

# A Review of Some Analytic Architectures for High Volume Transaction Systems

Robert Grossman  
Open Data Group and  
University of Illinois at Chicago

## 1. INTRODUCTION

In this paper, we review some of the different architectures that are used for analytics for high volume transaction streams. Although experts have used these approaches to build systems for over 15 years, they are not known as widely as they should be.

- We assume that our application fits into the following analytic model:
- The data source is a stream of transactions, or what we more abstractly call *events*.
- The event data is processed to produce derived data of various types that we call *features* and that features are organized into *profiles* or *signatures* (we use the terms interchangeably) about entities of interest. More precisely, profiles are vectors of features that are indexed by keys associated with entities of interest.
- By analytics we mean applying statistical or data mining models to profiles to produce scores.
- We assume that the scores are further processed to produce decisions about the entities of interest.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DM-SSP'07, August 12, 2007, San Jose, California, USA.  
Copyright 2007 ACM 978-1-59593-838-1/07/0008 ...\$5.00.

Some examples of applications that fit this analytic model are in Table 1. A good motivating example is to assume that the events are payment card transactions, the profiles are associated with the payment card holders, and the statistical models are fraud models.

From the view point of this paper, we are interested in several types of analytic architectures. The first dimension of interest is the size of the data: Some analytic architectures are designed to work with data that fits into memory; others are designed to work with data managed by a database; while others are designed to work with data too large to be easily managed by a database. In this case, some type of indexed file system is often used. The second dimension of interest is how distributed the data is: some architectures are designed to work with data managed by a single workstation; some with data distributed in a local cluster or grid; and others with data distributed over a wide area cluster or grid.

The amount of data that can be managed in memory and the amount of data that can be managed by standard commercial databases increases each year. At the time this paper was written (2007), a basic workstation can be configured for 32 GB of memory, while more specialized computers can be configured for larger amounts. Today, managing more than several hundred Terabytes of data with standard commercial databases can be quite challenging.

In Section 2 we discuss background material and related work. In Section 3 we discuss indexed file systems. In Section 4 we discuss integrated architectures. Section 5 is the conclusion.

Events	Entity	Profiles	Statistical Models	Decisions
Credit card transactions	Card holder account number	Profiles about account holders	Fraud models	Approve/decline credit card transaction
Phone calls	Phone number of caller	Profiles associated with callers	Fraud models	Whether the caller is making fraudulent calls
Purchase of an item	The person buying the item	Statistical profile of the buyer and his/her interests	Recommendation models	Additional items to suggest for possible purchase
Entering a query into a search engine	The person searching the web	Profiles about searchers	Response models	Which ads to place on a web page; in which order to place the ads

**Table 1. Some examples of events, entities and profiles.**

## 2. BACKGROUND AND RELATED WORK

From the perspective of this paper, there are two important transitions as the size of the event data increases. The first transition is when the event data is so large that it does not fit into the memory of a computer. The second transition is when the event data is so large that it cannot be managed conveniently by a database.

These days clusters and distributed clusters of commodity computers (which we call *nodes* as usual) are quite common. When the data is too large to fit in the memory of a single node, one option is to develop middleware that can manage in-memory data that is distributed across the nodes in a cluster or distributed cluster. Similarly, when the data is too large to be managed by a single database, one option is to develop (or use middleware that someone else has developed) that can manage databases associated with each node in a cluster or distributed cluster. In both cases, the design of the middleware is considerably simplified if we assume a shared nothing architecture, that is an architecture in which a computer's memory or disk is not shared with other nodes in the cluster. We note that the arguments and architecture below can be modified in the obvious way if this type of middleware is used.

It is instructive to begin with some history of communities that have had to work with very large data sets.

The high energy physics community was one of the first scientific communities with the need to develop an infrastructure for very large data sets. Beginning around

1990, the design of an infrastructure to handle the data for the Superconducting Super Collider (which was never actually built) was begun. The collider was expected to produce about a Petabyte of data a year, at a time when databases could not scale to a terabyte. A community of high energy physicists and computer scientists over a period of a two to three years worked out a few basic principles that are just as useful and relevant today as when they were first articulated in the early 1990's [Baden:1995]:

1. Use shared nothing parallelism. High energy physics data is organized into data records called events, representing possible collisions of high energy particles. Clusters of commodity workstations were just emerging in 1990 and it quickly became clear that a shared nothing architecture in which events were distributed over nodes in a cluster provided a very scalable architecture for event-based processing of data.
2. Optimize for efficient reading of columns not safe writing of rows. Databases are optimized for safe writing of transactions. In contrast, reporting and data analysis of large data sets is better served by an architecture that is optimized for efficient reading of attributes. Since relational data is usually viewed as rows (corresponding to transactions) and columns (corresponding to attributes), this is insight is sometimes expressed as saying that the data management infrastructure should be optimized for the efficient *reading of columns* versus *the safe writing of rows*.

These principles were well established by about 1994 [See for example, Bailey-1995, Brun-1997]. Interesting enough, it took quite some time for these principles to be recognized by the database community [Stonebraker-2005], despite the fact that there were column-oriented databases in the market at the time, such as Sybase-IQ [Rennhackkamp-1997].

A third principle also emerged:

3. When possible, employ a database to manage the data. The challenge was that the physics event data was too large to be managed easily by the databases available at that time. Three approaches emerged. The first approach was to process the data to produce intermediate data sets that were small enough to be managed by either a single database or a collection of databases, one for each node in a shared nothing cluster. This approach required middleware to scatter the data and collect the results of the per node computation. The second approach was to develop more scalable data management systems. With this approach, the data management systems could also be designed to support column-wise operations on data [Bailey-2005].

A good illustration of Principle 3 is the data management infrastructure built to manage the multi-terabyte Sloan Digital Sky Survey data [Gray-2002]. At the beginning of the project, it is probably fair to say that many of the astronomers were a bit surprised by how efficient standard commercial databases were at managing their data.

Closely related to Principle 3 is the following Principle 4:

- When databases did not scale sufficiently use an indexed file system. By an indexed file system, we mean an application that can manage a collection of files plus a mechanism for dereferencing a data record or data object to obtain the name of the file and the offset within the file of the data. Indexed file system can organize data into files either by row, by column or by some mixture of rows and columns. To achieve scalability indexed file systems break data into segments and store each segment in a separate file. Specifically segments consist of either chunks of columns or collections of rows, depending upon how the data is physically laid out on disk. To achieve greater scalability, segments may be distributed across nodes in a cluster or distributed cluster using a shared nothing architecture. Indexed file systems proved important for many applications since it is much easier to develop an indexed file system than it is to develop a SQL-database [Bailey-2005].

Perhaps the best known and most scalable indexed file system supporting rows and columns of data is Google's BigTable [Ghemawat-03], Chang-06].

We conclude this section by noting that some specialized techniques were developed during this period or earlier that are still relevant today:

- Compressing event data prior to storing it in a database. A related technique is to query the compressed data without decompressing it. An example of a database using this technique was AT&T's Daytona database for call detail records events [AT&T-07].
- Using specialized indexing methods to manage column-wise data, such as bit indexes [Rennhackkamp-1997].
- With large data sets, the number of distinct values of attributes is usually quite a bit smaller than the number of data records, except for some obvious exceptions such as keys. More precisely, given  $N$  records, an attribute might only assume  $n$  values where  $n \ll N$ . For this reason, coding these  $n$  values using the integers  $1-n$  can often significantly improve the overall end-to-end efficiency despite the extra dereference required every time an attribute is accessed.

### 3. INDEXED FILE SYSTEMS

In this section, we collect together information about indexed file systems and their applications for building statistical models.

We begin by recalling the definition of an indexed file system. From the point of view here, think of a file system as an application in which files can be accessed by specifying a directory and the name of a file in the directory.

(directory-name, file-name)

Operations on files include creating, writing and reading files. In an indexed file system, files contain records that are indexed so that they may be accessed by key:

(directory-name, file-name, record-key).

Dereferencing a record-key to the required off-set in the file is usually done using an auxiliary file that contains an ISAM or other index file [Ramakrishnan-2003]. An alternative is to provide access to attributes by specifying both the record key and attribute name or key:

(directory-name, file-name, record-key, attribute-key)

In addition to file operations on indexed files, operations on records in indexed file systems include reading, writing and updating records by index. In general, queries are limited to accessing records by index or by range query, although more complicated queries may be supported.

An important element of the design of an indexed file system is how records are physically laid out on disk. There are two main alternatives.

- Records may be arranged on disk by row.
- Records may be arranged on disk by column.

In addition, some combination may be used. With the decreasing cost of storage, it is sometimes useful to write out data in multiple formats to improve access performance.

Generally, if data is arranged on disk by row, then rows are further divided into units, that we may call row segments. Similarly, if data is arranged on disk by columns, then single columns are usually further divided into units that we may call column segments. Finally, segments may be grouped so that they may be managed together into *segment families*.

Here are some common variants:

- Keys may be integers or strings.
- Segments or families of segments may be compressed. Over the past several years, the power of CPUs has steadily increased, while the bandwidth to access disks has only modestly improved. For this reason, compressing data prior to writes and decompressing data after reads can speed up the overall throughput to disk.
- Segments may be replicated to provide safety in case of failure and to improve locality of access for distributed systems. Although this complicates writes, since indexed file systems are generally targeted to applications that require efficient reads, this is usually a good trade-off.
- Some attributes may be encrypted to preserve privacy. As mentioned above, with the power of CPUs steadily increasing, encrypting sensitive information prior to writes is often a good trade-off.

Here are some examples:

- Storing data in files and accessing it using auxiliary indexes such as ISAM is a standard technique [Ramakrishan-2003].
- Google's BigTable [Chang-06] is designed to handle sparse data and uses strings as row and column keys. BigTable also uses a time stamp as a key.
- The open source Hadoop Distributed File System (HDFS) is another example of a system that provides indexed access to data [Hadoop-2007].
- The open source Augustus system also stores data in an indexed file system [Augustus-2007].

We close this section by listing several advantages that indexed files have over databases when working with very large data sets:

- Indexed file systems are more scalable than databases. This is a direct consequence of the fact that today file systems are more scalable than databases and that indexed file systems are a relatively simple extensions of file systems.
- As mentioned, databases are optimized to provide safe writes on rows of data, while indexed files can be optimized for efficient reads on columns of data.
- As the amount of data retrieved by a query grows in size, file systems become more efficient than databases. This is true as the size of a single returned object grows in size (blobs) [Sears-06], as the size of the aggregate amount of data grows in size, and as the size of the amount of data scanned by the query grows in size [Grossman-07].
- Appending to indexed file systems is very simple. One common way to support efficient appending is to limit access to the data in an indexed file system to the snapshot obtained by restricting to all data in the indexed file system available before the append begins. Sometimes this approach is called temporal snap shots.

## 4. INTEGRATED ARCHITECTURES

Recall that in our analytic model, data arrives as a stream of events, features are computed and assembled into profiles associated with entities of interest, and analytic models are computed using sets of profiles as inputs. The event and profile data may be managed using a database, an indexed file system, or hybrid of the two.

In this section, we discuss several hybrid approaches that are sometimes used.

For small to medium size data, one can use a system that transparently supports both data that is in memory and data that is managed by an indexed file system. The open source Augustus system uses this approach [Augustus-2007]. Augustus uses specialized indexed files to provide relatively efficient access to file based data. It also memory maps data into memory so that when data does fit into memory there is relatively little performance loss. Augustus is also designed to interoperate with databases so that Augustus' indexed files may manage events, while profiles may be managed by a database for example.

This illustrates the second hybrid architecture. In this architecture, we manage event data using an indexed file system and manage profiles using a database. In more detail:

1. We collect event data and store it in an indexed file system, appending by date of collection.

2. As required, we compute profiles from event data.
3. We store the computed profiles in relational database.
4. We extract the required profiles of a database.
5. We estimate a statistical model from the extracted profiles.

This approach has two important advantages:

- The first advantage is that since the event store uses an indexed file system, it is highly scalable and relatively simple to administer.
- The second advantage is that since the profiles are stored in a database, the profiles can be queried using the full power of SQL, which is very convenient for some reporting and analysis.

The main disadvantage of this approach is that it is more complex than simply using an indexed file system or a database alone.

As a third example, it is also common to store event data in a database and to extract the data to compute intermediate derived quantities that are stored in a column oriented indexed file system. Profiles can then be defined efficiently using column-wise operations on the derived data, as well as statistical models.

## 5. SUMMARY AND CONCLUSIONS

Data required for analytics must be managed. For most applications, databases provide the best mechanism today for managing data. This is the case even when data is extracted prior to building analytic models, since the overall analytic process usually benefits when data is stored in a database.

For some applications though, in which the data volumes are very large, an alternative mechanism is desired. Many of these large volume applications consist of transactional or event data that are associated with entities of interest for which profiles are maintained and used for analytic modeling. In this paper, we have summarized some of the key facts of how indexed file systems may be used for storing large amounts of event data and how they may be combined with a database containing profile data.

We end this paper, by summarizing some of the discussion above in four principles:

Recall that in this paper we assume that we have a stream of events containing information about entities of interest and that we want to estimate statistical models that will be used to make decisions about these entities. Table 1 contains several examples.

**Observation 1.** Think carefully whether your data fits into memory, requires a database for its management, or benefits from an indexed file system. Except for

relatively small amounts of data, it usually makes sense to manage data using a database, independently whether analytic computations will be done by accessing data directly in the database or on data that has been extracted from the database and loaded into the analytic application. As the size of the data under management grows and as the size of the amount of data retrieved in queries grows, indexed file systems provide some advantages over databases.

**Observation 2.** Evaluate your requirements to decide whether to access data using a row oriented or a column oriented data management infrastructure. This is closely related to the observation that a system can be designed for safe writes of relatively small amounts of data or efficiently reading of relatively large amounts of data, but not both.

**Observation 3.** The data for many analytic applications divides naturally into events and profiles. Sometimes it is helpful to use a different data management or analytic architecture to manage events and profiles. For example, depending upon the application, events and profiles may be stored in a database, in an indexed file system, or a hybrid of the two. As one example, raw events may be stored in a database, while profiles may be stored in an indexed file system that supports column-wise operations on features. As another example, events may be stored in an indexed file system due to their large size, while profiles may be stored in a database.

**Observation 4.** Shared nothing parallelism is a very effective mechanism for scaling up analytic systems. By shared nothing parallelism we mean parallelism in which data is partitioned into separate nodes in a cluster and neither memory nor disk is shared between nodes. All this requires is a middleware for scattering data and analytic queries and gathering the results. Ensemble based modeling is particularly attractive over clusters since separate models may be built on individual nodes and gathered to form an ensemble.

## 6. REFERENCES

[ATT-07] AT&T, Daytona Research, <http://www.research.att.com/~daytona/>.

[Augustus-2007] Augustus, [www.sf.net/projects/augustus](http://www.sf.net/projects/augustus).

[Bailey-1995] S. Bailey, R. Grossman, and D. Hanley, D. Benton and B. Hollebeek, Scalable Digital Libraries of Event Data and the NSCP Meta-Cluster, Proceedings of the Conference on Computing in High Energy Physics 1995.

[Borthakur-2007] Dhruba Borthakur, The Hadoop Distributed File System: Architecture and Design, [lucene.apache.org/hadoop/hdfs\\_design.html](http://lucene.apache.org/hadoop/hdfs_design.html).

[Brun-1997] Rene Brun and Fons Rademakers, ROOT - An Object Oriented Data Analysis Framework, Proceedings

AIHENP'96 Workshop, Lausanne, Sep. 1996, Nuclear Instruments and Methods in Physics A, Volume 389, 1997, pages 81-86. See also <http://root.cern.ch/>.

[Chaves-06] John Chaves, Chris Curry, Robert L. Grossman, David Locke and Steve Vejck, Augustus: The Design and Architecture of a PMML-Based Scoring Engine, Proceedings of the KDD-06 DM-SSP Workshop.

[Chang-06] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber, Bigtable: A Distributed Storage System for Structured Data, ODSI 2006.

[Ghemawat-03] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, The Google File System, SOSP 2003.

[Grossman-2002] Robert Grossman, and Marco Mazzucco, DataSpace - A Web Infrastructure for the Exploratory Analysis and Mining of Data, IEEE Computing in Science and Engineering, July/August, 2002, pages 44-51.

[Grossman-2005] Robert L. Grossman, Alert Management Systems: A Quick Introduction, in Managing Cyber Threats: Issues, Approaches and Challenges, edited by Vipin Kumar, Jaideep Srivastava and Aleksandar Lazarevic, Springer Science+Business Media, Inc., New York, 2005, pages 281-291.

[Grossman-2007] Robert L. Grossman, David J. Hanley, and Jennifer M. Schopf, RAY: A System Supporting Multiple Contending Scanning Queries on Large Data Sets, submitted for publication, National Center for Data Mining at the University of Illinois at Chicago Technical Report, 2007.

[Gray-2002] Jim Gray and Alexander S. Szalay: The world-wide telescope, Communications of the ACM, Volume 45, Number 11, pages 50-55, 2002.

[Ramakrishnan-2003] Raghu Ramakrishnan and Johannes Gehrke, Database Management Systems, third edition, McGraw-Hill, 2003.

[Rennhackkamp-1997] Martin Rennhackkamp, Sybase Warehousing, DBMS, August 1997. See also [www.dbmsmag.com/9708d17.html](http://www.dbmsmag.com/9708d17.html).

[Sears-06] Russell Sears, Catharine van Ingen, Jim Gray, To BLOB or Not To BLOB: Large Object Storage in a Database or a Filesystem?, Microsoft Research Technical Report MSR-TR-2006-45, June, 2006.

[Stonebraker-2005] Mike Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Sam Madden, Elizabeth O'Neil, Pat O'Neil, Alex Rasin, Nga Tran, Stan Zdonik, C-Store: A Column-oriented DBMS, Proceedings of VLDB, 2005.