

A man in a white shirt and tie stands in a field, holding a glowing sphere. The background is a collage of digital and business imagery, including binary code, a globe, and a bar chart.

SAS® Software and Firebird

Firebird Conference 2005

Tom Cole, Platform R&D, SAS Institute Inc.
tom.cole@sas.com

SAS® and *Vulcan*

- § SAS and a new I/O architecture
- § Why Vulcan?
- § How Vulcan and SAS fit together
- § Challenges for the future

Shifting Technical Requirements

- § SAS® has long had an architecture that focussed on reading massive amounts of data and applying analytical processing to that data.
- § Building on our strengths in data manipulation and key analytics such as forecasting, we have added a “solutions” suite to our traditional “tools” suite.
- § These solutions bring new technical challenges, and new requirements for I/O support.

SAS® Technical History

- § 1970's - SAS grows from university research project to corporation, initial implementation on MVS in PL/1.
- § 1980's - Portability drives migration to C and major re-structure of architecture with “host” layering.
- § 1990's - Rapid evolution in RISC, storage, networking technology demands vertical performance focus
- § 2000's - SMP and 64 bits demand horizontal performance focus.

Analytical I/O

- § Data sources often read-only or ETL to warehouse
- § Terabyte and Petabyte data sets
- § Very wide tables (thousands of columns)
- § Some kinds of processing very order sensitive, justifying intermediate sorts
- § Very sequential with occasional rewinds to start of sorted groups.

Non-analytical I/O

- § Solutions focus brings new requirements
- § Integration requires new metadata store
- § User applications require new security, profiles, etc.
- § Interaction with 3rd party requirements for object persistence, such as J2EE Application Servers

New Requirements

- § Transactional Store - ACID
- § Multi-user: Hundreds -> Thousands of clients
- § SMP and cluster exploitation
- § New, non-analytic data types
- § Seamless interoperoperation with historical I/O models

SAS® Table Services

- § Total redesign of SAS I/O functionality
- § Retain historical analytical interfaces
- § Add new ODBC 3 interfaces, semantics
- § Fully threaded with scalable resource sharing
- § Implemented as an embedded library of services
- § Plug-in driver architecture

SAS® Table Server

- § Threaded, scalable server for n-tier architecture
- § Manages connections, thread-pooling, etc.
- § Uses SAS Table Services for actual I/O
- § Acts as persistent object server for mid-tier platform
- § Acts as I/O server for server and mid-tier
- § Can be accessed by user or 3rd party applications

SAS® Table Services Features

- § Support for ANSI SQL 1999 core-compliant syntax for historical data sources (Base SAS, SPDS)
- § Also allows pass-through to provider-specific drivers
- § Federated (cross-database) query management
- § Federated (cross-provider) security management

One interface for all SAS I/O requirements.

The need for a transactional store

- § SAS® has never had a transactional store of our own.
- § Many usage scenarios of SAS don't require transactions at all.
- § Provided a driver interface in an earlier architecture to access range of relational databases.
- § Requirements of metadata, objects, solutions, etc. mean we now need an ACID compliant store for many more usage scenarios.

Enterprise Class Customers

- § Evolved from individual knowledge worker to mission critical, multi-user, n-tier configurations
- § Today, SAS® is used heavily in Fortune 100 companies as an integral part of large-scale IT operations.
- § Typical deployments are 64-bit systems, 4-8 CPU's, with 16-32GB of RAM, and terabytes of disk space.
- § For our customers, time is most often the limiting resource - consuming system resources to complete a job in a fixed window of time is a good trade-off.

Business case for Open Source

- § “Build vs. Buy” - don’t build it, there are good ones out there already based on years of experience.
- § Potential to leverage large body of expertise and development infrastructure.
- § SAS® can focus on it’s product-specific goals and leverage our expertise in tandem with the community.

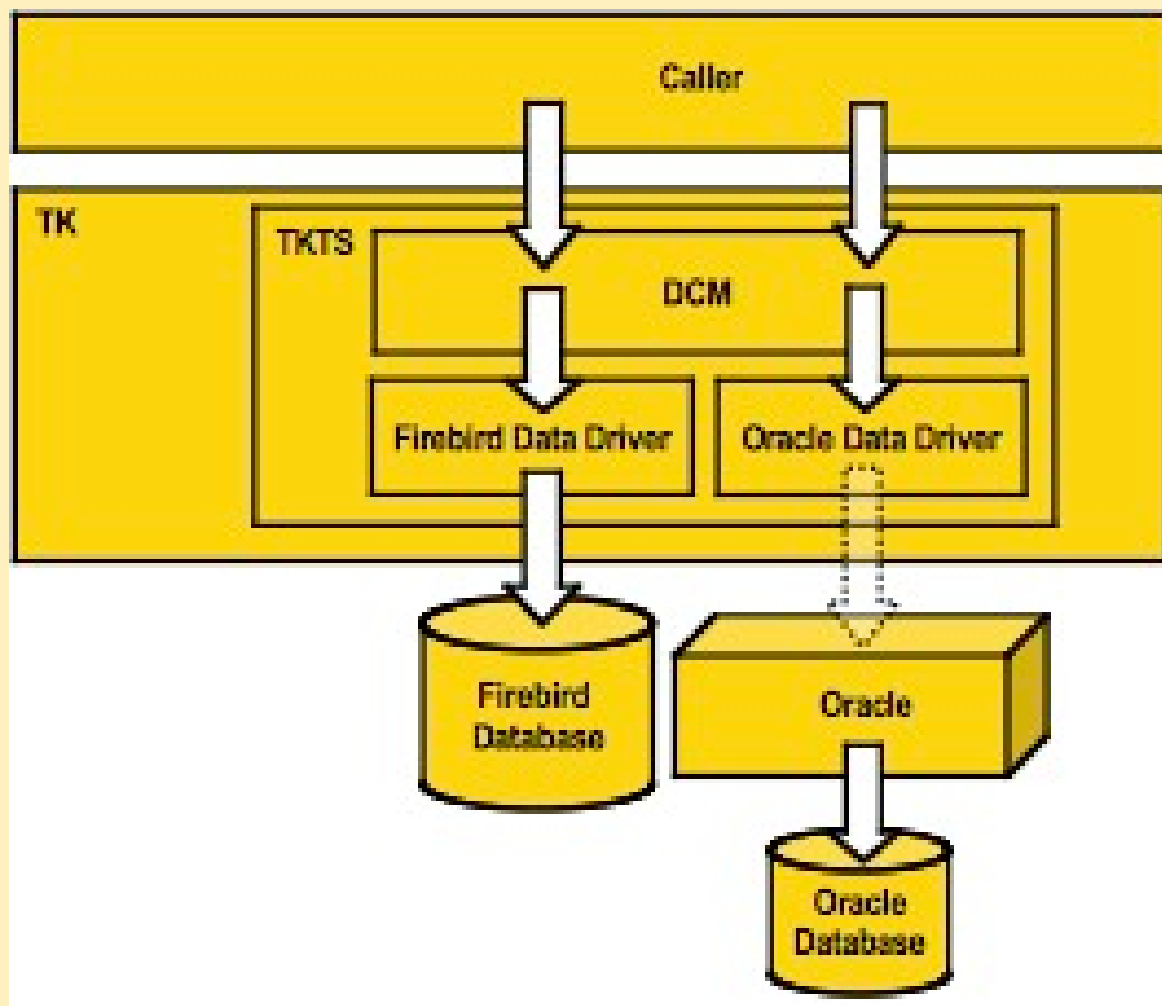
Challenges of Open Source

- § Aligning design goals
- § Aligning release requirements
- § Aligning product delivery dates
- § License requirements and restrictions
- § Infrastructure compatibility

Why Firebird/Vulcan?

- § Scalability objectives fit well - SMP support required, not just scalability through process replication.
- § Embedded support.
- § More likely to be successfully ported to wide range of host platforms we must support.
- § Active community open to our involvement.
- § Licensing flexibility and low cost.

Table Server Driver Architecture



SAS® Table Server Architecture

- § Table Server handles connection management using SAS-specific connection string syntax.
- § Table Server handles authentication requirements using a federated security data store.
- § Table Server identifies the required provider(s) for a request, and ensures proper drivers are loaded.
- § Table Server evaluates the SQL to determine if it must be recomposed or restructured for execution by multiple drivers.

SAS® Table Server Architecture

- § Table Server can process all the SQL directly in a driver-neutral ANSI standard fashion, and convert it to appropriate driver-specific dialects.
- § Table Server translates SAS-specific data types (special missing values, for example) and has access to the library of SAS formats and functions which can be used directly in SQL syntax.

Vulcan and SAS® Table Server

- § A Table Server driver written for Vulcan translates Table Server interfaces and semantics into Vulcan interface calls and semantics.
- § The driver arranges for any needed data type translations, character set management, and error handling.
- § The driver is dynamically loaded on demand by the Table Server based on client requests. The driver manages the loading (and optionally unloading) of the Vulcan shared libraries.

Vulcan and SAS® Table Server

- § Embedded Vulcan provides a very efficient database engine, without the performance costs of an additional hop to a provider.
- § Vulcan's locking model supports multiple instances of Table Server accessing the same database.
- § Vulcan's thread-safe implementation partners effectively with Table Server's thread pooling and resource management to support scalability.
- § Ability to support hot backups of running servers.

Vulcan and SAS® Table Server

- § In our first release of Table Server in 2006, we do not expect end users to directly interact with Vulcan.
- § That is, we want them to go through Table Server to interact with “our” transactional data store.
- § We provide an administrative interface that supports DBA functions via a Java application, which is common to all SAS products.
- § We will ship adapted versions of GBAK, etc. to support DBA and system management functions.

The Vulcan Environment

- § The Table Server driver explicitly sets process environment variables (VULCAN, VULCAN_CONF, etc.) to point to the SAS®-installed instance of Vulcan.
- § Table Server handles authentication and authorization to achieve federated security. Vulcan is invoked essentially as “SYSDBA”. Security is disabled.
- § User-installed instances of Vulcan shouldn’t collide with our use of Vulcan.

Building Vulcan across platforms

- § We deploy Vulcan on VMS, MVS, Windows, and Linux, as well as all name-brand Unix 64-bit systems.
- § We use an in-house source management, build and debug environment that supports multi-platform builds
- § Compilers on some platforms (MVS in particular) introduce additional constraints on building, such as header file locations.

Project Goals and Synchronization

- § SAS® development priority is portability and scalability for commercial release in 2006.
- § Community priority is in architectural review and revision.
- § Both are ultimately required, but compete with each other in the near-term.
- § Community focus has been on Firebird much more than on Vulcan.

How many Vulcans?

- § The unfortunate consequence is that right now, there are (at least) two versions of Vulcan.
- § The SourceForge version is changing rapidly with class-restructuring, etc. driving much needed architectural changes at the cost of stability.
- § The SAS® version (maintained privately) changes more slowly, and undergoes rigorous testing on multiple platforms with diverse loads.
- § Both goals are valid and needed, but conflict in the short term.

SAS®-Specific Vulcan

- § Build environment issues mean we do not mirror the open source build tree, but instead restructure it when we update.
- § For example, in our environment file names cannot be duplicated in different directories (only one copy of `ibase.h` allowed, etc).
- § Automated tools allow us to translate source files between our environment and the SourceForge one.

SAS®-Specific Vulcan

- § We regularly pull specific bug fixes into our code base from SourceForge.
- § Our intent is to provide batches of updates when we find and fix bugs - though we need to improve that process.
- § We are using `#ifdef SAS_FIREBIRD` to demarcate changes in open source files that we must post publicly to meet license requirements.
- § SAS-specific implementations in separate source files, called/included from conventional source.

SAS®-Specific Vulcan

- § SHARED_CACHE is not defined in our build; i.e. per-connection caching is done. This is different than the default build. Additional locking changes.
- § Per-statement cancel operation
- § Command-line tools define environment (VULCAN, VULCAN_CONF, etc.) based on directory of invocation.
- § SAS\$METADATA table added to all databases to track additional Table Server-specific data.

Reconciling Vulcan and Firebird


- § Since Vulcan was branched from Firebird, a lot has been changed in Firebird.
- § Many of these features (expressions in indexes, for example) are critical for our future use.
- § Several discussions have occurred about the challenges of finding a way to merge them together.
- § SAS sees this as essential for the long-term success of Firebird overall.

Conclusion

- § The Firebird community has made it possible for us tightly integrate a transactional DB far more quickly than we could have accomplished on our own.
- § The performance and scalability potential of the Vulcan design fit well with “enterprise class” software.
- § Short-term divergence of goals mean our participation in the community has been limited.
- § Our longer-term goal is to re-synch with the “one true” Vulcan, and be an active participant in advocating for embedded usage of Vulcan.

For More Information...

- § See the paper on “Embedded Classic” about how we use the Classic model of caching in the thread-safe embedded use of Vulcan, via the `SHARED_CACHE` compile-time symbol and its effect on locking.
- § See the paper “SAS/Firebird Porting And Testing Processes” for more information on our infrastructure.
- § See the related paper on “SAS/Firebird Performance Testing Strategy” on how we stress-test the scalability of the code.



Q&A?

The Power to Know®