ACTIVEVIAM

Analyzing data. Empowering the future.

WHITE PAPER

ActivePivot: Under the hood

TABLE OF CONTENTS

Introduction

Big Data analytics has emerged as one of the top priorities on the corporate agenda. Beyond the tremendous hype around 'Big Data', executives are increasingly grasping the competitive advantage that can be derived from unlimited, real-time insight into operational data. The transformational impact of Big Data is challenging the traditional IT architecture used by many organizations in the last decade. More specifically, it is disrupting the frontier between the transactional and analytical worlds, making the distinction between OLTP and OLAP systems no longer adequate for analyzing large volumes of dynamic data in real-time.

The need for speed

Today's analytical requirements are driven by the "need for speed". This need can be characterized by two interconnected elements: the speed of the data exploration process and the speed at which data is refreshed in the analytical environment. Addressing these distinct speed requirements is no trivial exercise. It ultimately translates into the implementation of **an in-memory analytical platform**, which can ensure that analytical queries can be executed in split seconds on data that gets refreshed on the fly.

Trying to solve the need for speed by optimizing OLAP-based architectures has proven to be a vain undertaking. OLAP was initially designed to allow business users to get the maximum useful information out of data. Yet, legacy OLAP failed to scale with usage diversification. It could not support the shift from "after the fact" reporting to a proactive use of analytics, with users interacting with data on an exploratory mode rather than through pre-canned reports. This more intuitive way of exploring data resulted in an explosion of aggregates due to the ever-increasing number of analysis dimensions. This outcome, also known as the **"curse of dimensionality",** considerably slowed down systems performance. It seemed that data navigation could only be achieved at the expense of performance and conversely, that performance could only be achieved at the expense of data navigation.

ROLAP and MOLAP are two examples of workarounds intended to solve the curse of dimensionality. MOLAP (Multi-dimensional OLAP) is based on storing data in a hypercube and pre-computing all possible aggregates. Whilst MOLAP meets the need for data exploration performance, it simply cannot meet the need for data freshness, because it relies on batch processing. On the other side of the spectrum, ROLAP (Relational OLAP) accesses the data in a relational database rather than in a hypercube. All aggregates are calculated "on-the-fly" for each new query. In this case, the speed of data navigation is at risk because of the computing time required to process complex queries.

The challenge, therefore, remains: **How to allow users to intuitively navigate through fresh data without compromising performance?**

The rise of in-memory computing

In-memory computing tackles processing speeds and dramatically accelerates query performance. It is about storing terabytes of data directly in the computer's RAM so that large volumes of data can be processed and analyzed rapidly.

This new computing style has benefited from several factors: First, the cost of memory hardware and chips has been dropping considerably. Since the RAM acts as a temporary workspace for the system's processing, the more the RAM available, the more tasks can be performed simultaneously and respond faster to analytics requirements. Secondly, the availability of commoditized multi-core processors made it possible to implement parallel algorithms that compress and index fast-moving data and speed up the time required to execute complex analytical queries.

In-memory computing clearly represents a powerful technological evolution to address the curse of dimensionality. In fact, modern in-memory visualization solutions rightfully claim to deliver on the promise of limitless data navigation at unparalleled speeds. Yet, the challenge of data freshness remains unsolved since users still cannot explore data that is updated on the fly. The reason is that in-memory

visualization does not support mixed workload requirements.

Mixed workload databases have the ability to perform short, tactical transactional queries, and at the same time to perform large, intensive analytical queries. A significant advantage of mixed workload databases is that they remove the latency between the transactional environment where data is created, and the analytical environment where data is explored. As a result, users are able to interactively explore data that is updated incrementally in real-time.

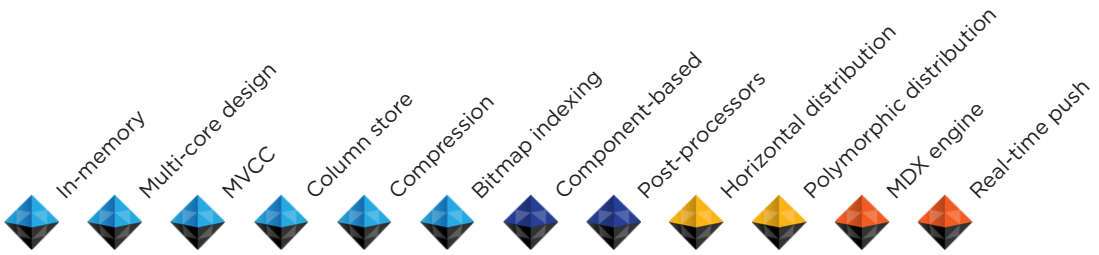
A change of paradigm

The only way to address the full "need for speed" spectrum is through a complete change of paradigm that deploys a **mixed workload**. This is the essence of the ActivePivot offering, provided through its real-time aggregation and analytics engine. **A mixed workload DBMS, ActivePivot is able to process multi-dimensional queries at unparalleled speeds on data that is updated on the fly, thus solving both aspects of the need for speed.**

ActivePivot combines the power of in-memory computing, parallel computing and many innovative algorithms to power a new generation of smart, analytics-enabled applications that support business users with their operational decision-making needs.

This white paper first examines the technical characteristics that make ActivePivot unique. It further explains how their combination delivers the technical benefits that are expected of true real-time analytics solutions.

ActivePivot's Technical Characteristics



Benefits Delivered

In-memory processing

The ability of in-memory computing to process data without requiring time-consuming and resource-intensive interfacing with the data storage is one of the top pre-requisites for rapid query performance. ActivePivot was natively designed to take advantage of in-memory computing in order to **meet the extreme performance requirements of both transactional and analytical environments.**

All the data processed by ActivePivot is stored in the computer's main memory. This means that transactions and queries never need to access a hard disk drive or any slower device, which typically provide several orders of magnitude slower data access.

Another benefit of using memory as the main storage is its random access support, resulting to constant time to read and write data to any part of the memory. This is a tremendous advantage compared to traditional hard drives, which require sequential accesses to be efficient, since they are significantly slower for random accesses.

The overall impact of ActivePivot's in-memory design is that, contrary to traditional RDBMS and OLAP systems, access to the stored data is easy. This feature enables ActivePivot to **perform transactions and queries significantly faster than traditional disk-based databases.**

Moreover, because data is stored in memory, ActivePivot can directly perform computation on it. Classical three-tier systems based on a disk-resident database typically extract data from the database, and then perform in-memory computation on the middle tier before sending results to the client. Put differently, traditional architectures 'bring the data' to the processors. ActivePivot, on the other hand, is an active database: because the data is already in memory, computations are directly performed on the ActivePivot server. Or in a sense, **ActivePivot brings computations to the data.**

Multi-core design

Year-after-year speed improvement of processors has started slowing down, the number of processors available on a single computer has increased. Today, quad-core CPUs have become the mainstream. The result of this hardware paradigm shift has affected how software programs are designed. Writing single-threaded algorithms with the lowest possible runtime complexity is no longer the best way to optimize the use of computer resources. Instead, resource optimization is achieved by writing parallel, multi-threaded algorithms that take advantage of all the available processing cores.

These algorithms cannot simply be added to a product as an afterthought. They require lockless concurrent data structures in order to efficiently function in parallel and not stall on large synchronization blocks. Parallel algorithms also require different coding techniques than traditional single-threaded algorithms, and rely on threading frameworks like Java's ForkJoinPool. Multi-threading must therefore be taken into consideration in the initial design of the product.

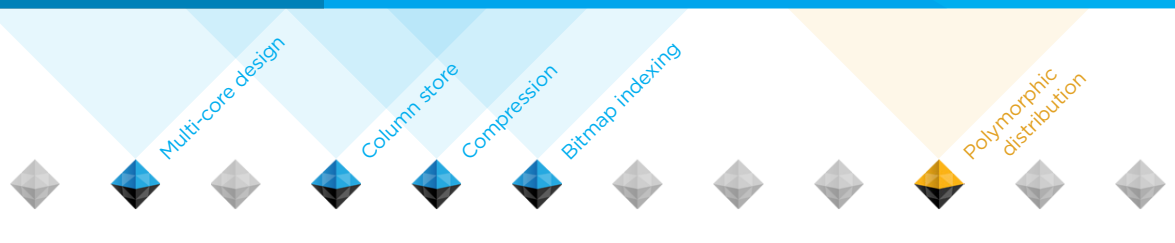
With this in mind, ActivePivot has been designed from the ground up to **leverage the computing power of a multi-CPU machine**, from dual-core Java VMs to the modern so-called "many-core" machines that embark several hundred cores. ActivePivot takes full advantage of NUMA (Non-Uniform Memory Access) architectures. On large servers with several processors, it means it can access 100% of the memory bandwidth of each individual processor, resulting in massive performance gains. The data structures used are thread-safe and all algorithms involved in transaction and query processing are heavily multithreaded – creating a design that lets ActivePivot 'squeeze' as much performance as possible from multi-core hardware.

TECHNICAL BENEFIT

High Dimensionality

One of the key challenges of OLAP systems is quickly aggregating data across different dimensions. Unlike most OLAP engines that only support a small number of analysis dimensions, ActivePivot supports multi-dimensional queries via:

- **a multi-dimensional 'Bitmap' index** that filters billions of records and supports hundreds of dimensions.
- **a column store indexer** that enables the processing of multi-dimensional queries at unparalleled speeds on data that is updated on the fly.
- **an efficient data compression mechanism** using a dictionary and Java primitive types, which when combined, enable the efficient data storage and access in memory.
- **multi-core architecture**, designed from the ground up, with parallel algorithms running on multiple threads that leverage the computing power of multi-CPU machines.
- **polymorphic distribution** that enables developing simple independent applications for heterogeneous data, which are federated on the fly.



Multiversion Concurrency Control (MVCC)

The most common way to ensure concurrency control in a database is to prevent users from reading and writing the same piece of data at the same time. However, this method based on locking can prove expensive in mixed workload environments that involve many updates, queries and calculations simultaneously.

Multiversion concurrency control (MVCC) works by storing multiple versions of the data in the database so that the data can be modified by a user while others continue to work with their own consistent versions. This ability for every user to write back in the database is the foundation of several key features of ActivePivot, such as what-if analysis, adjustments and validation processes.

Much unlike the back-ups of traditional database designs, ActivePivot only stores variations between versions. Accessing data in old versions has no overhead thanks to the in-memory architecture, resulting in an easy and fast access to previous versions of the database for "as-of" and root-cause analysis.

In short, MVCC is a key enabler of operational applications over large amounts of data that change in real time.

Column store

Conventional database management systems use a row-based store for their tables. Each fact is stored in a compact row, and any access/update to a field in the row requires reading/writing the entire row. While a row-based storage is suitable for transactional processing, it is not optimized for multi-dimensional analysis.

One of the key attributes of ActivePivot is its **column store**. In ActivePivot, all incoming facts are stored in the column store where each row represents a fact, and each column stores the value of a given "field" in the fact. When a new fact is inserted into ActivePivot (usually as a POJO – Plain Old Java Object, an array, or a map), a new row is created in the store; all the declared fields are extracted from that fact and stored in the relevant columns. The ability of column-based storage to access only the fields that are required for a particular query, instead of having to read/write the entire row **improves query performance.**

But there is more to it: The ActivePivot column store supports fact updates without requiring the use of an additional row-based store to serve data refreshing purposes. Unlike most columnar databases whose compression algorithms prevent them from easily updating facts, **ActivePivot's column store is fully updateable.**

Consequently, it doesn't require the combination of row-based tables with columnar tables to support mixed workload requirements. Offering a true mixed workload DBMS, ActivePivot is able to **process multi-dimensional queries at unparalleled speeds on data that is updated on the fly.**

Compression

ActivePivot is highly efficient in memory usage. Taking advantage of advanced compression algorithms, ActivePivot enables a **low memory footprint and incremental updates.**

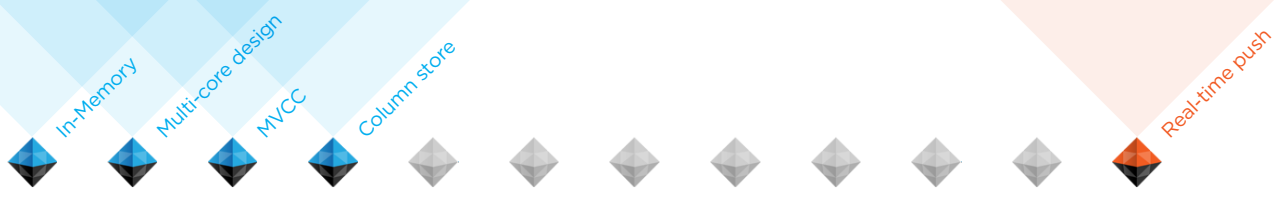
The first type of compression applied by ActivePivot is **"dictionary" compression**. In a database system, most fields only have a small number of possible values. Such fields are said to have low cardinality. For instance, a Transportation Management System (TMS) may store millions of facts, but there would be only a few hundred different possible values for each destination field. In such a case, ActivePivot will keep all the different destination values in a separate structure, called a dictionary, and assign a unique number from 1 to N to each destination (where N is the number of different destinations). This dictionary will be

TECHNICAL BENEFIT

Mixed Workload

While typically you are forced to select either an OLTP or OLAP system, depending on your workload, ActivePivot offers a mixed workload solution, supporting both system characteristics. The same database engine performs both data updates and complex analytical queries, so that analytical work is always done on fresh, up-to-date data that reflects the current state of the business. ActivePivot's mixed workload is enabled via:

- **an in-memory database engine** offering blazing-fast data access and the ability to insert/update large amounts of data while concurrently performing queries and analytics.
- **multiversion concurrency control** enables multiple queries to the database (both big and small) as well as write-backs to take place simultaneously.
- **an updateable column store** that stores the underlying facts so that they are easily updateable and quickly accessible for multi-dimensional queries.
- **multi-threaded algorithms** that maximize the usage of all computer cores and process both transactions and queries simultaneously to deliver the best possible throughput and response time.
- **a continuous real-time push engine** that ensures that the data used for analytical queries is always the most recent, propagated in real time to end users.



very small since it will only store a few hundred strings (the destinations). The destination column will then only store the integers that represent the destinations, instead of the large destination strings.

A second type of compression uses **Java primitive types**. When storing integers in a column, ActivePivot creates arrays of Java int primitives instead of highly inefficient Integer objects. Moreover, if the range of values of these integers is known, ActivePivot can only use the required number of bits instead of the full 4 bytes (32 bits) required by a Java integer. For instance, if we know that a column will only store values between 0 and 15, we only need to use 4 bits to store these numbers. This interacts nicely with dictionary compression: a dictionary usually has a low cardinality of N members, and the associated integer column can therefore be compressed by using only the number of bits required to encode an integer up to N while still supporting an unlimited number of members.

Bitmap indexing

An OLAP engine's goal is to execute multi-dimensional analytics queries. Queries typically need to first select facts based on the value of their fields on some dimension of the analysis, and then aggregate the facts along these dimensions to return business indicators.

One of the complex parts of the query is the selection of the facts based on multi-dimensional criteria. **An ActivePivot instance can contain billions of facts**, so being able to efficiently retrieve and filter them is crucial. As an in-memory engine, ActivePivot starts with a huge advantage over its disk-based competitors. Even in cases where a query needs to retrieve all the facts and aggregate them, reading the data directly from memory already provides an order of magnitude speed advantage.

However, when a query needs to filter facts based on conditions specifying the values of fields, an OLAP engine must use one or more indexes to efficiently retrieve filtered facts. Common database indexes (like B-tree), index a single attribute. For example, a "currency" index quickly locates records in "EUR" currency, the "year" index quickly locates records made in "2012", etc. Traditional databases therefore perform multidimensional queries by using several simple indexes. This process becomes extremely slow even when applying to only a few dimensions on any serious dataset.

Conventional databases have tried to resolve this issue using "composite" indexes – for example, an index dedicated to "Year + Quarter + Territory". But to answer all queries fast, one would also need to build multiple composite indexes, such as "Year + Quarter", "Year + Territory", etc.

Assuming the cube has more than a few dimensions, the resources required for such indexes is unmanageable (factorial of n number of dimensions).

ActivePivot resolves this issue with a proprietary multidimensional index, called **"Bitmap Index".** The Bitmap Index (a name that reflects its underlying binary arithmetic and compression schemes) selects records for any random combination of predicates on any number of dimensions. It is a lightweight data structure that filters at high speed over billions of records and easily supports more than 100 dimensions.

Component-based

Implemented in Java, ActivePivot is essentially an **object-oriented OLAP framework**, configured using Spring and XML files. The architecture of ActivePivot is based on the inversion of a control paradigm and is therefore inherently open to customizations and extensions. ActivePivot can be customized at all stages of the information flow:

- acquiring data from different sources
- feeding data objects into the database

TECHNICAL BENEFIT

What-if analysis

With ActivePivot you can perform instant simulations and 'What-If' analysis on huge data sets to evaluate alternative scenarios, make projections, and reach informed business decisions. It leverages ActivePivot's ability to perform real time transactions and to execute queries immediately on the updated data. Thanks to MVCC, when conducting 'What-If' analysis, users create their own splinter environments on-the-fly where they can perform personalized 'What-If' simulations and queries with no impact on other users.

- **post-processors,** which allow for complex and non-trivial aggregation mechanisms that can rely on both internal and external data.
- **an updateable column store** that stores the underlying facts so that they are easily updatable and quickly accessible for multi-dimensional queries.
- **an in-memory database engine** that enables performing real-time transactions (such as insert or update data) with the query performed immediately on the updated data.
- **multiversion concurrency control** enables users to create splinter environments on-the-fly, adjust any parameters, analyze and share the results and ultimately validate or discard the changes.

- enriching objects with additional calculated data, prior to aggregation
- defining the database in terms of dimensions and measures
- aggregating data values within the database
- calculating additional measures, following aggregation
- querying the database information
- interfacing with client applications

An important implication of this approach is that any kind of object can be stored, processed and aggregated in ActivePivot. Numerical values and vectors are naturally handled by ActivePivot, but it is also **extremely easy to inject logic to aggregate more complex objects** like matrices or custom business objects, and then compute statistical or predictive measures directly in the ActivePivot server memory.

A key feature of OLAP systems is the ability to quickly aggregate data across different dimensions. For instance, in a trading application, each trade has a Profit and Loss (PnL) value attached to it. An OLAP system will quickly be able to return the total PnL for all the trades in the application, or the PnL breakdown per trade currency, per sector, or any combination of analysis dimensions.

However, an important restriction of traditional OLAP engines is that the aggregation function applied to the data needs to have an inference property. The aggregated value of a measure for a set of facts needs to be a function of the value of this measure for each of these facts. For instance, the PnL of a portfolio of trades needs to be a function of the PnL of each trade (in this case, it is the sum).

Another restriction is that the aggregation function applied to the data performs the same computational work, regardless of the query. For instance, it will apply the same formula to aggregate each trade PnL, regardless of whether the PnL is aggregated per sector or per currency. One last restriction is that the values of the measures should only depend on the current state of the cube, and cannot rely on external data such as a stock price, a Forex rate, and so on.

Post-processors

ActivePivot eliminates all these restrictions by introducing a feature called **post-processors**. A post-processor is defined on the server-side using Java code, and is evaluated at query time for each requested aggregate. Post-processors are extremely flexible. They can use pre-aggregated data (e.g. classical OLAP measures), custom calculations, other postprocessor results and/or any external resources to compute its value (e.g. external FOREX rates).

A first advantage of post-processors is that they are exposed to ActivePivot's users as **plain vanilla measures**. As a result, **post-processors are more flexible than traditional databases' stored procedures**, like Oracle's PL/SQL: they always have a value, and this value can depend on the query context.

Also, because the post-processor is written in Java and can perform any computation to return its result, it is much **more powerful than MDX's calculated measures**, which are limited in functionality. A post-processor essentially offers infinite possibilities for aggregation.

Example: Consider a position-keeping trading system, in which the facts/positions are defined as follows: (stockId, trader, boughtPrice, quantity). A fact/position is a stock that a given trader has bought at a given price, with its quantity. In order to compute a position's current PnL, we need to apply the computation: PnL = quantity * (currentPrice - boughtPrice)

The value of a stock is a very volatile number that can change many times a second. It is therefore not reasonable to store it in ActivePivot, or any kind of OLAP system for that matter. However, a traditional OLAP system would have no choice but to store the "current" price in its database. And because it cannot update all facts multiple times a second, the OLAP system would use a delayed price, updated every 20 minutes or so. Such an approach is not acceptable for a real-time, critical trading system. On the other hand, it is very easy for ActivePivot's post-processor to accurately compute the PnL of a trade, using real-time data streams, and aggregate its value across any dimension.

TECHNICAL BENEFIT

Alerts and monitoring

The main requirement for monitoring business indicators and alerting users is to be informed of a change in underlying data. ActivePivot's alert and monitoring capabilities let you be notified about any change in a specified set of data; using more advanced measures, you can also be alerted on any business logic you define. For instance, you may be alerted if a KPI increases/decreases by more than the average of your competitors during a specific time frame. Contrary to conventional databases that use triggers or polling to monitor their data, ActivePivot's alerts and monitoring capabilities are enabled via:

- **an updateable column store** that stores the underlying facts so that they are easily updatable and quickly accessible for multi-dimensional queries.
- **an in-memory database engine** offering blazing-fast data access and the ability to insert/update large amounts of data while concurrently performing queries and analytics.
- **a continuous, real-time push engine** that alerts users when the data in which they are interested has changed.
- **a rule engine** (ActiveMonitor) that allows users to register rules that will continually monitor the cube, and generate an alert when the rule is satisfied or a measure stay pervasively over some threshold for a period of time.

Real-time push

Horizontal distribution

ActivePivot's distributed architecture supports multiple servers, spread across a local area network or a wide area network to **deliver increased performance and improve resilience**. Horizontal distribution is a natural distribution solution and is based on selecting a field value to partition the data across multiple servers. For example, all trades booked in Europe will go to the first server instance while trades booked in North America will go to the second server. This approach is well suited for managing historical data and geographical distributions.

Horizontal distribution is built on a **'shared nothing' (SN) architecture** with two operating principles in mind: First, seperate cubes that can be started independently. For example, if one server fails, other servers and cubes remain unaffected. Second, each cube loads data independently. Thus, if one cube can load data at rate of 100,000 facts per second then a grid of four cubes will load 400,000 facts per second.

Servers are registered into a group with no knowledge of each other. There is no slave/master relationship and all servers are at the same level. Servers can come and go and be replaced or added at will to provide more capacity. Servers communicate with each other using a reliable, industry standard multi-cast technology. This architecture ensures the **simplicity of deployment.** When an ActivePivot instance joins a group, it receives only a copy of the dimensions available throughout the group from its peers. The group propagates dimension updates between peers so that each server instance has the full dimension content. This way, small discovery requests that are expected to run very fast, can run locally on a server instead of involving group communication.

Primitive aggregate queries (handled by an aggregate provider) are distributed. The same query is broadcasted to peers and the partial results are merged by the server that received the original query. This ensures that there is no degradation of performance, as each server uses its full power to process the query without duplication of efforts to retrieve aggregates. Some post-processed measures can also be computed in a distributed fashion. They are broadcasted and executed locally on each of the peers. Results are then merged on the server that received the initial query based on user-defined logic.

Polymorphic distribution

Polymorphic distribution consists of splitting strictly disjoint measures and their associated data across multiple servers. It is well suited to address aggregation of data from different logical data chunks or combining heterogeneous data such as daily delivery schedules and inventories.

Because ActivePivot is object oriented and extendible, it can be used for complex projects that **mix heterogeneous analytics**, such as PnL, Sensitivities, and VaR. The challenge here is that dimensions are not the same: VaR has a scenario dimension, some sensitivities are tenor based, while PnL values are scalars. This puts a lot of stress on development teams to make these disjoint attributes coexist.

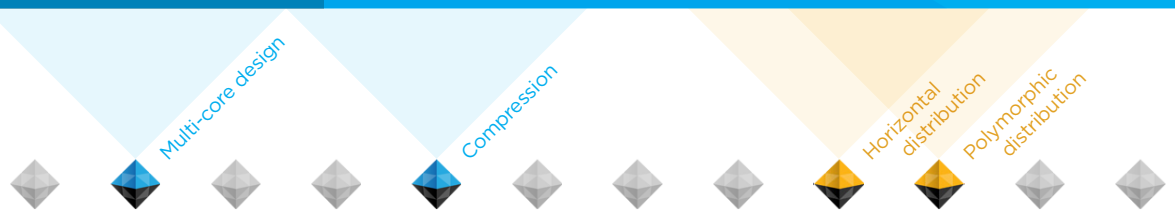
With polymorphic distribution, ActivePivot enables the development of simple cubes (e.g. one for PnL, one for sensitivities and one for VaR). Each application is independent, developed at its own pace, and is easy to maintain. Then ActivePivot **federates them on the fly**, joining on what they have in common, and resolving any mismatches.

TECHNICAL BENEFIT

Scalability

Designed to scale-up and scale-out, ActivePivot supports expansion of business data and the need to continuously increase the scope and depth of your data analysis. This is achieved via:

- **multi-threaded parallel algorithms**, which enable ActivePivot to maximize its use of computer cores. Adding more cores allows ActivePivot to scale-up on a single box.
- **efficient data compression algorithms**, which allow storing a large amount of data within each ActivePivot instance. This enables the application to grow while using a minimal amount of memory.
- **horizontal distribution**, which automatically adds more instances to a distributed ActivePivot deployment, resulting in increased computing power and available storage.
- **polymorphic distribution**, which simplifies the data architecture and enables the efficient management of multiple cubes for different business areas.



MDX engine

As an Online Analytical Processing (OLAP) engine, ActivePivot is designed to **answer multi-dimensional analytical queries**. These queries first select facts based on the value of their fields and along a set of dimensions. Values are then aggregated along the dimensions and the result or "business indicator" is returned.

For instance, a typical analytical query would be: "sum all the electronic sales from Q1 2010". This is a multi-dimensional query on the "product type" (electronics), "quarter" (Q1) and "year" (2010) dimensions. The returned business indicator would be the company's revenue for Q1 2010.

Such multi-dimensional analytical queries are best described with a **language called MDX** (Multidimensional Expressions). A specialized syntax for querying the multidimensional data stored in cubes, MDX provides much more intuitive data analysis than SQL. Using MDX a user can:

- Start with high-level aggregates, then navigate into the details
- Slice and dice
- Filter and group
- Crossjoin

MDX is the **standard query language for OLAP** systems. All major OLAP players have adopted MDX, including database vendors such as Oracle, SAP and vendors of visualization tools such as Tableau, Tibco and Microsoft.

The ActivePivot engine relies on a custom MDX parser and interpreter that has been designed from the ground up to provide best-in-class response times for some of the most complex analytical queries.

Real-time push

Providing "end-to-end" real-time push, ActivePivot enables a user to register a **"continuous query"** and to receive the results when the database is updated. When new data is streamed into ActivePivot (such as a new record in the cube itself or a contextual update like a stock price tick) - the engine precisely computes the impact on the registered continuous queries, and only recomputes what has potentially changed. Then the cells that have changed (and only those cells!) are pushed in real-time to clients or downstream applications.

Contrary to naïve implementations that fully recompute queries and thus fail with a few dozen concurrent subscriptions, ActivePivot optimizes the processing power to recompute queries and minimize the bandwidth to transport updates. Consequently, **a single ActivePivot node can scale up to hundreds of concurrent real-time continuous queries**, with ActivePivot itself processing tens of thousands of updates per second, per core.

ActivePivot's end-to-end, real-time push consists of:

- **Real-time data sources**: Being object-based, ActivePivot naturally accepts multiple heterogeneous real time data sources.
- **Continuous update to the cube**: ActivePivot updates a cube continually in real-time. While some users might work with ad-hoc queries, other users need to see live reports that are updated whenever the cube data changes.
- **Streaming API:** ActivePivot notifies registered clients through a proprietary streaming API as soon as the data is updated.

TECHNICAL BENEFIT

Data Visualization

ActivePivot provides a web-based interface for intuitive visualization of query results and data analysis in a timely fashion. Data visualization is enabled via:

- **multiple GUIs like Excel, Spotfire, Tableau and ActiveUI**, which provide a graphical interface to the ActivePivot engine in MDX, the standard query language for multi-dimensional analysis.
- **a continuous, real-time push engine**, which ensures users view the freshest data so the informed business decisions can be made.
- **a multi-dimensional index** that supports hundreds of dimensions.
- **user-defined aggregation functions and post-processors**, which allow users to define the exact data sets to be visualized for their business goals.

LONDON
6th floor, Shaftesbury House
151 Shaftesbury Avenue
London WC2H 8AL
Tel: +44 20 7836 8820

NEW YORK
1412 Broadway Suite 2300
New York, NY 10018 USA
Tel: +1 646 688 4442

PARIS
46 rue de l'Arbre Sec
75001 Paris, France
Tel: +33 1 40 13 91 00

SINGAPORE
29 C Stanley Street, #03-01
068738 Singapore
Tel: +65 62 24 46 63

HONG KONG
21/F On Hing Building
1 On Hing Terrace
Central, Hong Kong
Tel: +852 3971 9154

About ActiveViam

ActiveViam provide precision data analytics tools to help organisations make better decisions faster.

ActiveViam started in 2005 with the vision of leveraging in-memory technology to create an analytics platform where businesses could leverage the largest data sets without restrictions, keep them up-to-date in real time and use them to empower their decision makers.

Our goal at ActiveViam, is to let organisations not only make decisions faster, but better; to not only reach their data, but their potential; to not only see their data, but find their way into the future.

ActiveViam is a privately owned company with offices in Paris, London, New York, Hong Kong and Singapore.

For more information please visit:
www.activeviam.com