NETEZZA
an IBM Company

Understanding Analytic Workloads

Meeting the complex processing demands of advanced analytics

Table of Contents

Introduction ..... go to P.3

Characteristics of an Analytic Workload ..... go to P.4

- Extreme Data Volume ..... go to P.5
- Data Model Complexity ..... go to P.5
- Variable and Unpredictable Data Traversal Paths, Patterns, and Frequencies ..... go to P.6
- Set-Oriented Processing and Bulk Operations ..... go to P.6
- Multi-Step, Multi-Touch Analysis Algorithms ..... go to P.7
- Complex Computation ..... go to P.7
- Temporary or Intermediate Staging of Data ..... go to P.7
- Change Isolation/Data Stability Implications ..... go to P.7

Workload Types and Sample SQL ..... go to P.8

- Online Transaction Processing (OLTP) ..... go to P.8
- Light-to-Moderate Decision Support ..... go to P.9
- Heavier Decision Support/Business Intelligence (BI) ..... go to P.12
- Complex, In-Database Analytics ..... go to P.14

Characteristics of an Analytic Processing Environment ..... go to P.16

- Analytics vs. OLTP ..... go to P.16
- Transactional/OLTP Environments ..... go to P.16
- Analytic Processing Environment (APE) ..... go to P.17

Conclusion ..... go to P.19

Written by: Allan Edwards, Director, Competitive Benchmarking & Analysis,
and George Davies, IBM Netezza Strategic Marketing

Introduction

> *ANALYTIC is the antonym of TRANSACTIONAL.*
>
> *Curt Monash**

The practice of analytics involves applying science and computing technology to vast amounts of raw data to yield valuable insights, and the "analytics" label covers a wide array of applications, tools, and techniques.

While there are many analytic variants and subspecialties—predictive analytics, in-database analytics, advanced analytics, web analytics, and so on—this text focuses on the characteristic demands that nearly all analytic processing problems place on modern information systems. We refer to these demands as an analytic workload. Every data processing problem has its own unique workload, but analytic workloads tend to share a set of attributes, with strong design and deployment implications for the processing systems assigned to handle these workloads.

Analytic vs. Transactional

Transactional processing is characterized by a large number of short, discrete, atomic transactions. The emphasis of online transaction processing (OLTP) systems is (a) high throughput (transactions per second), and (b) maintaining data integrity in multi-user environments.

Analytics processing is characterized by fewer users (business analysts rather than customers and POS operators) submitting fewer requests, but queries can be very complex and resource-intensive. Response time is frequently measured in tens to hundreds of seconds.

Transactional and analytics processing tasks constitute very different workloads, and transactional and analytic information systems are designed with these differences in mind.

This text focuses on the server side of the analytics processing paradigm. After defining the key characteristics of an analytic workload, we'll present several public examples of different workload types before moving on to outline the architectural characteristics of an analytic infrastructure, or processing environment.

* Monash now prefers the term "Short Request" over "Transactional."

Characteristics of an Analytic Workload

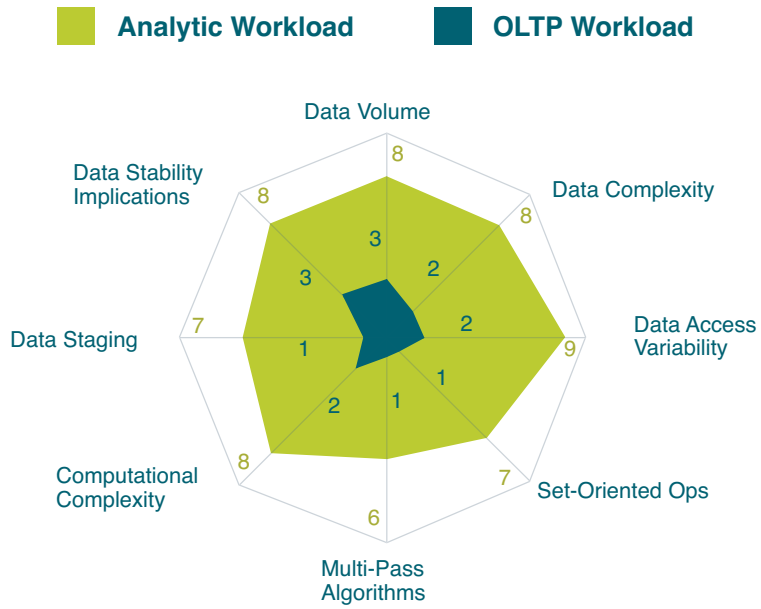
Before selecting, constructing, or deploying an analytic infrastructure, it makes sense to try to understand the basic characteristics and requirements of an analytic workload. Later, in addition to helping us outline an effective analytic infrastructure, these workload criteria can be used to evaluate a specific project or problem, yielding a rough measure of analytic complexity.

An analytic workload will exhibit one or more of the following characteristics, each of which elevates a given workload's degree of difficulty:

- Extreme data volume
- Data model complexity
- Variable and unpredictable traversal paths, patterns, and frequencies
- Set-oriented processing and bulk operations
- Multi-step, multi-touch analysis algorithms
- Complex computation
- Temporary or intermediate staging of data
- Change isolation/data stability implications

Rating each of these characteristics on a 1-10 scale yields the following general comparison for some idealized sample workloads:

Figure 1: **Analytic and Transactional Workloads**



Extreme Data Volume

While there is no specific threshold that makes a data set "large," it's fair to say that data volumes tend to be large in analytics processing. Applications like fraud detection, web analytics, and decision support are all routinely associated with the largest data stores, often measured in petabytes of data. Two direct metrics combine to drive up data volume:

- Row Cardinality (Number of Rows) – A single table's rows may number in the billions, tens of billions, or even hundreds of billions. Analytic workload requirements increase directly with row numbers. Simple physics dictates that when analyzing billions of rows, any inefficiency or overhead cost, no matter how small, becomes expensive.
- Row Width (Row Size) – It is not uncommon for tables to contain tens or hundreds of columns. Workload complexity increases as column counts increase, because larger rows consume more space, for both storage and processing. Larger row sizes also tend to introduce physical data sparsity as columns are skipped during various operations.

Data volume is further multiplied by any database management system (DBMS) that implements indexes, which store the indexed data redundantly, along with other metadata designed to streamline serial and selective data retrieval.

Data Model Complexity

> *Talking only about big data can lead to self-delusion.*
>
> *Merv Adrian, Gartner*

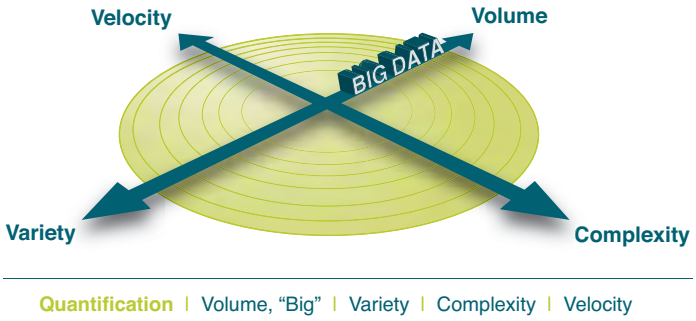
Large data volumes amplify the need for efficient, streamlined processing. Compounding large volume with complex data structures can result in processing demands that border on the unachievable. In addition to sheer size, "big data" or "extreme data" generally involves several dimensions, including:

- Data Object Complexity (Many Tables and Relations, Views, Data Structures, Etc.) – Data representation is typically spread across multiple data objects that must be combined, or "joined," at run time by the processing platform. As the quantity of relationships increases, so do the magnitude and complexity of the resultant processing. If virtual objects such as views are introduced, additional processing ensues to either materialize the view or merge the base objects (underneath the view) into the overall statement plan.
- Data Variety – Native system data types, data codification schemes, and other implementation details tend to vary across applications, systems, and enterprises. Analytic repositories often ingest data from many varied sources, encountering many different styles and types of data. These representations frequently must be transformed and/or converted so that consistent interpretation and logical cohesion can be achieved. While some systems convert data upstream with analytic processing, it is also common for conversion to be deferred until the point at which data are analyzed. This creates a spike of additional load on the underlying processing system.

- Data Model Style – Nearly any modeling style can be embraced in support of analytic processing—normalized, dimensional, etc. The objective is to employ a modeling style that constitutes a reasonable compromise and is sufficiently flexible to serve a variety of use cases, such as loading, retrieval, and archiving. No single style can be considered consummate or universal.

Analytic applications often characterized by data complexity include medical diagnostics, predictive modeling, portfolio analysis, and many others. According to Gartner, mixed data types—tables, media, clickstreams, sensor/metering data, text, and so on—comprise 70%-85% of all data.

Figure 2: **The Four Axes of Big Data**



Source: Gartner (March 2011)

Variable and Unpredictable Data Traversal Paths, Patterns, and Frequencies

Effective analysis requires fast and reliable performance regardless of data traversal routes and direction. Freedom to "roam the data" with consistent performance is essential in marketing analytics, forensics, fraud detection, etc. Bottom-up, top-down, and random-path scanning are all fair game for analytic practitioners, and this variable data path requirement is a distinctive aspect of analytic workloads—one with profound architectural implications. OLTP architectures generally presume predefined data access paths, which can be tuned and optimized for. Caching, clustering, colocation, partitioning, and indexing are all access optimizations that reduce traverse times for expected data access paths. Unfortunately, these structures tend to penalize traversal via alternative paths—an unacceptable result on an analytic platform.

Set-Oriented Processing and Bulk Operations

In another distinctive feature of analytic workloads, inter-row pattern aggregates—working sets, large or small—are frequent targets of research. This is in sharp contrast to the single-row, row-at-time, scalar analysis associated with typical transactional workloads. In a single analytics operation, working sets of hundreds, thousands, or millions of rows are common, and the data footprint for a single query can be enormous. As working set row counts increase, workload complexity escalates, sometimes exponentially. Sample applications include weather forecasting, forensic analysis, and economic modeling, which challenge analytic infrastructure designers in a way that transactional workloads do not.

Multi-Step, Multi-Touch Analysis Algorithms

Like variable traversal paths and set-oriented processing, multi-step/multi-pass data scanning and analysis are commonplace during analytic research. Sophisticated pattern analysis, for example, often requires multiple touches or passes of the same data during a single investigation. Furthermore, analytic functions—programming primitives—often decompose into multiple internal steps. Cross-correlation, lead-lag analysis, moving averages and aggregates, and many other applications involve this approach. Analytic functions and capabilities embedded close to the database, or within the database itself, can accomplish such work in a single pass or in fewer passes than pure SQL, potentially reducing by orders of magnitude data movement and server resource demand.

Complex Computation

Analytic processing frequently involves statistical analysis and/or additional sophisticated computational methods. A wide variety of mathematic and statistical operations is employed to help distill patterns, summaries, and other "interesting" results from raw data populations. Computational complexity increases demands on the server layer and the amount of work to be performed during a given request. As you might guess, computational complexity also travels hand in hand with other analytic workload characteristics, such as set-oriented processing, multi-pass analysis, and intermediate data-staging requirements.

Temporary or Intermediate Staging of Data

Analytic operations commonly stage intermediate data sets and interim results, particularly for multi-step/multi-pass algorithms and other sophisticated modeling and analytic methods—what-if scenario modeling, for example. These techniques presume the ability to write, retrieve, and integrate intermediate data at high volumes and high speeds, raising the processing requirements of the related queries substantially.

A Note Regarding the Data Movement Problem – Intermediate data staging is one of several analytic workload characteristics (including large data, multi-pass analysis, and set-oriented processing) with the potential to involve massive levels of internal data movement. Bulk data movement is a primary bottleneck and kills analytics processing performance on systems not optimized for these workloads. Architectural innovations that minimize data movement are a key feature of any analytics-oriented system.

Change Isolation/Data Stability Implications

Because analytics research typically involves an iterative sequence of requests and traversals to discover, drill, hypothesize, verify, and so on, it's important that any change to the underlying data be able to be isolated so as not to compromise or undermine active investigations. Predictive modeling and what-if analysis, for example, may involve longer-running requests that rely on stable underlying data sets.

Now, with the distinctive attributes of an analytic workload in mind, let's take a closer look at several workload types before moving on to outline the characteristics of an analytics processing environment that might be expected to satisfy an analytic workload's unique requirements.

Workload Types and Sample SQL

Workloads do not fall reliably into neat categories; they fall along a continuum with some recognizable waypoints. We'll look briefly at four points along the spectrum:

- Online Transaction Processing (OLTP)
- Light-to-Moderate Decision Support
- Heavier Decision Support and Business Intelligence (BI)
- Complex, In-Database Analytics

To illustrate various SQL usage patterns, let's examine some SQL excerpts from several benchmark suites developed by the Transaction Processing Performance Council (TPC). These excerpts are comparatively realistic while remaining simple enough to be useful for illustration. (You can skip the actual code examples without missing the key points about workload types and how they differ.)

Online Transaction Processing (OLTP)

In OLTP, the primary operations are (a) creating data records; (b) updating data records; and (c) retrieving data records, usually as singletons or small sets.

Optimal SQL for OLTP tends to be:

- Precise – Highly targeted to as few tables and rows as necessary. Each SQL data manipulation language (DML) statement typically modifies data in only one table.
- Prescribed – Coded, compiled, and "locked in." OLTP queries are almost never ad hoc or adjusted at run time based on variables or other input.
- Minimalist – Touching only the necessary database objects and columns.
- Transactional – Multiple steps packaged in a single transaction, or "unit of work."

The following SQL statements are excerpted from the *TPC-C benchmark*. Keep in mind that additional, "invisible" statements (triggers, referential integrity, etc.) are often implied, and these execute in the background, or "behind the scenes":

Figure 3: **Excerpts from TPC-C**

TPC Benchmark™ C (TPC-C) is an OLTP benchmark.

Delivery Transaction:

```
EXEC SQL SELECT o_c_id INTO :c_id FROM orders
                WHERE o_id = :no_o_id AND o_d_id = :d_id AND o_w_id = :w_id;

        EXEC SQL UPDATE orders SET o_carrier_id = :o_carrier_id
                WHERE o_id = :no_o_id AND o_d_id = :d_id AND o_w_id = :w_id;
```

```
        EXEC SQL UPDATE order_line SET ol_delivery_d = :datetime
                WHERE ol_o_id = :no_o_id AND ol_d_id = :d_id AND ol_w_id =
:w_id;

        EXEC SQL SELECT SUM(ol_amount) INTO :ol_total FROM order_line
                WHERE ol_o_id = :no_o_id AND ol_d_id = :d_id AND ol_w_id =
:w_id;

        EXEC SQL UPDATE customer SET c_balance = c_balance + :ol_total
                WHERE c_id = :c_id AND c_d_id = :d_id AND c_w_id = :w_id;

        EXEC SQL COMMIT WORK;
```

This SQL is compact, precise, and streamlined. There is no activity of an analytic nature involved here.

It's important to recognize that while basic operational facts are captured in the mainline OLTP SQL, more complex calculations, if required, are often deferred to post-processing and/or analytical phases in order to minimize the processing effort and data storage footprint of the primary OLTP application.

Another Note Regarding Data Movement – Ultimately, most workloads consist of multiple steps, or building blocks, and the fundamental building block is often an SQL statement. This is true for nearly all workload types. In general, it is beneficial to accomplish as much as possible in each step rather than subdivide processing tasks into a large number of discrete sub-steps. The goal is to minimize data movement and its attendant performance penalty. More steps equal more back-and-forth, more messaging, more data shipping and staging, more data movement, and degraded performance.

Light-to-Moderate Decision Support

Next, we'll look at SQL that was designed roughly a decade ago to represent "Decision Support and Reporting." This code differs substantially from the OLTP example, because the primary usage pattern here is retrieval rather than data creation.

The following query excerpts are extracted from the *TPC-H benchmark*:

Figure 4: **Excerpts from TPC-H**

The TPC Benchmark™ H (TPC-H) is a decision support benchmark. It consists of a suite of business-oriented ad hoc queries and concurrent data modifications. The queries and the data populating the database have been chosen to have broad, industrywide relevance. This benchmark illustrates decision support systems that examine large volumes of data, execute queries with a high degree of complexity, and give answers to critical business questions.

```
TPC-H Q12

        select
                l_shipmode,
```

```
                sum(decode(o_orderpriority, '1-URGENT', 1,
'2-HIGH', 1, 0)) as
                        high_line_count,
                sum(decode(o_orderpriority, '1-URGENT', 0,
'2-HIGH', 0, 1)) as
                        low_line_count
        from
                orders,
                lineitem
        where
                o_orderkey = l_orderkey
                and l_shipmode in ('[SHIPMODE1]',
'[SHIPMODE2]')
                and l_commitdate < l_receiptdate
                and l_shipdate < l_commitdate
                and l_receiptdate >= date '[DATE]'
                and l_receiptdate < date '[DATE]' + interval
'1' year
        group by
                l_shipmode
        order by
                l_shipmode;

TPC-H Q18

        select
                c_name,
                c_custkey,
                o_orderkey,
                o_orderdate,
                o_totalprice,
                sum(l_quantity)
        from
                customer,
                orders,
                lineitem
        where
                o_orderkey in (
                        select l_orderkey
                        from lineitem
                        group by l_orderkey
                        having sum(l_quantity) > [QUANTITY]
                        )
                and c_custkey = o_custkey
                and o_orderkey = l_orderkey
        group by
                c_name,
                c_custkey,
                o_orderkey,
                o_orderdate,
                o_totalprice
        order by
```

```
                o_totalprice desc,
                o_orderdate;

TPC-H Q21

        select
                s_name,
                count(*) as numwait
        from
                supplier,
                lineitem l1,
                orders,
                nation
        where
                s_suppkey = l1.l_suppkey
                and o_orderkey = l1.l_orderkey
                and o_orderstatus = 'F'
                and l1.l_receiptdate > l1.l_commitdate
                and exists (
                        select  *
                        from  lineitem l2
                        where  l2.l_orderkey = l1.l_orderkey
                        and l2.l_suppkey <> l1.l_suppkey
                        )
                and not exists (
                        select *
                        from  lineitem l3
                        where
                                l3.l_orderkey = l1.l_
orderkey
                                and l3.l_suppkey <> l1.l_
suppkey
                                and l3.l_receiptdate >
l3.l_commitdate
                                )
                and s_nationkey = n_nationkey
                and n_name = '[NATION]'
        group by
                s_name
        order by
                numwait desc,
                s_name;
```

These SQL segments differ from the OLTP example in several ways:

- The activity is retrieval only.
- Multiple database objects are involved in a single SQL statement.
- Some minor calculations are being performed.

- The selectivity of the SQL is uncertain—depending on the data values supplied and the data population, it could be selective or not.
- There is some aggregation and sorting happening via GROUP BY and ORDER BY clauses.

All in all, this SQL is still extremely simple and straightforward, presenting summarized facts rather than deep analytical insights.

Heavier Decision Support/Business Intelligence (BI)

Now we'll look at a slightly more complex retrieval example from *TPC-DS*, which is a little more recent and also somewhat more sophisticated. This benchmark is positioned to model decision support, which typically involves some amount of analytic activity.

Figure 5: **Excerpts from TPC-DS**

The TPC-DS benchmark models the decision support system of a retail product supplier, including queries and data maintenance. Although the underlying business model of TPC-DS is a retail product supplier, the database schema, data population, queries, data maintenance model, and implementation rules have been designed to be broadly representative of modern decision support systems.

```
TPC-DS Q36

         select
                  sum(ss_net_profit)/sum(ss_ext_sales_price) as
gross_margin
                  ,i_category
                  ,i_class
                  ,grouping(i_category)+grouping(i_class) as
lochierarchy
                  ,rank() over (
                           partition by grouping(i_
category)+grouping(i_class),
                           case when grouping(i_class) = 0 then
i_category end
                           order by sum(ss_net_profit)/sum(ss_
ext_sales_price) asc) as rank_within_parent
         from
                  store_sales
                  ,date_dim    d1
                  ,item
                  ,store
         where
                  d1.d_year = [YEAR]
                  and d1.d_date_sk = ss_sold_date_sk
                  and i_item_sk  = ss_item_sk
                  and s_store_sk  = ss_store_sk
                  and s_state in
('[STATE_A]','[STATE_B]','[STATE_C]','[STATE_D]','[STATE_E]','[S
```

```
TATE_F]',’[STATE_G]’,’[STATE_H]’)
         group by rollup(i_category,i_class)
         order by
                  lochierarchy desc
                  ,case when lochierarchy = 0 then i_category end
                  ,rank_within_parent

TPC-DS Q51

         WITH
         --
         web_v1 as (
         select
                  ws_item_sk item_sk, d_date,
                  sum(sum(ws_sales_price))
                           over (partition by ws_item_sk order by d_
date rows between unbounded
 preceding and current row) cume_sales
         from web_sales
                  ,date_dim
         where ws_sold_date_sk=d_date_sk and d_year=[YEAR] and ws_
item_sk is not NULL
         group by ws_item_sk, d_date),
         --
         store_v1 as (
         select
                  ss_item_sk item_sk, d_date,
                  sum(sum(ss_sales_price))
                           over (partition by ss_item_sk order by d_
date rows between unbounded preceding
 and current row) cume_sales
         from store_sales
                  ,date_dim
         where ss_sold_date_sk=d_date_sk
                  and d_year=[YEAR]
                  and ss_item_sk is not NULL
         group by ss_item_sk, d_date)
         --
         select *
         from (select item_sk
                  ,d_date
                  ,web_sales
                  ,store_sales
                  ,max(web_sales)
                           over (partition by item_sk order by d_date
rows between unbounded preceding
 and current row)
         web_cumulative
                  ,max(store_sales)
                           over (partition by item_sk order by d_date
rows between unbounded preceding
 and current row)
         store_cumulative
```

```
                  from (select case when web.item_sk is not null
then web.item_sk else store.item_sk
 end item_sk
                              ,case when web.d_date is not null then
web.d_date else store.d_date end d_date
                              ,web.cume_sales web_sales
                              ,store.cume_sales store_sales
                         from web_v1 web full outer join store_v1 store
on (web.item_sk = store.item_sk and
web.d_date = store.d_date)
                         )x )y
         where web_cumulative > store_cumulative
         order by item_sk
                  ,d_date;
```

These SQL segments include operations that are analytic in nature (particularly Q51), as evidenced by the SQL Analytic functions **Over (Partition by …)**, and these segments are considerably more complex than our previous examples. In particular:

- The activity is retrieval only.
- Multiple database objects are involved in a single SQL statement.
- Some minor calculations are being performed, but no heavy, complex computation.
- The selectivity of the SQL is uncertain—depending on the data values supplied and the data population, it could be selective or not.
- There is some aggregation and sorting happening via **GROUP BY** and **ORDER BY** clauses.
- Multiple sets of rows are being processed in a single execution using the SQL Analytic functions.
- There is sub-query refactoring. The SQL **WITH** clause represents an optimization commonly found in complex retrieval statements.
- The statements themselves are larger and more involved, with multiple sub-queries, phrases, features, etc.

Complex, In-Database Analytics

Finally, to get an idea of still greater analytic complexity, we'll touch on the notion of in-database analytics—sophisticated analytic functionality that resides within the data server platform and that can perform advanced analytical operations very efficiently via simple, straightforward application statements.

In market analysis, revenue forecasting, predictive modeling, and other business processes and applications, important analytic exercises include:

- Segmentation
- Selection
- Response Attribution

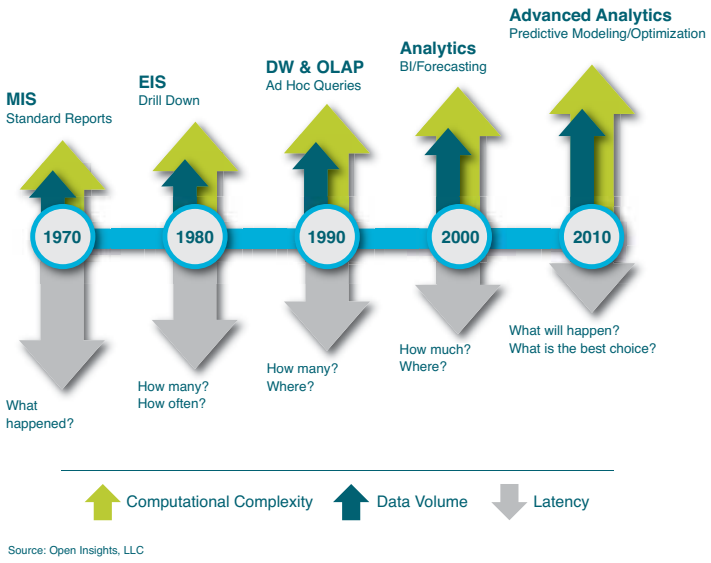
- Loyalty/Churn Analysis
- Geospatial Analysis
- And Many Others

Each of these analyses may involve very complex, sophisticated analytic operations against vast data repositories. A given enterprise's implementation typically involves a variety of integrated applications, which are unique and tailored to the local procedures, data, and computing technology stack. Common analytic primitives include:

- Linear Regression
- Quantile Calculation
- Random Sampling
- Principal Component Analysis
- Bayesian Network Analysis
- Canonical Correlation
- And Many Others

Unlike straight SQL systems, the interface to in-database analytics is not standardized, and the code that interacts with them is less transportable. Accordingly, no public benchmarks exist in this space and no code examples are offered here. However, note that on modern, state-of-the-art analytic platforms, many of the analytic primitives listed above can be invoked with a single application statement. A key benefit of systems architected for analytic workloads is that the advanced analytics code has been optimized for the platform on which it executes, and ideally, this code executes locally with a minimal amount of data movement or transport. This differs from traditional analytic applications, which run on a separate tier and must physically move data from the storage server(s) to the analytics server(s) for processing, after which any results must be moved back again for safekeeping.

Figure 6: **The Evolution of Analytics**



Characteristics of an Analytic Processing Environment

Having examined the distinctive attributes and requirements of an analytic workload, we can now outline the dimensions of a processing environment designed to handle such a workload. It will also help contrast an analytic processing environment with the requirements of transactional workloads and modern OLTP environments.

Analytics vs. OLTP

Transactional processing is characterized by a large number of short, discrete, atomic transactions. The emphasis of OLTP systems is (a) high throughput (transactions per second), and (b) maintaining data integrity in multi-user environments. OLTP systems are architected accordingly. OLTP databases are usually highly normalized, with many tables. Other characteristic architectural features include strongly codified and restricted code paths, infrastructure (indexes, partitioning, clustering, caches, etc.) to optimize predictable usage patterns and ensure integrity, and so on.

Analytics processing is characterized by fewer users submitting fewer requests, but queries can be very complex and resource-intensive. Response time is a key measure (not transactions per second), and analytics systems are architected accordingly. For example, databases tend to be denormalized, with fewer tables, in a star or snowflake schema; massive parallelism is employed to favor fast scanning and ingestion over fast selection; and data movement is minimized by in-database processing.

Conventional database systems exhibit dramatically variable performance when asked to process analytic workloads; this variability is exacerbated when the same system handles transactional workloads as well. On these systems, indexes, caches, partitioning, and clustering are all access optimizations designed to accelerate a particular data traversal pattern. Unfortunately, retrieval optimizations that benefit one access path are likely to penalize alternative paths. It is essential to be aware of this, and to tune the system repeatedly when attempting to deploy analytics on conventional or multi-use systems. This problem of unpredictable performance frequently plagues database professionals' early forays into the analytic realm.

Transactional/OLTP Environments

Transactional processing systems are typically characterized by small, simple, precise operations, and these systems exhibit the following characteristics:

- Optimization for short, atomic, repetitive, select-oriented operations and transactions—finding or updating very specific bits of data. Note that these access patterns lend themselves very much to tuning, hence the proliferation of indexes, caches, and tuning parameters in a typical OLTP environment.
- Prescribed, codified, discrete code paths.
- Heavy reliance on caching.
- Shared resources: shared data structures; shared memory; and the indexes, locks, latches, triggers, and other infrastructure required to manage concurrency and referential integrity.

- Data partitioning, data clustering, colocation, indexes, caches, and other tuning and configuration options that can be used to tune a system for the expected access patterns and data traversal paths.

In summary, the OLTP environment includes substantial infrastructure to support getting to one piece of data or one record very quickly, by any subset of a potentially large user population. Note also that the data partitions, indexes, caching, and tuning regimens result in a de facto "grain"—a set of optimized traversal paths—in the data. This is desirable in an OLTP system, where you know in advance (or come to know after a short time) exactly how you will need to traverse the data. However, these structural elements—indexes, shared memory, locks, caches, etc.—all impose performance and complexity penalties in an analytics environment, where unpredictable ("against the grain") access paths and patterns are the rule.

Analytic Processing Environment (APE)

The key virtues of an effective analytics environment include functional richness, processing speed, and ease of use. Because analytic research is typically a guided discovery process, where future questions depend on the outcome of the current question, speed and efficiency are paramount. With faster answers, more questions can be asked.

Modeling, forecasting, decision support, forensics, and ad hoc analysis—variants and cornerstones of analytic processing—require the agility to build and test models and queries very quickly, without having to tune or otherwise redefine the underlying system. Furthermore, because analytic requirements invariably evolve over time, some degree of flexibility will be expected of any analytics environment. Server-side APE characteristics include:

- DW-Level Storage Capacity – Analytics and large data sets are inseparable.
- Massively Parallel Architecture – Analytic platforms are optimized for fast scanning and ingestion rather than fast selection. Analytic workloads tend to operate on very large data subsets, often in variable and unpredictable sequences, as opposed to the traditional OLTP data access patterns involving small bits of data accessed more or less sequentially.
- Fast Data Movement Capacity – Because data movement can incur onerous performance penalties during large data set operations, a fast intra-system network fabric is a common feature of modern analytic environments. Data movement can be minimized, but it cannot be eliminated entirely.
- Minimization of Data Movement – Processing happens close to the data, as demanded by multi-touch/multi-pass processing algorithms. Dedicated disk-CPU combinations are a common feature, as is an intelligent code optimizer. Note that most analytic architectures incorporate one or more analytic client applications, each acting as a "workbench," issuing a series of analytic workload tasks to one or more data/analytics servers. Efficient, cooperative processing is vital. The ability to perform the heavy lifting—the actual analytic computations—in the data tier minimizes or eliminates the costly movement of data back and forth between servers and clients. Because data payloads can be so immense, data movement becomes enemy #1, the most likely barrier to success. When movement is eliminated through efficient execution in the data server layer, query response times are reduced by orders of magnitude.
- "Shared Little" or "Shared Nothing" Environment – The shared structures (memory, caches, etc.) common to transactional systems introduce unwanted contention and attendant performance penalties in an analytics environment, so these structures are generally minimized or omitted.

- Set-Oriented (versus Cursor-Based) Functions – Analytic systems need functions that operate on sets, not rows and cursors. (Set-related functions are common to most systems, whether analytic or transactional, so the shift from OLTP to analytics infrastructure is, in this case, largely a matter of adjusting programming style.)
- Analytic Functions Built into the Server – Beyond basic SQL operators, the server must include analytic functions. Modern analytic systems typically include some combination of proprietary and third-party analytic function libraries, some of which are in-database.
- Analytic Function Extensibility – An optimal analytics platform should include a user-visible layer or interface to permit reasonably straightforward additions of end-user functions and function libraries.
- Map-Reduce Options – Many systems include a built-in grid option (Hadoop or variant) as required to process map-reduce tasks.
- Streamlined and Efficient Code Plans, Data Movement, and Processing; Minimal Instructions per Data Element – This critical architectural "feature" is actually a full set of attributes implemented to a greater or lesser degree, depending on the system, with various innovations in code optimization, compression, hardware design, network fabrics, etc.
- Manageable by Business Users – To repeat, modeling, forecasting, decision support, forensics, and ad hoc analysis all require the agility to build and test models and queries very quickly, without having to tune or otherwise redefine the underlying system. Ease of use has become business critical, owing more to the lost time and opportunity costs of tuning and maintenance delays than to the real costs of personnel and expertise acquisition.
- Integration and Support for Sophisticated Analytic Client Applications – Although this text focuses on the server side, modern analytic environments incorporate one or more analytic client applications, each acting as a "workbench," issuing a series of analytic workload tasks to the server tier. Tight integration and support for a variety of client-side tools are mandatory.
- Genuine Scalability – More so than traditional systems, high-end analytic systems now presume continued exponential growth in both data stores and analytic processing demands. Hardware additions and system upgrades that yield nearly linear scaling have evolved to meet this requirement.
- Limited Reliance on Caching – The combination of too much raw data and unpredictable usage patterns greatly diminishes the worth of caching and cache-related overhead.

Conclusion

We define an analytic workload as a workload that exhibits some of the following characteristics:

- Large data volume.
- Complex data model (many and complex relationships, large rows, and diverse data sources and types).
- Unpredictable data traversal paths and patterns that cannot be easily anticipated or optimized for.
- Computational complexity.
- Set-oriented processing (rather than cursor-based, row-by-row access) and intra-query analytic operations involving many sets of rows.
- Intermediate and temporary data-staging requirements with high-speed access and short transport paths.
- Multi-step, multi-touch, multi-pass data access algorithms.
- Data change isolation requirements (so multiple long-running investigations can execute sequentially against a large, stable data store).
- Investigative query sets that cannot be predicted at the outset—guided-discovery analytic progressions that often must be time-boxed.

Several of these attributes are especially distinctive analytic workload markers. Most of the attributes, individually and collectively, have potent architectural implications as well. A server tier that can accommodate workloads with these attributes requires capabilities above and beyond conventional data serving. The demands of heavy computation—with multi-touch analytic algorithms using variable traversal paths over large, set-oriented data tranches—greatly exceed the capabilities of most general-purpose information systems. As a result, modern analytics processing platforms exhibit a variety of innovations, including massive parallelism, in-database processing, and advanced code optimization.

For analytics practitioners, the objective is to construct an analytics processing infrastructure to match the size and complexity of current and projected analytic workloads. For analytics system designers, vendors, integrators, and consultants, the goals are continued innovation and improvements to the analytic environments deployed for customers and end users. We hope the information herein can help with both of these objectives.

> *"Advanced analytics brings out the real value of data."*
>
> *Usama Fayyad, CEO, Open Insights*

NETEZZA
an IBM Company

Netezza, an IBM Company
26 Forest Street
Marlborough, MA 01752

+1 508 382 8200 TEL
+1 508 382 8300 FAX

www.netezza.com
http://thinking.netezza.com