The Scalable Performance Data Server™: Real World Experiences

Daniel Sargent
Clinton Rickards
Oxford Health Plans, Norwalk, CT

Abstract
The Scalable Performance Data Server (SPDS) is a high performance data store, and is SAS Institute’s foray into true client-server processing. It offers user sites a number of benefits not available with standard SAS data engines. These benefits include virtually unlimited data set sizes, performance gains from parallel processing, advanced indexing, and better sorting, and more efficient disk space utilization. It also supplies a comprehensive set of tools to control user access. Being a true client-server product, it allows non-SAS applications access to the data via ODBC and an API. Finally, its flexibility allows the creation of secure and robust production and development environments.

Oxford Health Plans has been using SPDS since early 1997. This paper will describe our experiences in the real world of using SPDS as the host for a data mart. We will present an overview of SPDS, describe the configuration and setup of the product in our Sun Solaris environment, how it has been used to our advantage, and some of the early mistakes and corrections that we made. We will also present a summary of performance comparisons between SPDS and standard SAS.

Introduction
Although the “normal” SAS engines are fairly efficient, they tend to bog down when working with high volumes of data, particularly when trying to satisfy certain SQL queries, WHERE conditions, or many users. The advent of data warehouses and data marts, with their massive volumes of data, has exacerbated these problems. The Scalable Performance Data Server is a true high-performance client-server database product designed for the data warehouse market.

The Medical Analysis Group at Oxford Health Plans deployed a medical data mart in October, 1997, to serve the needs of approximately 30 users and 15 departments throughout the company. As of April 30, 1998, it housed data for approximately 3 million members, 400,000 thousand providers, and 107 million claims. Since the initial introduction, utilizing version 1.2 of SPDS, we have evaluated our physical design and determined that we could achieve far better performance. In June, 1998, we deployed a new data mart with the same table design but a totally new physical design. This paper will outline the options affecting performance, the choices we made, and the performance gains we achieved.

Brief Tutorial on SPDS
SPDS relies heavily on SMP (Symmetric Multi-Processor) technology to achieve its performance improvements as compared to standard SAS data sets. It uses multiple threads, or processes, across the available processors to satisfy the requirements of a particular request. Matching the physical design of the database to the SMP capabilities of SPDS is critical to achieving high performance. A number of techniques, used in combination, are available to a SPDS administrator. These techniques include partitioning, separation of components, and indexing.

Partitioning is the splitting of a data set into a number of physical files. Each processor can spawn multiple threads, each of which can process a partition. SPDS manages these multiple partitions so that they are viewed as a single data set. The partition size is set in three ways: by default; in the spdsserv.parm file, used when the data server is started; or as a data set option. The process used by the SPDS data server to create the physical files will be discussed later.

Separation of components is the process of distributing data across disk drives controlled by multiple controllers. Separating components reduces hardware contention and increases throughput. There are three components that must be kept as separate as possible:

1. WORK library
2. SPDS metadata and data
3. SPDS indexes

WORK library is the normal SAS WORK library. It is heavily used by any SAS program and can become a bottleneck.

SPDS metadata and data is the actual data being stored. Metadata and data are analogous to the descriptor and data components of traditional SAS data sets.

SPDS indexes have counterparts in traditional SAS data sets but can be separately managed under SPDS.
Indexed data sets require more disk space and CPU resources to build and maintain than non-indexed data sets but can satisfy certain queries much faster by allowing more direct access to the data. SPDS supports more index types than the balanced b-tree index available with the Base SAS engine. These new types create smaller, more efficient indexes. The following index types are available with SPDS:

- **hybrid bitmap** is most useful for keys of non-unique, discrete values. The SPDS documentation indicates that cardinality is not an issue. For example, state codes, zip codes, or dates might be well served by a hybrid bitmap index.

- **global b-tree** is most useful for unique or nearly unique keys. Nearly unique keys occur when each non-unique key value occurs as a small percentage of the total data set. A global b-tree index might be appropriate for social security numbers and policy numbers, for example.

- **segmented b-tree** is a variation of the global b-tree. The index is broken into multiple, fixed length files which can be processed in parallel. If queries commonly use “less than” or “greater than” WHERE clauses, a segmented b-tree index will be more efficient than a global b-tree index. You must create a global b-tree index to create a segmented b-tree index.

**Defining and accessing a SPDS domain**

A SPDS domain is the data managed by an SPDS data server. The domain is defined in the libnames.parm file that is processed when the SPDS data server starts. The libnames.parm file contains a libname statement for each domain. The libname statement contains options that define a directory for each major component of a SPDS domain: metadata, data, and index. The general syntax is:

```
libname libref 'domain-name'
pathname=directory1
roptions="datapath=('directory2' 'directory3' ... )
indexpath=('directory4' 'directory5' ... )"
owner=ownerid;
```

SPDS administrators should always use “roptions”, or reserved options, to define the domain. This prevents users with write access to the SPDS library from extending the library themselves. Similarly, defining the owner of the library prevents users from creating new data sets unless they have been granted permission to do so.

- `pathname=` is the directory to store metadata
- `datapath=` is one or more directories to store data
- `indexpath=` is one or more directories to store indexes

Accessing a SPDS domain from a client requires a new libname engine: SASSPDS. This engine must be installed on all client servers (except ODBC) that need to access an SPDS domain. A number of new options are used to identify the SPDS database to access, the user, and the password. The general syntax is:

```
libname libref SASSPDS 'domain-name'
server=servername.data-server-name
user='userid'
password='password';
```

- ‘domain-name’ is the domain name defined on the server
- `server=` is the name of the physical server and the SPDS instance running on that server. For example, UNXSERV1.SPDSNAME identifies the SPDS instance SPDSNAME running on the server UNXSERV1.

We will not be discussing these and other SPDS libname statement options for the server and client sides in detail. Please refer to the SPDS documentation for more information.

Once the client side libname has been created, users access SPDS data using the familiar `libref.dataset` syntax. SPDS is designed to be totally compatible with traditional SAS syntax and processing.

**How SPDS builds partitions**

When writing a SPDS data set, SPDS will divide the data set into multiple partitions, or physical files. The size of each partition is set by the PARTSIZE= SPDS option (see Partitioning, above). The partition size for a data set can not be changed after the data set is created without rewriting the file. Also, the partition size can not be less than the MINPARTSIZE= value, specified in the spdsserv.parm file.

When a SPDS domain is defined in the libnames.parm file, one or more directory paths are specified to hold the individual partitions. For example:

```
libname spdspapr
pathname=/spds-paper-metadata
roptions=
  datapath=(' /spds-paper-directory1'
                   '/spds-paper-directory2'
                   '/spds-paper-directory3'
                   '/spds-paper-directory4'
            '
indexpath=(' /spds-paper-directory1'
                   '/spds-paper-directory2'
                   '/spds-paper-directory3'
                   '/spds-paper-directory4'
            '
owner=ownerid;
```

defines a domain named “spdspapr” that stores its metadata in the directory `/spds-paper-directory1` and its
partitions in the four directories listed in the datapath option. Note the use of roptions; this reserves control of the domain to the SPDS administrator.

Let us consider a simplified example to illustrate how SPDS creates partitions. Assume a SPDS data set with an observation length of 1K, 2 million observations, and a partition size of 16 MB. To keep the arithmetic simple, further assume there is no disk space overhead.

Data spdsppapr.example1 (partsize=16);
array texts (8) $128;
do I = 1 to 2000000;
output;
end;
run;

SPDS will begin writing observations into the first partition in the first directory (/spds-paper-directory1) until the partition size of 16MB is reached (16,384 observations). SPDS then writes observations to a partition in the second directory (/spds-paper-directory2) until the partition size of 16MB is reached. SPDS then writes to the third directory (/spds-paper-directory3), and then the fourth (/spds-paper-directory4). When the fourth partition is full, SPDS begins the cycle again, starting with the first directory (/spds-paper-directory1). SPDS ends up creating 123 partitions in the 4 directories.

How SPDS builds indexes

When a SPDS domain is defined in the libnames.parm file, one or more directory paths are specified to hold the index files. Extending our example for building partitions:

libname spdsppapr
pathname=/spds-paper-metadata
roptions=""
datapath=(’/spds-paper-directory1’
 ’/spds-paper-directory2’
 ’/spds-paper-directory3’
 ’/spds-paper-directory4’
 )
indexpath=(’/spds-paper-index1’
 ’/spds-paper-index2’
 ’/spds-paper-index3’
 ’/spds-paper-index4’
 )
"

identifies four directories to store the indexes for domain “spdsppapr”. Unlike data partitions, index files are not partitioned. Index files are written to the first directory until the directory is full, then the second directory is used, and so on.

Our changing server environment

As mentioned in the abstract and introduction, our medical data mart has been using SPDS Version 1.2 since October, 1997. We have been experiencing adequate performance, i.e. no complaints from our users. However, our review of the 2.0 Beta documentation and the results of some experiments with the 2.0 Beta software led us to redesign our hardware and software hosting the data mart. This redesign reflects our growing understanding of how to best take advantage of the SPDS software.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Old host</th>
<th>New host</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>Sun E4000</td>
<td>Sun E5000</td>
</tr>
<tr>
<td>OS</td>
<td>Solaris 2.5</td>
<td>Solaris 2.6</td>
</tr>
<tr>
<td>SPDS Version</td>
<td>1.2</td>
<td>2.0</td>
</tr>
<tr>
<td>Processors</td>
<td>8x167 MHz</td>
<td>6x250 MHz</td>
</tr>
<tr>
<td>RAM</td>
<td>2 GB</td>
<td>3GB</td>
</tr>
<tr>
<td>Model 214</td>
<td>3 - 528 GB</td>
<td>4 - 704 GB</td>
</tr>
<tr>
<td>SCSI disk arrays</td>
<td>3 - 528 GB total capacity</td>
<td>4 - 704 GB total capacity</td>
</tr>
</tbody>
</table>

We moved to a new Sun server mostly a convenience arising from a larger effort to upgrade a number of the Medical Analysis Group’s servers. An E5000 server is essentially the same as an E4000 in a different case. The new E5000 is arguably slightly more powerful than the old E4000, due to the faster (but fewer) processors, additional RAM, and the latest software. However, we believe that the most important factors in our performance improvement were the addition of more disk space and the effective utilization of that space to minimize contention and maximize the amount of parallel processing.

Test results

Most of the tests that are reported in this paper were performed on our new Sun E5000 server with no other users on the server. Our LABS data set was used as the base data. It contains approximately 10.6 million observations and consumes roughly 1.6 gigabytes of disk space. We also loaded the LABS data into standard Version 6.12 data sets for comparison. The tests generally consisted of multiple simultaneous runs of identical queries. We measured run time in seconds and disk space in gigabytes (GB) when needed. For experiments 1-3 and 5 the query we ran was:

```sql
/* datalib is either a SPDS library*/
/* or a Base SAS library */
proc sql ;
create table temp as
select *
from datalib.data-set-name
where lob = 'MD' and memsex = 'M' ;
```

Experiment One - Demonstration of the affect of multiple disk arrays. This experiment copied the LABS data set into 4 different SPDS libraries. Each library was
spread across 1, 2, 3, or 4 arrays. A partition size of 16 MB was used for all data sets. The query was run 5 times simultaneously against each data set.

Experiment Three - Comparing the effect of index type on run time. This experiment copied the LABS data set into the same library 8 times. The partition size of the 8 data sets varied from 16 MB to 256 MB. After building the indexes on LOB and MEMSEX appropriate for the test, the same query was run 5 times against all 8 data sets, a total of 40 simultaneous jobs. The average of the run times for each data set was then computed. We also compared the size of the index file created for each type.

The hybrid bitmap is the most efficient index type of the three for this query. It used 35% less run time and 99% less disk space than the global and segmented b-tree indexes, which achieved the second best results. These results were consistent across the different partition sizes. Although this experiment tests only one type of query (looking for specific, non-unique values), these results provide convincing evidence that performance depends on matching the type of index to the queries you are most likely to run.

Experiment Four - Creating multiple indexes. This experiment created 6 indexes on the LABS data set. The Base 6.12 library and the SPDS domain were each spread across 4 arrays. We created 5 hybrid bitmaps and 1 global b-tree.

SPDS has a huge advantage over Base SAS because it is capable of creating multiple indexes simultaneously.

Experiment Five - Demonstrating the effect of partition size on run time. This experiment copied the LABS data set repeatedly into the same library which was spread across 4 arrays. The partition size was varied from 16 MB to 48 MB. The query was then run 5 times simultaneously against each data set.
Effect of Partition Size

The test results show that a larger partition size tended to reduce the run time. 24 MB was an exception for which we have no clear explanation.

Experiment Six - Comparison of creating 2 large data sets. Our production data mart contains another data set called MEMBERS with approximately 3 million observations. This experiment copied MEMBERS and then LABS to a 6.12 library. A second program was run at the same time to copy the data sets to a SPDS library.

<table>
<thead>
<tr>
<th>Partition Size</th>
<th>Run Time (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base 6.12 on 4 arrays</td>
<td>185.4</td>
</tr>
<tr>
<td>16 MB</td>
<td>49.0</td>
</tr>
<tr>
<td>24 MB</td>
<td>51.0</td>
</tr>
<tr>
<td>32 MB</td>
<td>43.0</td>
</tr>
<tr>
<td>48 MB</td>
<td>34.0</td>
</tr>
</tbody>
</table>

SPDS has a clear performance advantage over base SAS.

Experiment Seven - Comparison of SQL joins. We developed a new query that requires a SQL join to complete.

```sql
proc sql;
create table temp as
select *
/* datalib is either a SPDS library*/
/* or a Base SAS library */
from datalib.labs a,
datalib.members b
where lob = 'MD' and memstate = 'CT'
and a.memcode = b.memcode ;
run ;
```

The query was run 5 times concurrently against SPDS and base SAS versions of the data sets. The data was sorted by memcode but had no indexes. We then indexed the data sets - a hybrid bitmap for memstate and a global b-tree for memcode - and reran the queries.

<table>
<thead>
<tr>
<th>Run Time (Seconds)</th>
<th>Run time (Seconds)</th>
<th>Index Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base 6.12</td>
<td>195.0</td>
<td></td>
</tr>
<tr>
<td>SPDS 2.0</td>
<td>176.0</td>
<td></td>
</tr>
<tr>
<td>Performance Gain</td>
<td>10%</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Run Time (Seconds)</th>
<th>Run time (Seconds)</th>
<th>Index Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base 6.12</td>
<td>169.0</td>
<td>301.0</td>
</tr>
<tr>
<td>SPDS 2.0</td>
<td>176.0</td>
<td>362.0</td>
</tr>
<tr>
<td>Performance Gain</td>
<td>-4%</td>
<td>-20%</td>
</tr>
</tbody>
</table>

For comparison, we wrote an equivalent data step merge:

```sas
data temp ;
merge
datalib.labs
   (where=(lob='MD') in=a)
datalib.members
   (where=(memstate='CT') in=b) ;
by memcode ;
if a and b ;
run ;
```

<table>
<thead>
<tr>
<th>Run Time (Seconds)</th>
<th>Run time (Seconds)</th>
<th>Index Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base 6.12</td>
<td>242.0</td>
<td>301.0</td>
</tr>
<tr>
<td>SPDS 2.0</td>
<td>106.0</td>
<td>6.5</td>
</tr>
<tr>
<td>Performance Gain</td>
<td>56%</td>
<td>98%</td>
</tr>
</tbody>
</table>

General rules for high performance
Oxford has begun to develop a set of guidelines to help us achieve high performance with SPDS software without spending a phenomenal amount of money.

1. Use multiple disk hardware. Our benchmarks indicate a tremendous improvement when the data are spread
across multiple disk arrays. In our case, we are now using 4 Sun 214 disk arrays. Each array contains 6 SCSI controllers managing 7 4.2 GB disks, a total of 168 disks (4x6x7). Each array is connected to the server by a 25 Mb Fibre connection.

2. Separate data, indexes, and SAS work onto different sets of controllers to minimize contention.

3. Partition the data. Many partitions allow SPDS to spawn numerous threads to attack the problem.

4. Do not create too many partitions. We have imposed our own limit of approximately 128 partitions per data set. Why this seeming contradiction to Rule 4? First, the amount of data available in a partition may not be worth the overhead of managing the threads. Second, we ran into a problem while testing Version 2.0. In one of our experiments, we used a small (16 MB) partition size and a 29 GB data set. The resulting number of partitions exceeded 1800; we could create the data set but not read it. It is unclear whether the problem lies in SPDS or in the Solaris limitation on the number of open files (currently set at 4096). Increasing the partition size, and hence reducing the number of partitions, eliminated the problem. We hope to do more testing on this issue.

5. Build indexes appropriate to the queries you expect to run most often. The right query will run faster and will save disk space.

6. Instruct SPDS to do the sorting. Care needs to be taken when sorting an SPDS data set. Using the SPDSSORT option on PROC SORT instructs SAS to sort the data set on the SPDS server, presumably a more powerful server than the client. SORTs without this option will result in the sort taking place on the client, increasing the network traffic and utilizing a less powerful server.

7. Allow SPDS to automatically sort the data. SPDS will automatically sort input SPDS data sets when a BY statement is used in a DATA or PROC step and the data set is not sorted by those variables and an appropriate index is not available. Since the resulting data are passed directly to the DATA or PROC step, total I/O is reduced as compared to using a preliminary SORT step. The SPDS option &SPDSBSRT defaults to YES to enable automatic sorting.

8. Create multiple indexes in parallel. Setting the SPDS option &SPDSIASY to YES will allow SPDS to create multiple indexes at once. SPDS must use a great deal of temporary disk space to do this, increasing the risk of running out of disk space, but creating indexes in parallel is much faster than the repeated scans of the data set used by Base SAS.

Future experiments
SPDS 2.0 has several new features that we have not yet been able to test. These include SQL Pass-Thru support, ODBC compliance, an experimental link to Oracle called Oracle Tunnel, and an external API. SAS Institute is working to add Multi-Dimensional Data Base support and some client-side enhancements to further improve procedure performance.

Conclusions
The Medical Analysis Group has been very happy with the performance of SPDS 2.0. Although there have been a few problems with the software, the tremendous performance gains - in one case, dropping run time from 31 seconds to 4 seconds and disk space usage by 10% (Experiment Two) - has justified the time and money we have invested in its use.

For Further information on SPDS and its abilities, contact SAS Institute or visit their SPDS web site at:
http://www.sas.com/software/components/spds.html

References and Acknowledgments
The authors would like to thank our teammates for their help and continued support while we were having the fun of exploring the capabilities of SPDS. We also thank Bubba Talley, SAS Institute Technical Support, and Amitava Ghosh, SPDS Development Manager, for reviewing the paper and patiently answering our questions.


“Scalable Performance Data Server Release 2.0”, HTML available on SAS/SPDS 2.0 installation media, SAS Institute, Inc., Cary, NC, 1996

SAS® and SAS/SPDS® are registered trademarks of SAS Institute, Inc., Cary, NC