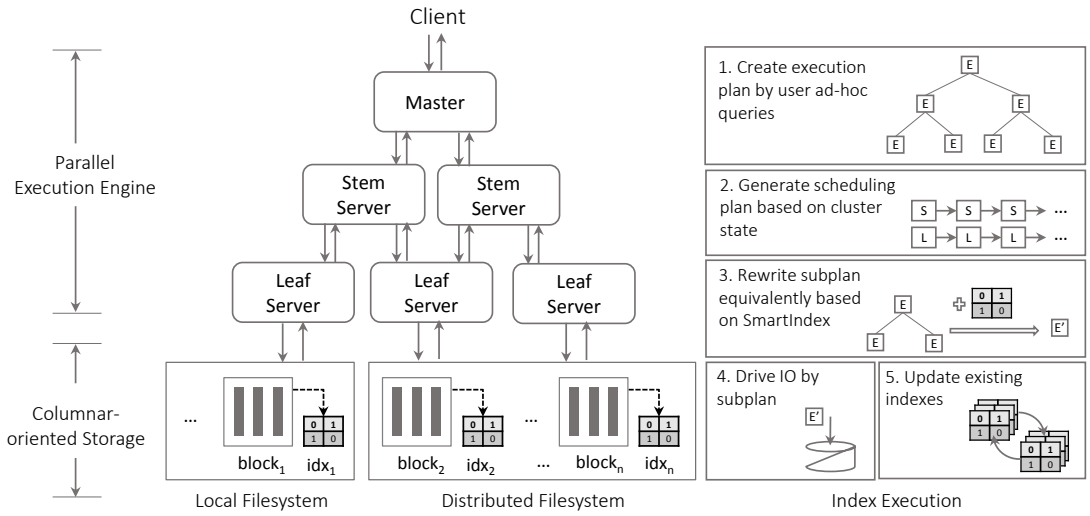


Fig. 3: Feisu's architecture

- Cluster manager. Cluster manager manages runtime information of workers (i.e. leaf server and stem server). It communicates with the job manager using periodic RPC to keep information updated. Feisu does not adopt systems like Zookeeper [11] for survival detection because the number of workers is too large and the workers are geographical distributed.
- Job Scheduler. Job Scheduler creates the scheduling plan for candidate jobs after collecting the information from job manager and cluster manager. Then candidate jobs with scheduling plans will be put into an emitting queue, and tasks will be dispatched to corresponding workers.
- Entry Guard. It is the entry point of whole system, executing the security checking of access flows and dispatching the incoming traffics. It is also responsible for capability protection to avoid malicious attacks.

To support large clusters that contain tens of thousands of nodes, master components are implemented as services, communicated via an RPC channel and deployed in standalone machines. With a large number of workers in the cluster, the network connection for worker heartbeat will reach the upper limit of a single machine. Our design of separated cluster management components can easily solve this issue by horizontal-scaling the cluster manager. Job manager is designed in a similar way.

For reliability, components (the primary) are running with backups, which don't provide service until the primary ones crash. The backup components get checkpoint and operations log from the primary in realtime, so that they will reach the same running state as the primary. Since the backup ones are shadows of the primary, they can provide functionalities such as monitoring running information to reduce the burdens on the primary.

Stem and Leaf servers. Feisu's workers can work as both stem servers and leaf severs, which is determined when the service starts. Stem servers aggregate task execution results generated by leaf servers or by other stem servers. The data flow information is defined in the execution plan, which is sent to all related workers.

Feisu achieves task fault-tolerance using backup tasks, which are activated by job scheduler when status update of a specific task times out or service crash is detected. Feisu schedules backup tasks considering data locations, response times, and workloads on candidate workers. To enhance interactive response, user can optionally configure the processed ratio of total data sets to avoid long-tail influence, or directly limit the total elapse time of whole query running. If processed data ratio reaches or response time limitation is elapsed out, the whole job will abandon the unfinished tasks (including backup tasks) and return back.

Client. The client-end is a versatile component with pluggable framework to support command-line tool, website-based service, and third-party tools. It has two major functionalities: query syntax checking and access right verification. Query syntax checking verifies syntax legitimacy and guides users to write the proper SQL-like query command. Access right verification checks user identity, accessed resource right and quota before submitting a query to the servers. The client-end also collects user query histories to personalize data indexing and caching. Differently from the query collection in master component, collection on the client side is used for *SmartIndex* to build private index for specific users or user groups.

Common Storage layer. The common storage layer provides unified data view for other components. To make heterogeneous storage systems work together smoothly, all data files are given full paths with prefix flags to activate different storage plugins. For example, the file path in Hadoop filesystem will be "/hdfs/path/to/filename", and in Fatman filesystem the path will be "/ffs/path/to/filename". If a prefix string can not be recognized, local filesystem is activated by default. Besides unifying naming, Single-Sign-On (SSO) is implemented to allow cross-domain access all storage systems in Feisu, by mapping their authentication information to running

job credential.

IV. System Optimization

In this section we present our optimizations in Feisu based on the study of a two-month query log traces after Feisu was deployed in Baidu. We find that queries show an access pattern of similarities in a short time span. Based on this discovery, we implement a caching mechanism in Feisu and design SmartIndex, which utilizes the evaluation results of query predicates to improve the performance of later queries.

A. Query Similarity and Data Locality

We collected a two-month user query log and analyzed the query features in the log. We find that in a short time period: (1) a small set of query columns are likely to be repeatedly accessed (i.e. data locality); and (2) a small set of query predicates are likely to be reused (i.e. query similarity).

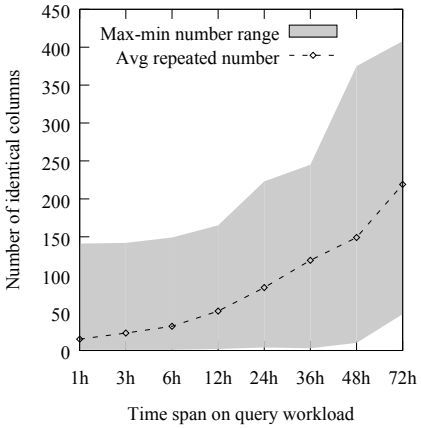


Fig. 4: Number of accessed identical columns with different time spans

Data Locality. We split the query log traces based on fixed time span (e.g., 1-hour, 2-hour) and analyzed the number of repeated accessed columns in the time span. The results are shown in Figure 4. As can be seen, there is a small set of columns that are repeatedly accessed in a given time span. The number increases when the time span becomes larger. This indicates that Feisu can cache the hot columns to improve the query performance.

Query Similarity. Similar to the analysis of data locality, we also split the query log traces based on fixed time spans and analyzed the characteristics of query predicates in the log. The reason is that when executing a query plan, each leaf server in Feisu only executes a sub-plan. In this case, the evaluation of query predicates has a significant impact on the data access performance. In the analysis the predicates are converted to the conjunctive form. Figure 5 shows the ratio of queries that have at least one exact same query predicate with different time spans. As can be seen, in a given time span, a large number of queries have at least one same query predicate. This indicates that we can improve Feisu's performance if the evaluation of hot query predicates can be optimized.

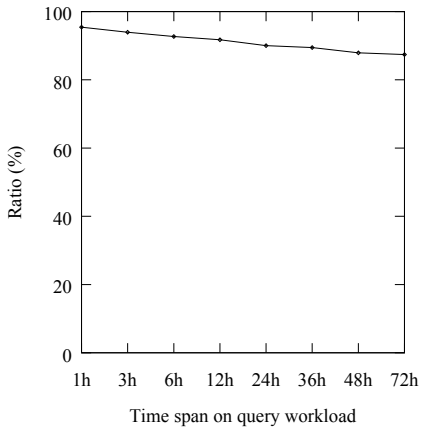


Fig. 5: Ratio of queries that have at least one same query predicate with different time spans

Data locality and query similarity reflect some representative access patterns of Feisu users. Human users usually explore the data in a trial-and-error approach. For example, when diagnosing bugs, a user is likely to first issue an aggregation query without query predicates and then add predicates one by one based on the query results. In this case, same columns will be repeatedly accessed and query predicates are repeatedly evaluated.

Next, we describe how we utilize data locality and query similarity to improve Feisu's query performance.

B. Data Cache

To utilize data locality, we implement a cache layer in Feisu's storage system using SSDs. The SSD cache is managed using LRU. Currently not all query's data will be cached. Only data that may be accessed by critical business applications can utilize SSDs. We manually set the cache preferences for different data based on practical knowledge.

The reason is that even if hot columns exist in user queries, it is very difficult to keep the hot data in SSD cache because of the large amounts of ad-hoc queries. We evaluated different cache management methods, all of which incur more than 80% of cache miss rates. Without a manual interference, SSD resources would not be efficiently used in Feisu.

C. SmartIndex

To utilize query similarity, we design SmartIndex for efficiently reusing the evaluating results of query predicates. Each SmartIndex is a 0-1 vector, which stores the evaluation results of a query predicate. Feisu stores all SmartIndices in the memory of leaf servers. In this case, the leaf servers can avoid accessing the data and evaluate the query predicate if a corresponding SmartIndex exists.

1) Index Format: The SmartIndex is stored in the corresponding leaf server's memory in a format similar to the bitmap index, as shown in Figure 6. It stores the evaluation results of a query predicate and meta information of the corresponding data in Feisu's storage. Feisu can compress the index to improve memory efficiency.

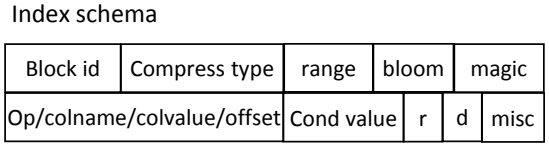


Fig. 6: The structure of SmartIndex

2) Index Management: Feisu creates a SmartIndex each time a query predicate is evaluated in a leaf server. Feisu manages the indices based on the size of the cache memory in the leaf servers and the time the index has been in the cache since creation. An index will be deleted from the cache if: (1) the cache memory is full (by a LRU based approach); or (2) the index has been in the cache for too long. Current the Time-To-Live (TTL) for each index is set to 72 hours based on our experiences.

Feisu also provides interfaces for users to set preferences and retire strategies on indices to increase the possibility that they are cached in the memory for better performance. For example, indices with preferences can remain in the memory when their TTL expire if the cache memory is not full.

3) Query Execution with SmartIndex : With SmartIndex, Feisu's leaf servers will transform the predicates in query sub-plans into conjunctive forms and check if there exist a SmartIndex for each data block it will process. If a SmartIndex exists, the scan of the data block and the evaluation of the predicate are avoided, which can significantly improve the throughput. Otherwise the data are read from the storage system and a new SmartIndex will be created.

```
Q1: SELECT COUNT(*) FROM T
          WHERE (c2 > 0) AND (c2 <= 5)
```

Figure 7 shows how an aggregation query with two query predicates is executed on a leaf server when the indices for both predicates exist. In this case, all computations are conducted in memory. No scan operation is actually needed.

V. Other Implementation Issues

A. Authentication and Authorization

Feisu manages heterogeneous storages, each of which is in an independent storage domain. Each domain has its own access control and resource management strategy. To grant Feisu to access data on each storage system, each system must support Single-Sign-On (SSO) access [12]. Authentication and Authorization (AA) are offline executed on X509-based certification system [13][14]. To make the authentication process transparent, we implement the AA functionalities as standard PAM plugins in the storage system to quickly support SSO access [15][16][17]. To guarantee that Feisu doesn't affect the service quality of the business application on top of each storage system, we define a resource consumption agreement between Feisu and each storage system. Each storage system must synchronize its agreement to Feisu such that Feisu doesn't over-schedule tasks to the storage system.

B. Hardware Resource Utilization Control

Hardware has become increasingly powerful in recent years, which makes it possible for each machine to serve more than one application. To increase hardware utilization, we consolidate servers by deploying multiple containers in the same sever [18][19]. All containers are generally scheduled by our central cluster manager. It is inevitable that some of low-priority containers have to give up resources to guarantee the provision of high-priority online services or for balancing the workload via instance migration. This affects system throughput and latency, particularly for storage systems. In this case, Feisu's design of task scheduling and fault tolerance must consider the fluctuation and avoids the temporary unavailability via dynamically communicating (e.g., heartbeat) with the cluster manager. Sometimes the cluster manager may not timely detect server crashes. To reduce the impact on performance, Feisu communicates with applications to schedule more resources.

C. Traffic Flow Management

Traffic control is important for Feisu's performance because of the large amounts of severs in the cluster. Feisu divides traffic control into the following three types: control and state flow, write date flow and read data flow. The first type is control and state flow, which includes control information such as cluster-level operation commands and heartbeat. The control flow has the highest priority because these information items are used to control the cluster state in Feisu. Besides the communication priority designed in Feisu system, we also activate the type-of-service (TOS) flag in switch device to reserve bandwidth for priority communication. The second traffic type is write data flow. Although queries on Feisu are read-only, Feisu still needs to write data (e.g., temporary data and intermediate results) during query execution. These written data are transmitted in a bypass channel to a global distributed storage. The third traffic type is the read data flow for collecting back the analyzed data. If the data are too big, it will be dumped to global storage and only the location information is passed. Read data flow has the lowest priority in Feisu because the cost of read bandwidth is cheaper than write flow and re-try mechanism is more acceptable and flexible when data are stored on persistent storage with replicas.

VI. Evaluation

A. Experiment Setup

We have measured the experiments on one of online clusters in Baidu, which contains 4,000 nodes. Each node in the cluster is equipped with a 4-core 2.4 GHz Xeon processor, 64 GB of memory, four 3-TB SATA disks, and one 500GB SSD disk. All nodes are connected through 1 Gbps full-duplex Ethernet. In the experiments, Feisu uses 512 MB of memory by default to store SmartIndex. The cluster has two HDFS storage systems managed by Feisu. Each data block in the storage system has three replicas.

The experiments use three datasets based from real-world applications, as shown in Table I. Datasets *T1* and *T2* are from user business log data with the same schema, carrying URL-clicked information and query attributes. Dataset *T3* is from a sample set of traced webpage URLs downloaded from Baidu's

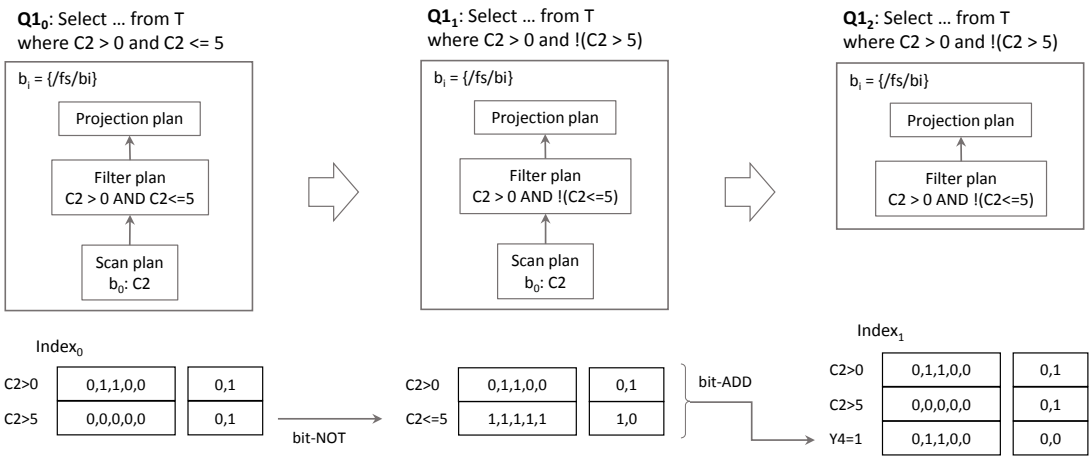


Fig. 7: Plan transformation and index calculation during query execution

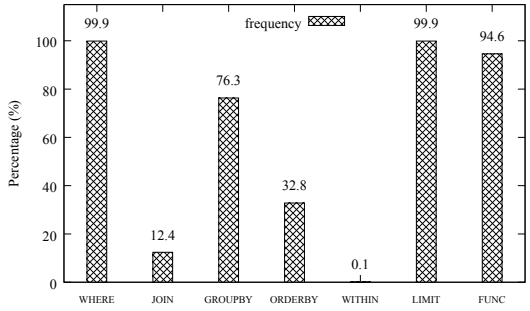


Fig. 8: Keyword frequency

webpage database. *T3*'s attributes are a subset of *T1*'s attributes and *T2*'s attributes.

TABLE I: Experimental datasets

Table name	Number of records	Uncompressed Size	Number of fields	Storage
T1	30 billion	62 TB	200	A
T2	130 billion	200 TB	200	B
T3	10 billion	7 TB	57	A

We use scan query to evaluate Feisu's performance. The reason is that scan queries (including aggregation) are most frequent in Feisu, based on our analysis of a three-month user query log, as shown in Figure 8. They occupy more than 99% of all queries in Feisu and thus it is important to show how they perform in Feisu.

B. Performance Results

1) Scan Performance on One Storage System: The workload we use to evaluate the scan performance is in the following format.

```
SELECT a FROM T1
  WHERE b OP1 value1 [[AND | OR] c OP2 value2]
  (OP is comparison operator, or CONTAINS)
```

We randomly generate the query parameters to run the query on Feisu. For a comparison, we also implemented B-tree index in Feisu. The results are shown in Figure 9.

Figure 9(a) shows the query performance with SmartIndex and without SmartIndex. This demonstrates SmartIndex's efficiency because query performance improves as more queries are processed by Feisu. When the number of queries processed goes above 4,000, the performance is improved by more than 3x compared to the case when SmartIndex is disabled. The performance improvement comes from SmartIndex's reduction of I/O when a query predicate has SmartIndex.

SmartIndex performs differently compared to B-tree index, as shown in Figure 9(b). The query performance when using B-tree index remains almost constant as more queries are processed by Feisu, but it is not as effective as SmartIndex because SmartIndex not only reduces I/O but also the computation execution time for predicate evaluation.

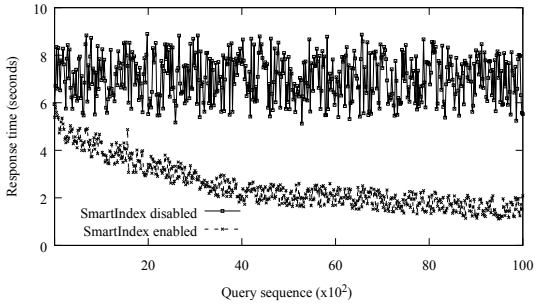
2) Scan Performance on Multiple Storage Systems: The scan queries used in these experiments are in the same format as previous experiments. The only difference is that each scan query in the current experiments will scan both T2 and T3, which are stored on different storage systems. Recall that T2's attributes are a subset of T3's attributes.

```
SELECT a FROM T
  WHERE b OP1 value1 [[AND | OR] c OP2 value2]
  (OP is comparison operator, or CONTAINS)
```

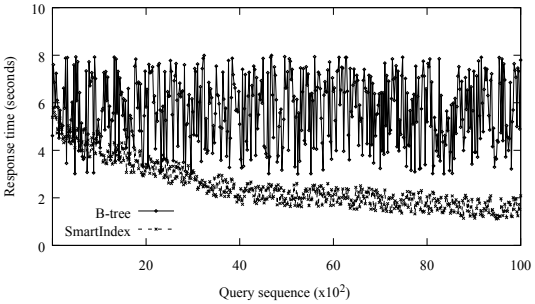
We randomly generate query parameters to run the query on Feisu. We measure the averaged scan throughput of each server in the cluster and the results are shown in Figure 10.

As can be seen, after SmartIndex is enabled, the averaged throughput on a single server can be improved by up to 1.5x. This is constant with previous experiments because SmartIndex can effectively reduce I/O and computation.

3) The Impact of Memory Size on the Performance of SmartIndex: Feisu's SmartIndex is stored in the memory of each leaf server. The memory size on each server used by Feisu significantly affects the overall throughput. To illustrate the impact, we use the workload for evaluating scan performance



(a) Performance ith and without SmartIndex



(b) Comparison of SmartIndex and B-tree

Fig. 9: Scan performance on one storage system

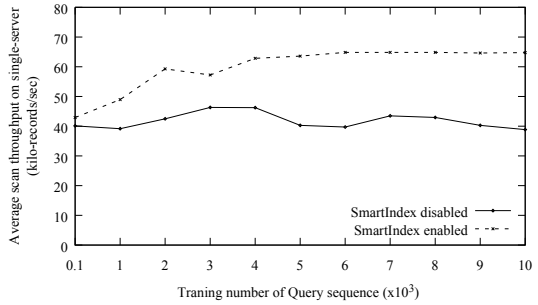


Fig. 10: Averaged scan throughput of a single server on different storage systems

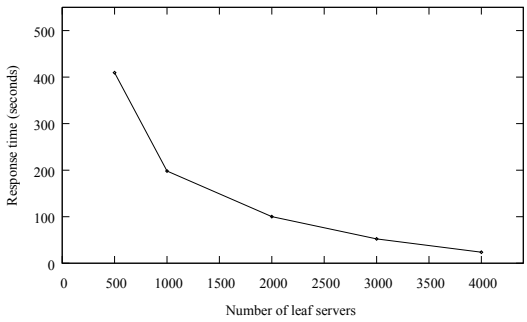


Fig. 12: Response time with different number of nodes

on multiple servers and measure its performance when varying the size of memory on each server used by Feisu. The results are shown in Figure 11.

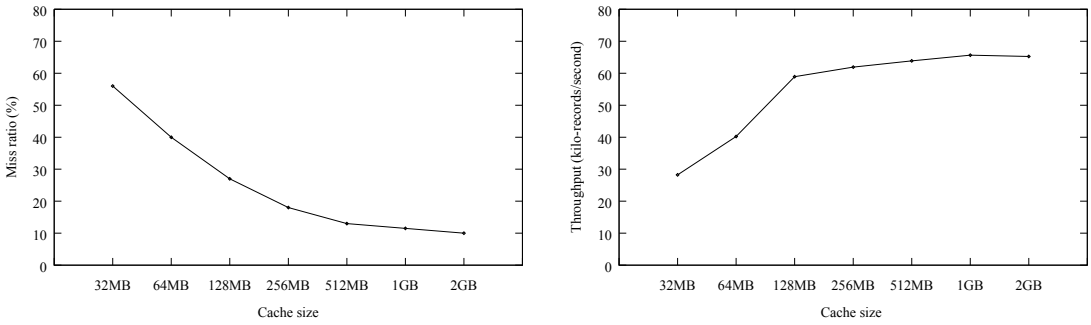
As expected, Feisu's performance increases with the increase of available memory on each server. This is what we have expected since the more indices be loaded into cache, the better the performance will be. Another observation we can obtain is that the performance of Feisu with 512 MB memory is comparable to that with 2GB memory. This shows the effectiveness of Feisu's index management, and Feisu doesn't consume too much memory on each server.

C. Scalability

We use the workloads in the previous experiment and measure its performance using different number of nodes in the cluster to evaluate Feisu's scalability. The results are shown in Figure 12. As can be seen, Feisu's performance increases linearly with the number of nodes. This is contributed by Feisu's scale-out design and indicates Feisu's strong scalability.

VII. Online Service in Production System

Feisu has been used for large number of data analytics applications for more than two years. In the earlier period, its deployment was only on several single clusters with around five hundreds of machines and several hundreds TB of data sets. Soon, data sharing among these clusters requires Feisu to be able to cross-storage data processing. Therefore, the common storage layer is developed to deal with the issue. As data sets become more and more huge, the response latency of Feisu engine would be hard to meet user requirement under the limited resources. Thus, we have encapsulated Feisu worker in container and elastically launched them in the online service machine to share their idle CPU and memory resources. When the number of worker reaches more than five thousands, master's memory becomes rare resources. Hence, we separated job manager from master as standalone service to guarantee its resource provision. One year ago, Feisu was used to analyze the full log data of whole search service. Data growth requires more resources put into Feisu to meet the response latency, and the increasing number of workers continues put new challenges to Feisu's architecture. Especially when the worker number reaches eight thousands, the network overhead of internal communication (e.g. heartbeat, task dispatch) began affecting external user experience (job submission, monitoring information access, etc.). Feisu's master was too busy in handling the internal requests and can not respond to user's request in time. So we have to further separate another components of scheduler and cluster management from master and make each of them more scalable. Another challenge in full-log analytics is that data can not be copied to central global file system because we don't have such big global filesystem to hold dozens of petabytes of data. Designing proper protocols to manage the data location in each disks is a non-trivial task. Currently Feisu's protocol only carries the location information (e.g. , file name, and offset) during computation to narrow down the data workload. Feisu has already released eight major versions during the past two years, and has witnessed the above



(a) Miss ratio with different memory size

(b) Throughput with different memory size

Fig. 11: The impact of memory size on Feisu's performance

evolution.

Currently, Feisu system has been deployed with about tens of thousands of workers, managing dozens of petabytes of data cross mostly product data. The total number of product using Feisu as data analytics platform is above one hundred. The number of users has doubled in the past half year.

- Above 150 users access the system in recent half year averagely. Most of them are doing the rapid prototyping and product analytics. The total number of queries in one day can reach six thousands. Compared to Hadoop MapReduce framework, no training is needed for users to exploit the data treasure.
- More than 93% queries focus on those data sets are less than 200 TB. And, their response times are always below 20 seconds. Most of these queries simply focus on statistic applications by filtering specific columnar data. It is thus useful to find the similarity in specific query sequence.
- Product iteration productivity has been improved dramatically. The product iteration period has been decreased from months to days; product analysis time to figure out problems has reduced from days to hours.

VIII. Related Work

There are a number of recent big data systems developed for large-scale data analysis running on large clusters. MapReduce-based systems [20], such as Apache Pig [21], Apache Hive [22], had been widely used in the Hadoop ecosystem. However, due to high overheads of starting and executing MapReduce jobs, these MapReduce-based systems are more suitable for batch processing instead of satisfying the requirements of interactive data analysis [10]. Shark [23] and Spark SQL [24] are recent SQL engines running on Apache Spark [25], which is a general cluster computing engine optimized for in-memory computing and iterative algorithms. There are also other big data analytic systems which do not need a MapReduce-like general engine [26], such as Dremel [10], Impala [27], HAWQ [28],Presto [29], and VectorH [30]. These systems executes SQL queries directly on cluster file systems (e.g., Hadoop File System) in order to support interactive data analysis. The major difference between our solution Feisu and these existing systems is that Feisu is mainly designed for the purpose of data integration in order to provide a unified data view for various developers and engineers in the team of Baidu search engine. To the best of our knowledge, none of the above mentioned systems can be directly used to address our challenges.

Traditional database systems have been extended to provide data integration functionalities, such as IBM DB2 [31], Microsoft SQL Server [32], and PostgreSQL [33]. A recent data integration system is the Data Tamer System [34], which focuses more on the schema mapping [35] issue across heterogeneous data sets. It is unclear whether these data integration systems can run on a large cluster with thousands nodes, such as ours in Baidu. More importantly, unlike our solution Feisu, these systems do not provide execution optimizations based on query similarity that is an important feature and optimization opportunity in our scenario. In addition, the request window technique for data integration [36] exploits the request locality [37] to combine a group of similar remote queries. Different from such a batch-processing technique, Feisu exploits the idea of query result reusing to accelerate executions of similar queries.

IX. Conclusion

Interactive data analysis over heterogeneous data sources has become important to improve the quality of the Baidu search engine. Due to the cost of storage and networking, distributed data sets cannot be collected into a single system in a conventional warehousing way. A data integration solution not only requires the ability of handling heterogeneous challenges in the storage layer, but also needs minimized intervenes applied to the data sources on which other production workloads are co-running. Addressing these challenges, we have designed and implemented Feisu, a SQL query engine with columnar storage across heterogeneous storage systems. A key optimization in Feisu is to exploit the opportunity of query similarity in user queries and accordingly utilize a SmartIndex technique to accelerate query executions.

Feisu has been widely used in Baidu's critical business running on more than tens of thousands of cluster nodes. We believe this work also benefits to system researchers and practitioners on data management in large clusters.

X. Acknowledgments

We thank the following people: Haifeng Wu and Jianfeng Zhan gave us valuable suggestions. Ran Zheng and Ketie Cen directed our integration test in Baidu's biggest cluster management system. Zheng Li took change in the online system operation, and cooperated with Ying Lian and Luming Cai on the testing framework. The team at The Ohio State University was supported in part by the National Science Foundation under grants OCI-114752, CNS-1162165, and CCF-1513944.

References

[1] "Baidu, inc." http://www.baidu.com.

[2] "Hadoop distributed filesystem," http://hadoop.apache.org/docs/stable/hdfs_design.html.

[3] A. Qin, D. Hu, J. Liu, W. Yang, and D. Tan, "Fatman: Cost-saving and reliable archival storage based on volunteer resources," *Proceedings of the VLDB Endowment*, vol. 7, no. 13, pp. 1748–1753, 2014.

[4] "Apache hadoop," http://hadoop.apache.org.

[5] "Apache hive," https://hive.apache.org.

[6] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 2–2. [Online]. Available: http://dl.acm.org/citation.cfm?id=2228298.2228301

[7] "Apache impala," http://impala.apache.org.

[8] P. Pedreira, C. Croswhite, and L. Bona, "Cubrick: Indexing millions of records per second for interactive analytics," *Proceedings of the VLDB Endowment*, vol. 9, no. 13, 2016.

[9] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, P. O'Neil, A. Rasin, N. Tran, and S. Zdonik, "C-store: A column-oriented dbms," in *Proceedings of the 31st International Conference on Very Large Data Bases*, ser. VLDB '05. VLDB Endowment, 2005, pp. 553–564. [Online]. Available: http://dl.acm.org/citation.cfm?id=1083592.1083658

[10] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis, "Dremel: interactive analysis of web-scale datasets," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 330–339, 2010.

[11] "Apache zookeeper," http://zookeeper.apache.org.

[12] A. Pashalidis and C. Mitchell, "A taxonomy of single sign-on systems," in *Proc. ACISP'03*, 2003, pp. 249–257.

[13] F. W. Housley R., Polk W. and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile [RFC 3280]," 2002.

[14] V. Welch1, I. Foster, C. Kesselman, O. Mulmo, L. Pearlman, S. Tuecke, J. Gawor, S. Meder, and F. Siebenlist, "X.509 Proxy Certificates for Dynamic Delegation," in *3rd Annual PKI R&D Workshop*, 2004.

[15] A. Qin, H. Yu, C. Shu, and B. Xu, "Xos-ssh: A lightweight user-centric tool to support remote execution in virtual organizations," in *Proc. First USENIX Workshop on Large-Scale Computing (Lasco'07)*, 2008.

[16] A. Qin, H. Yu, C. Shu, X. Yu, Y. Jegou, and C. Morin, "Operating system-level virtual organization support in xtreemos," in *Proceedings of the 9th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'08)*, 2008, pp. 234–243.

[17] V. Samar, "Unified login with pluggable authentication modules (PAM)," *Proceedings of the 3rd ACM conference on Computer and communications security*, pp. 1–10, 1996.

[18] S. Soltesz, H. Potzl, M. E. Fiuczynski, A. C. Bavier, and L. L. Peterson, "Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors," in *EuroSys'07*, 2007, pp. 275–287.

[19] "Kenel cgroups," http://www.mjmwired.net/kernel/Documentation/cgroups/.

[20] A. Floratou, U. F. Minhas, and F. Özcan, "Sql-on-hadoop: Full circle back to shared-nothing database architectures," *Proceedings of the VLDB Endowment*, vol. 7, no. 12, pp. 1295–1306, 2014.

[21] A. F. Gates, O. Natkovich, S. Chopra, P. Kamath, S. M. Narayanamurthy, C. Olston, B. Reed, S. Srinivasan, and U. Srivastava, "Building a high-level dataflow system on top of map-reduce: the pig experience," *Proceedings of the VLDB Endowment*, vol. 2, no. 2, pp. 1414–1425, 2009.

[22] Y. Huai, A. Chauhan, A. Gates, G. Hagleitner, E. N. Hanson, O. OMalley, J. Pandey, Y. Yuan, R. Lee, and X. Zhang, "Major technical advancements in apache hive," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 2014, pp. 1235–1246.

[23] R. S. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica, "Shark: Sql and rich analytics at scale," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '13. New York, NY, USA: ACM, 2013, pp. 13–24. [Online]. Available: http://doi.acm.org/10.1145/2463676.2465288

[24] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi *et al.*, "Spark sql: Relational data processing in spark," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015, pp. 1383–1394.

[25] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 2–2.

[26] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[27] M. Kornacker, A. Behm, V. Bittorf, T. Bobrovytsky, C. Ching, A. Choi, J. Erickson, M. Grund, D. Hecht, M. Jacobs *et al.*, "Impala: A modern, open-source sql engine for hadoop." in *CIDR*, 2015.

[28] L. Chang, Z. Wang, T. Ma, L. Jian, L. Ma, A. Goldshuv, L. Lonergan, J. Cohen, C. Welton, G. Sherry *et al.*, "Hawq: a massively parallel processing sql engine in hadoop," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 2014, pp. 1223–1234.

[29] "Presto query engine." https://prestodb.io/.

[30] A. Costea, A. Ionescu, B. Răducanu, M. Switakowski, C. Bârca, J. Sompolski, A. Luszczak, M. Szafrański, G. de Nijs, and P. Boncz, "Vectorh: Taking sql-on-hadoop to the next level," in *Proceedings of the 2016 International Conference on Management of Data*, ser. SIGMOD '16. New York, NY, USA: ACM, 2016, pp. 1105–1117. [Online]. Available: http://doi.acm.org/10.1145/2882903.2903742

[31] V. Josifovski, P. Schwarz, L. Haas, and E. Lin, "Garlic: A new flavor of federated query processing for db2," in *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '02. New York, NY, USA: ACM, 2002, pp. 524–532. [Online]. Available: http://doi.acm.org/10.1145/564691.564751

[32] J. A. Blakeley, C. Cunningham, N. Ellis, B. Rathakrishnan, and M.-C. Wu, "Distributed/heterogeneous query processing in microsoft sql server," in *21st International Conference on Data Engineering (ICDE'05)*. IEEE, 2005, pp. 1001–1012.

[33] R. Lee and M. Zhou, "Extending postgresql to support distributed/heterogeneous query processing," in *International Conference on Database Systems for Advanced Applications*. Springer, 2007, pp. 1086–1097.

[34] M. Stonebraker, D. Bruckner, I. F. Ilyas, G. Beskales, M. Cherniack, S. B. Zdonik, A. Pagan, and S. Xu, "Data curation at scale: The data tamer system." in *CIDR*, 2013.

[35] R. J. Miller, L. M. Haas, and M. A. Hernández, "Schema mapping as query discovery." in *VLDB*, vol. 2000, 2000, pp. 77–88.

[36] R. Lee, M. Zhou, and H. Liao, "Request window: an approach to improve throughput of rdbms-based data integration system by utilizing data sharing across concurrent distributed queries," in *Proceedings of the 33rd international conference on Very large data bases*. VLDB Endowment, 2007, pp. 1219–1230.

[37] R. Lee and Z. Xu, "Exploiting stream request locality to improve query throughput of a data integration system," *IEEE Transactions on Computers*, vol. 58, no. 10, pp. 1356–1368, 2009.