

SAS[®] Software and Firebird

INTRODUCTION

This paper describes the architecture and requirements for implementing a relational, transaction-based storage mechanism in SAS[®] software – a commercial business intelligence product that uses the Vulcan code base in an embedded mode.

The requirements for large-scale software products, such as SAS, including characteristic customer configurations and workloads are reviewed. The discussion addresses why SAS selected Firebird as the basis for its transactional storage model, and why Vulcan, in particular, meets the company's needs. There is also a brief discussion of the challenges in integrating open source projects with commercial projects, and some of the extensions to Vulcan that SAS has added to support the company's needs.

Finally, this paper identifies future challenges in integrating product and open source goals and code bases.

COMMERCIAL TECHNICAL REQUIREMENTS

SAS software has long had an architecture that reads massive amounts of data and applies analytical processing to that data. Building on its strengths in data manipulation and key analytics, such as forecasting, SAS added a Solutions suite to its traditional Tools suite. These solutions bring new technical challenges and requirements for I/O support.

SAS software has evolved considerably over the decades as the product and the market grew. This evolution started with monolithic institutional computing in the 1970s but adapted to the success of sophisticated CISC (complex instruction set computer) mini-computers, the dominance of the Microsoft Windows operating system on the desktop, proprietary and “open” operating systems, and the shift from CISC to RISC (reduced instruction set computer) computing.

This history is summarized as follows:

- 1970s SAS evolves from a university research project to a for-profit corporation; the initial implementation was on IBM mainframes in PL/1.
- 1980s Market growth of super-minicomputers drives a code migration to the C programming language and a major re-structuring of architecture with host layering.
- 1990s Rapid evolution in RISC systems, storage, and networking technology demands a *vertical* performance focus.

2000s Lower costs mean that SMP (symmetric multiprocessing) and 64 bits become mainstream and demand a new *horizontal* performance focus.

The historical tool-centric approach for using SAS software strongly influenced the I/O architecture of the software. Data from data sources are often read-only and taken directly from a customer's own sources, or from a data warehouse or datamart that is created through an ETL (Extract, Transform, and Load) process on a customer's large-scale, operational data stores.

This resulted in the creation of a model for I/O processing that can be classified as *analytical* I/O. This model reflects the data volumes and I/O patterns that are required to do high-performance analytical calculations on the user's data. The analytical I/O model typically involves large volumes of data; terabyte and even petabyte data sets are not uncommon. Additionally, because the data comes from customer operational stores, result sets and even table column widths are often very large – thousands of columns are encountered in these cases.

The data access patterns are sometimes order-sensitive, which requires either intermediate sorts, if the subset of the data that are being processed is small enough, or indexing technologies. SAS software has I/O models that support sophisticated threaded access to the indexes that are created during ETL processing and data set loading.

Analytical I/O such as forecasting or pricing model construction tends to be *skip-sequential*. That is, the code will read segments of the data set (sometimes hundreds of records; sometimes millions) sequentially, but then will rewind to the start of the group or will skip to another group (based on the dimensions of the analytical process, such as sales periods, product groupings, case numbers, etc.).

Including solutions in the SAS product suite has changed the software development requirements significantly in a number of key areas. The solutions are designed to take advantage of the analytical processing power via an n-tier architecture. However, the solutions can also introduce new requirements because they are more interactive with customers and are often connected to third-party architectures.

An example of these new requirements is the need for new metadata storage to track the details of both back-end processing such as the ETL operations and front-end operations such as report generation, user profiles, and security. J2EE-based application tiers are used to provide scalability for the user-interface, but require object persistence according to the application server's architecture. Web-based Distributed Authoring and Versioning (WebDAV) technology is used to create, for example, repositories for reports or information portal elements, and require database technology to manage object hierarchy, permissions, and searching.

THE NEW REQUIREMENTS FOR SAS SOFTWARE

Introducing the new non-analytical I/O to SAS software brings a new set of fundamental product requirements, which are very different than the requirements for traditional analytical I/O operations. The new requirements are:

- Transactional Store: ACID
- Multi-user: Hundreds to thousands of clients
- SMP and cluster exploitation
- New, non-analytic data types
- Seamless interoperation with historical I/O models

Clearly for SAS, this means that a relational database that meets the company's integration and scalability requirements have to be part of the product mix. SAS software has offered a *driver* interface to databases since the 1980s, but this access largely focused on integrating customer operational data that is stored in those databases into SAS analytical processes, or supporting enterprise-class ETL operations. SAS was moving from an issue of interoperability to one of integration. Unsurprisingly, SAS didn't want to do this in a way that gave competitive ground to the database vendors who were beginning to edge into the applications market themselves.

This shifting market requirement resulted in a new internal design called SAS Table Services. SAS Table Services is a total redesign of SAS I/O functionality with an eye towards the future. A key requirement was to retain the historical analytical interfaces – not just functionality but also performance characteristics. To those analytical interfaces, SAS added a new ODBC 3-based set of interfaces with the accompanying semantics that shifted from a rectangular table to a fully relational model.

In addition to bringing relational and transactional semantics to the I/O model, SAS had to maintain its historical performance edge over the competition and provide increased *horizontal* scalability on very large-scale, computing resources. To support today's requirements, the architecture not only has to be thread-safe but thread-enabled; it must take advantage of SMP configurations that are beginning to dominate the SAS customer base. To maintain future growth, the architecture has to lend itself well to exploitations of clustering (in many and varying forms) and the evolution of storage and networking products that go with clustering solutions.

In order to support the legacy I/O models and access to commercial third-party databases, a plug-in driver model was also a requirement. The plug-in architecture has to be robust enough so that drivers could call other drivers to implement complex layering, such as an SQL optimizer that was implemented as a driver, but could send dialect-specific SQL to specific underlying providers.

The architecture finally manifested itself as a suite of services and a server that could support it in a way that is similar to the embedded versus server models of Firebird. SAS Table Services is a thread-safe I/O provider architecture. SAS Table Server uses the

services suite, but adds connection management and resource management to provide a scalable solution for n-tier deployment. SAS Table Server provides standards-based interfaces and higher-performing proprietary interfaces. This means that it can act as the object persistence manager for an application server, but also provide access to federated views of the analytical and transactional storage models for mid-tier and server-side processing.

Finally, in order to support both new (relational, transactional) and historical (tabular) I/O models, the new I/O architecture must include the necessary functionality to federate usage of either model by either style of client, within the limits of the interfaces. Therefore, SAS Table Server must support an SQL language processor of its own that can be used to query against traditional analytical I/O and identify when ODBC 3-style operations reference data types, formats, and function calls that are unique to the SAS environment, and process them appropriately. The federated view includes the ability to execute queries that span databases and providers, with limits on transactional processing.

In short, SAS needs one architecture and one interface library for all SAS I/O requirements.

Enterprise Class Customers

At the same time that more sophisticated relational and transactional storage technology had to be introduced into the suite of SAS software products, the customer environment was also significantly changing.

For much of its history, SAS software usage fell into two fundamental categories – the individual interactive user and the background or batch process. The interactive user was predominantly a Windows operating system desktop user, and batch operations were done in a variety of modes on mainframe systems such as System 370 systems that run MVS or Alpha-based implementations running OpenVMS.

However, the increased influence of UNIX, along with the evolution of reasonably-priced SMP computing, changed the landscape of the market considerably. Over the last decade, most of the SAS Fortune 500 customers have run on powerful SMP configurations that were previously reserved for super-computing tasks. Even desktop systems are becoming more commonly multithreaded, either through multi-core or hyper-threading implementations.

Today, the customers who invest in solutions to solve enterprise-class problems are running them on enterprise-class hardware and expect their software solutions to capitalize on that hardware.

Typical deployments are on 64-bit systems. Each system is an SMP configuration with 4-8 CPUs. Each system is also memory-rich with 16-32GB of RAM. These systems are

connected via high-speed networking and storage links, and typically have terabytes of disk space.

For many SAS customers, time is the one non-renewable resource. Business processing cycles – which are increasingly complex in global economies – mean that periodic processing jobs must be done in limited windows of time. If a business has only a few hours between shift changes in major markets to rebuild a critical data warehouse that is used for daily reporting, then the software must be able to do the work in that time. In summary, consuming system resources to complete a job in a fixed window of time is a good trade-off.

Benefits and Challenges of Open Source

Given that SAS understands the business value of tightly integrating a relational database with robust transactional capabilities, the next decision that any company deals with is “Build vs. Buy”. For SAS, this was not a terribly difficult decision. Every man-year that is spent implementing a database technology is a man-year that is taken away from core areas of expertise in analytical processing and solutions design. There are too many good database technologies – both open source and proprietary – to warrant exploring a “do it yourself” approach.

The next step was evaluating open source versus a number of potential commercial proprietary partners. Using open source software was attractive for many reasons. It provides the opportunity to leverage tremendous pre-existing experience in database technology. Open source software also offers the capability of leveraging a community of participants that would amplify the effect of SAS development and testing by enabling more users and developers to use and extend the technology than SAS could afford to employ.

Before SAS could just jump into an open source community effort, there were several challenges to address as well as the benefits. First and foremost, SAS needed to find a technology solution that fit with the company’s requirements. However, SAS also had practical concerns to address such as how a commercially-focused company would interact with an open source community.

Aligning Design Goals. There are many reasons why open source projects are developed and sustained over time. Sometimes these projects preserve a technology when a business driver no longer exists. Sometimes they promote an academically pure approach to a technology. Sometimes there is a need to explore a technology for potential future commercial value. Sometimes they are developed just for fun. *Few of these are compatible with what a commercial venture needs from a product that is to be shipped to customers on market-driven schedules.*

Aligning Release Requirements. Even when a project has technical objectives that fit well with the potential consumer’s needs, it is often difficult to synchronize release requirements. A feature may be a requirement for a given market-driven release, but may

not yet be on the priority list for the community at large. In other situations, the community at large may be focusing on a very large-scale project that makes it difficult to manage having a stable instance of the product to release incrementally.

Aligning Product Delivery Dates. This is a direct consequence of misaligned design or release goals. Market forces that are often beyond our direct control drive commercial products. Additionally, integration and testing with a complex product suite may require far more time for the commercial venture than is allowed for in the open source community timeline. Finally, the drivers behind why something is open source can sometimes result in a valid but difficult point of view of “it is ready when it is ready.”

License Requirements and Restrictions. There are dozens of versions of open source licenses that are adopted by various open source communities. Some are loosely designed to allow maximum participation and use, while others are designed to encourage specific intellectual property viewpoints or business models. In short, some licenses make it difficult for a commercial product to embrace an open source technology, while others are open to such uses.

Infrastructure Compatibility. Many large-scale commercial endeavors have constructed internal infrastructures that support their particular product requirements and software engineering styles. This is mostly true when a company has been in the software development business long before the evolution of open source standards. Many open source projects make assumptions about the use or deployment of their project which makes it difficult to adapt to commercial requirements for software portability, source management, build, debug, and test environments.

The Case for Firebird and Vulcan

SAS reviewed a variety of open source database technologies to assess (based on the previously stated critical characteristics) which one might be best suited to integrate with SAS products. This search included PostgreSQL, MySQL, and Firebird, among others. SAS needed to find an open source project that matched its technical requirements and would integrate successfully.

At this time, the Firebird community was beginning the Vulcan project, led by one of the original designers and implementers of the overall Firebird architecture. Based on discussions with the IBPhoenix team it appeared that Vulcan had the potential to be the technology that SAS needed. However, there were a number of critical issues that affected the decision.

True Transactions. Firebird supports a full two-phase commit model for databases, which SAS required for its application. At the time of the analysis, not all open source databases had the same level of support for a complete ACID model.

Embedded Support. SAS has had a lot of experience with database performance issues and integration challenges that required using some variation of a remote interface. SAS

felt that an embedded implementation was far more attractive than one that required additional parameter marshalling and “hops” via inter-process communication pipes or network stacks. During this time, Firebird was one of the few products that offered an embedded model.

Architectural Compatibility. In addition to the embeddable nature of Firebird was the benefit of the overall product architecture. Firebird’s architecture (and even more so that of Vulcan) modeled SAS views about the relationship between consumers and providers. It also shared SAS views on the value of abstraction and interface layering, and the nature of modular plug-in support for extensibility.

Scalability Objectives. Because true SMP support was required, the Vulcan project showed great promise. Many database technologies depended solely on process replication to achieve scalability, which conflicted with SAS server design and the need for embedded support.

Portability Objectives. Conversations with IBPhoenix indicated that the code-base was already on its way to achieving the portability that SAS needed to host the database on the full range of SAS platforms. The initiation of the Vulcan project was a chance to clean up issues that were specific to 64-bit architectures.

Active Community. SAS observed that the Firebird community was actively involved in the evolution of the database.

Licensing Flexibility. The licensing model for Firebird enabled SAS to deeply integrate the Vulcan technology in SAS products, with reasonable requirements for publishing any changes to the open source. The licensing did not include the “touch of death” feature that is found in some open source licenses, where direct interface of proprietary and open source software forces the proprietary software to become open source itself.

Table Server and Vulcan

Once the decision was made to use Vulcan as the code base for providing an embeddable driver for an integrated database, it was time to make them fit together.

SAS uses Vulcan in its embedded mode and is dynamically loaded into the address space of the consumer of the SAS Table Services architecture. For the following discussion, the consumer is the SAS Table Server, a threaded standalone process that provides back-end I/O services for SAS mid-tier and client applications and interfaces used by third-party data stores that are associated with SAS Solutions. The SAS Table Server includes the following primary functions:

Connection Management. SAS Table Server handles connection management by using SAS-specific connection string syntax. Connections are made via standard ODBC or JDBC remote interfaces, a COM object interface on a Windows operating system, or a proprietary connection interface called IOM (Integrated Object Model). The connection

management layer handles connection lifetime and any needed parameter marshalling. It also determines characteristics of the remote side such as preferred locale and character set.

Authentication. SAS Table Server handles all authentications on behalf of all providers in the SAS Table Server architecture. For some providers, this requires synchronization between the underlying security model in the provider and SAS' security model. However, the embedded nature of Vulcan combined with a SAS-private deployment allows SAS to run in a mode in which Vulcan security is not used (at this time).

Resource Management. SAS Table Server is responsible for managing the incoming connections and matching them to a pool of threads that has been sized based on the server requirements and the physical system requirements, via administrative interfaces. The Connection Manager enables more connections that can be expressed as individual threads by pooling a set of host-native threads and assigning connection tasks to them as tasks complete. The resource management does not manage the relationship between the threads, only task assignments to execution threads.

Provider Management. SAS Table Server uses the connection string data to identify which provider(s) are needed to support the request – a given request may require multiple providers in a stack or as a cross-provider join. The Provider Management function arranges to dynamically load (and unload where possible) driver modules to support the needed functions. Vulcan is supported as a dynamically loaded driver that is linked against the “Why-Valve” library.

SQL Evaluation. SAS Table Server evaluates the SQL to determine if it must be recomposed or restructured for execution by multiple drivers or to support interfaces with SAS-specific functionality.

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

The Vulcan Driver

A SAS Table Server driver that is written for Vulcan translates SAS 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. For example, SAS Table Server returns a SQLSTATE to reflect the result of an operation, which is common across all providers. This enables a client to detect an error in a provider-neutral way, and then optionally request provider-specific diagnostic information if that is meaningful for the application or end-user.

There are a number of important technical features of Vulcan that facilitate tight integration with the SAS Table Server:

- The embedded model of Vulcan provides an efficient database engine, without the performance costs of an additional hop to a provider.
- Vulcan's locking model supports multiple instances of SAS Table Server and accesses the same database.
- Vulcan's thread-safe implementation partners effectively with SAS Table Server's thread pooling and resource management to support scalability.
- External administration tools can interoperate with a SAS Table Server instance. For example, Firebird's GBAK command line tool can still be used to do "hot backups" of databases that are managed by running SAS Table Server instances.

In the first release of SAS Table Server in 2006, SAS does not expect end users to directly interact with Vulcan. That is, SAS wants end users to go through SAS Table Server to interact with "the SAS" transactional data store. SAS provides an administrative interface that supports database administration functions via a Java application, which is common to all SAS products. SAS will ship adapted versions of the GBAK tool, etc. to support database administration and system management functions.

SAS anticipates the possibility that one SAS user will also be using Vulcan in its more traditional standalone database mode. As such, SAS has taken steps to ensure that the SAS deployment of Vulcan and a user deployment cannot collide.

The SAS Table Server driver explicitly sets process environment variables (VULCAN, VULCAN_CONF, etc.) to point to the SAS-installed instance of Vulcan. Additionally, the command line tools were modified to automatically detect where they are run from and set the environment symbols accordingly. So if the SAS version of ISQL is run, it will connect only to the SAS instance of the Vulcan database domain.

Building Vulcan across Platforms

An area in which SAS is not yet fully leveraging the benefits of the open source community is unification of the source versions. A combination of factors, including differences in the SAS build environment, SAS-specific features, and stability requirements mean that it is very difficult to keep the SAS version of Vulcan in synch with the SourceForge repository.

Much of this is driven by the short-term goals that SAS has for its commercial product. The commercial priority is portability and scalability implementations that are suitable for commercial release in 2006. By contrast, the community priority is in architectural review and revision. Both are ultimately required, but compete with each other in the near-term. Additionally, the community focus has been on Firebird much more than on Vulcan due to the release goals of Firebird 2.0.

The unfortunate consequence of this is that right now, for SAS there are two versions of Vulcan. The SourceForge version is changing rapidly, with class restructuring, etc., and driving much needed architectural changes - at the cost of stability. The privately

maintained SAS version changes much more slowly, and undergoes rigorous testing on multiple platforms with diverse loads on a weekly basis. Both approaches meet the distinct goals of SAS and the community, but conflict in the short term. This conflict is resulting in unwanted duplication of effort at this point.

SAS-Specific Vulcan

SAS deploys its software products on the OpenVMS, MVS, Windows, and Linux operating systems, as well as virtually all modern UNIX 64-bit systems. To accomplish this, SAS uses an in-house source management, build and debug environment that supports multi-platform builds. This lets a developer make a change and perform cross-platform tests before pushing to the code base, as well as supporting configure/install/deploy mechanisms that are common to all platforms (customers often buy a suite of products on a variety of host platforms).

Vulcan introduced some new challenges to the SAS build environment, not the least was the use of C++ as the implementation language - compilers on some platforms (MVS in particular) introduce interesting constraints on building, such as header file locations. The host-specific features of SAS deployments mean that SAS had to make some adaptations to its process, and some changes to Vulcan.

Issues with the SAS build environment mean that SAS does not mirror the open source build tree directly, but instead restructures it when the code is refreshed through update or push fixes via a commit. For example, in the SAS environment, file names cannot be duplicated in different directories (only one copy of "exe.h" is allowed, etc). Automated tools allow SAS developers to translate source files between the SAS environment and the SourceForge environment.

SAS regularly pulls specific bug fixes into its code base from SourceForge. The intent is to provide batches of updates when bugs are found and fixed - though SAS needs to improve that process so that it is more frequent.

In order to identify the changes that SAS makes, all changes are bracketed with `#ifdef SAS_FIREBIRD`. This enables SAS developers to easily automate tools to identify files that are candidates for pushing back to SourceForge. For cases in which proprietary extensions were made, SAS can publish the source code to any module that also exists in the open source repository and ensure that it builds correctly if it is used by an external open source user. (The actual proprietary algorithms or SAS-specific implementations are stored in separate source files, called/included from conventional source, and are not published.)

Some changes have been pushed already, but SAS is unsure of their value. For example, SAS modified the source in a numerous places to conditionally build with a shared cache for all connections versus a cache-per-thread. This change was required so that SAS could achieve specific stability and scalability goals in the near term. However, it is entirely possible that this mode is not being used by anyone else – the default is

SHARED_CACHE, which is Jim Starkey's original model; SAS builds without SHARED_CACHE.

Other changes are specific for internal use and SAS has not yet brought them forward to the open source community, although SAS could use guidance on their value. For example, SAS supports per-statement cancel operations. The SAS Table Server architecture specifies that a statement can be cancelled during PREPARE, EXECUTE, or FETCH to terminate long-running queries (this is done either administratively or via client-specific interfaces). The nature of the SAS Table Server is such that there are always administrative threads that can respond to the inbound cancel request, so SAS allows them to use a Statement Handle to terminate a statement. This causes a specific error to be posted, which includes information on whether the statement can be restarted or must be released and re-constructed if it is to be used again.

Another example is modification of the command line tools (ISQL, GBAK, etc.) to detect whether they have been run from the SAS installation of Vulcan and coerce the environment variables to connect only to the SAS instance of Vulcan.

The Future

Since Vulcan was branched from Firebird, a lot has changed in Firebird. New features, performance enhancements, and straightforward bug-fixes have been pushed, that have greatly enhanced the value of the Firebird 2.0 code base. Many of these features (for example, expressions in indexes) are critical for SAS' future use.

There have been several discussions about the challenges in finding a way to merge them together. SAS believes that this is essential for the long-term overall success of Firebird.

The Firebird community has made it possible for SAS to tightly integrate a transactional database far more quickly than the company could have accomplished on its own. The performance and scalability potential of the Vulcan design fits well with enterprise-class software.

Short-term divergences of goals have been unavoidable and have limited SAS' participation in the community. A longer-term goal for SAS is to re-synch with the one true Vulcan and be an active participant in advocating for embedded usage of Vulcan.

More Information

See the paper "Embedded Classic" which explains how SAS uses 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 the SAS infrastructure.

See the related paper “SAS/Firebird Performance Testing Strategy” which describes how SAS stress tests the scalability of the code.

Credits

Tom Cole is the primary author of this paper. As is often the case, many others contributed both content and time in the form of technical reviews. Specifically, thanks go to Steve Krueger, Bill Oliver, and David Shamlin for their invaluable input.

Contact Information

Comments and questions are valued and encouraged. Contact the author:

Tom Cole, Principal Staff Technical Advisor, Platform R&D
SAS Institute Inc.
Cary, NC
tom.cole@sas.com

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies. Copyright © 2005, SAS Institute Inc. All rights reserved.