Coding For Performance

Robert Rosofsky and Malcolm Rucker
Boston Area SAS Users Group
March 23, 2016

Sentinel Operations Center
Harvard Pilgrim Health Care Institute

This PDF slide deck file includes the following:

1) All slides in landscape mode, one per page

2) Selected slides, repeated with accompanying additional information, in portrait mode

Coding For Performance

Robert Rosofsky and Malcolm Rucker
Boston Area SAS Users Group
March 23, 2016

Sentinel Operations Center
Harvard Pilgrim Health Care Institute

Outline

- Introduction
- Core Concepts
- Benchmark Test Environments
- Individual Techniques
- Acknowledgements
- Questions

Introduction

Why bother coding for performance?

Efficient coding may boost the "bottom line" by:

- Reducing time finding solutions to business problems
- Increasing quality of solutions found (e.g. more time for in-depth and careful analysis)
- Reducing expenditures on hardware upgrades

When to code for performance?

- Many factors may influence the decision
  - Execution times are unacceptably long
  - Program runs multiple times
  - The data volume is “large” compared to your system
  - The data processing is “complex” or “has many steps”
  - The additional benefits outweigh the additional costs

What are the costs?

Additional effort may be needed for:

- Learning
- Designing
- Coding
- Benchmarking Performance
- Quality Control

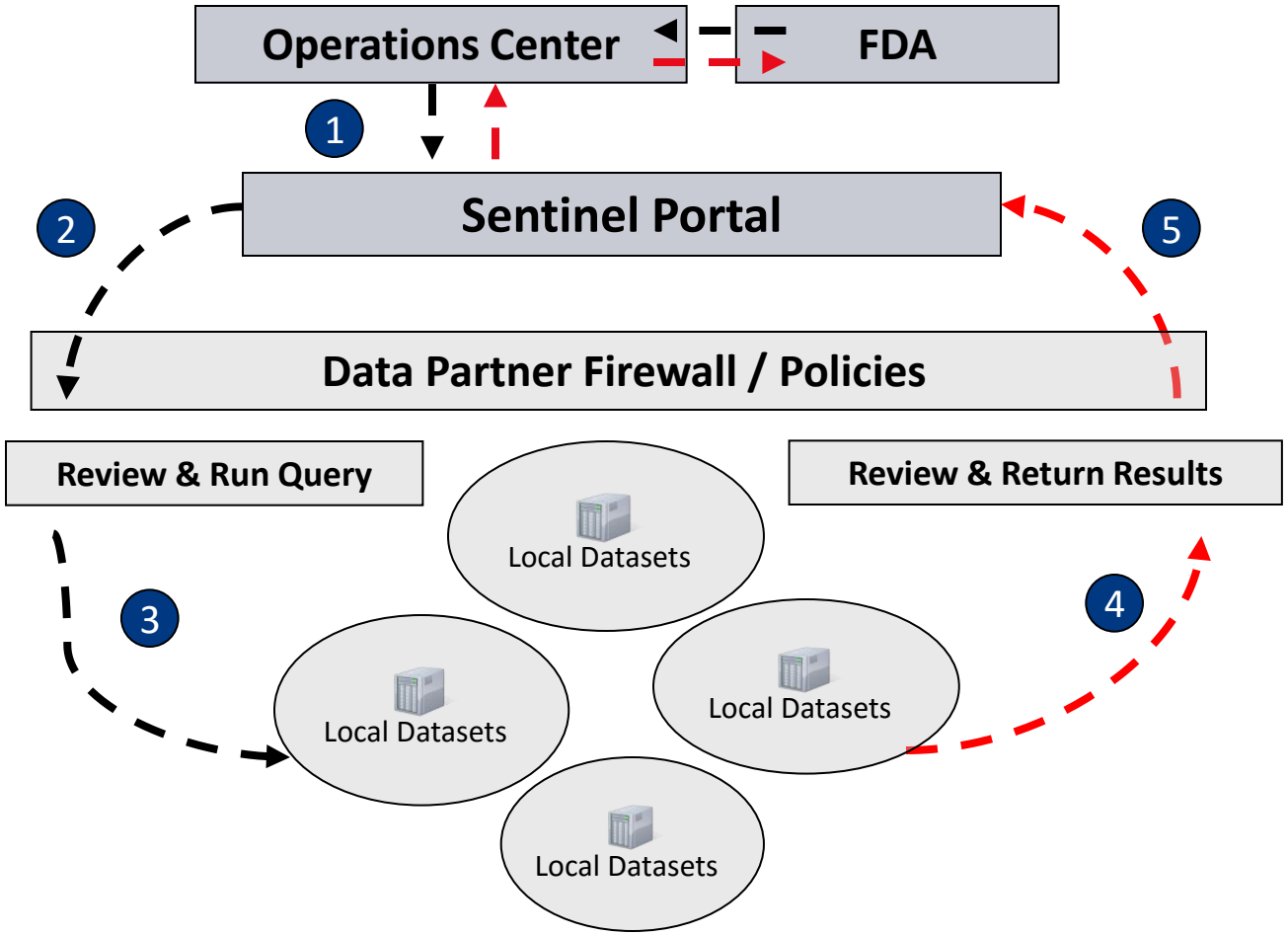
Sentinel Data Partners
N=18 total

HealthCore
HealthCore
WELLPOINT
OPTUM®
Humana Pharmacy Solutions®
KAISER PERMANENTE®
Aetna®
VANDERBILT SCHOOL OF MEDICINE
health care systems research network
Harvard Pilgrim Health Care
MASSACHUSETTS

Sentinel Data Partners (Continued)

- Over 213 million members throughout the network
- National health insurers & Health Maintenance Organizations (HMOs)
- # of members, per DP, from <300,000 to >50 million
- Data time range for some DPs is as early as year 2000
- DP source data are extracted, transformed, and loaded into the Sentinel Common Data Model format (SCDM) as SAS tables queried by distributed SAS programs
- SAS installations at DPs range from SAS 9.3 - SAS 9.4

Sentinel Distributed Data Network



1- Query (an executable program) is submitted by Coordinating Center to the Portal
2- Data Partners retrieve the query
3- Data partners review query and perform analysis locally by executing the distributed program
4- Data partners review results
5- Data partners return results to the Portal

Technical Challenges for Sentinel Distributed Data Network

- Over time, we expect even heavier workloads across the Sentinel distributed data network driven by increasing levels of
  - Data volumes
    - # members, # years, # of data attributes collected
  - Query volumes
    - # of requests, # of query scenarios per request
  - Query complexity
    - Complex analytic techniques and event definitions

Technical Challenges (Continued)

- Traditional coding approaches do not account for the extreme variation found across the network in
  - Data volumes
  - System capabilities
  - System loads

- As a result, we have sometimes found that a query performs well at some sites while at other sites it fails (e.g. runs out of disk space or RAM) or runs for days (e.g. we have seen queries take up to 2 ½ weeks)

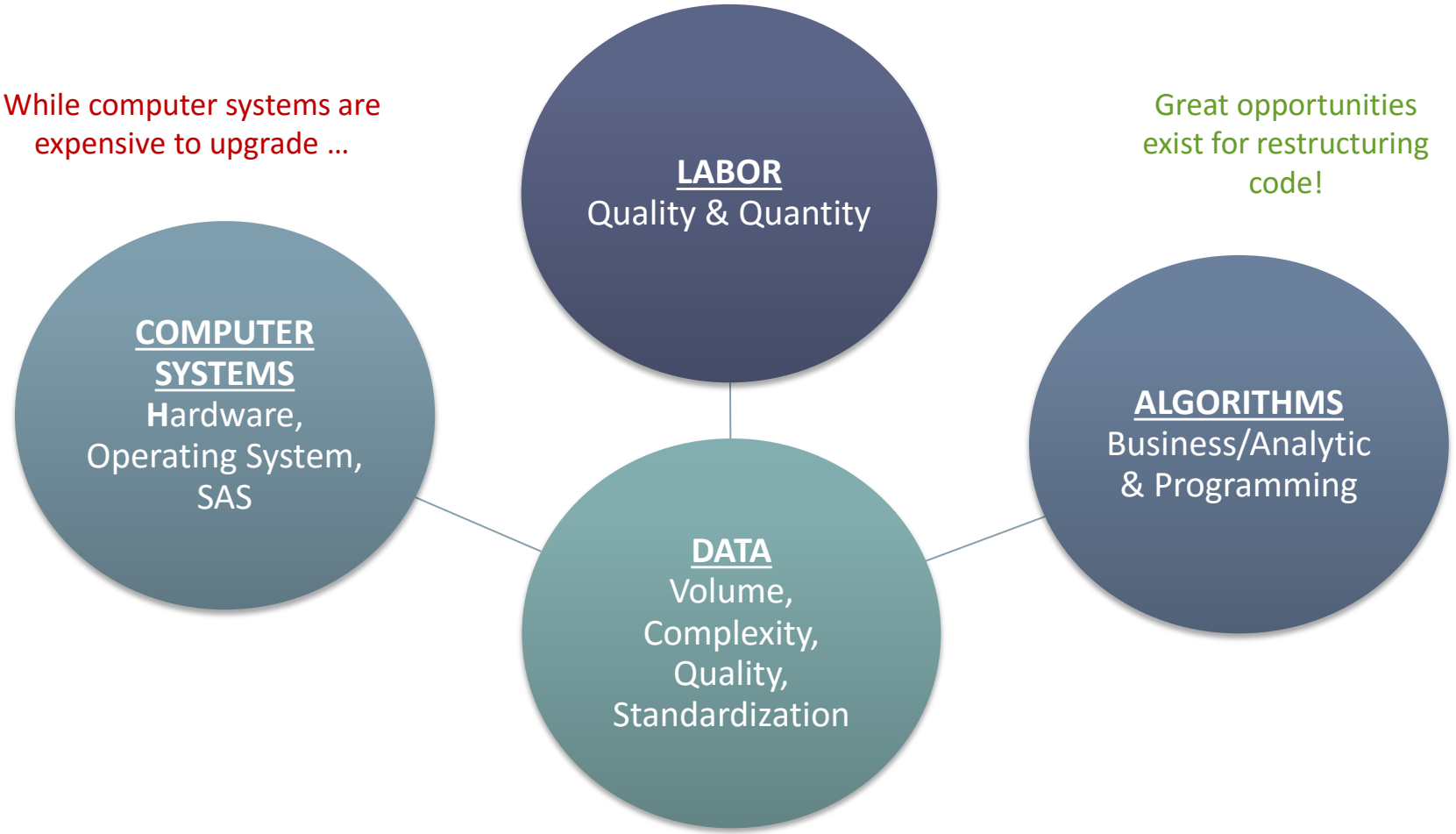
Sentinel Response to Technical Challenges

- The Sentinel Operations Center (SOC) launched an initiative in 2015 to explore alternative methods for storing and querying “Big Data”

- With the help of our Data Partners, we sought to find solutions that significantly boost performance while using available resources more efficiently

Resource Opportunities & Constraints

While computer systems are expensive to upgrade …



Great opportunities exist for restructuring code!

Use-Case: Summary Tables Creation

- The summary tables creation package is run after each data refresh at a Data Partner site
- All diagnosis, drug, and procedure code events are extracted (from typically very large tables), transformed (in multiple ways), and aggregated (multiple times)
- The result are summary tables of prevalent codes and incident codes that are stratified by year, age-group, sex, encounter type

Use-Case: ST Creation (Continued)

- While able to support very quick feasibility analysis (e.g. for sample size estimates), creating the summary tables themselves takes a very long time

- In the Fall and Winter of 2015, we completely rewrote the summary tables creation package as a use-case for alternative approaches to “Big Data”

Use-Case: ST Creation (Continued)

Summarize multiple detail tables (such as these)…

Diagnoses

PatID	VisitDate	DX_Code
2213	2/12/2006	250.0
6235	6/7/2007	300.5
*2 **Billion** Rows*		
8089	8/11/2007	569.6
9292	9/17/2008	655.8

Demographics

PatID	Sex	Birth_Date
2213	M	4/23/1955
6235	F	6/22/1960
Millions of Rows		
8089	M	11/9/1990
9292	F	7/15/1997

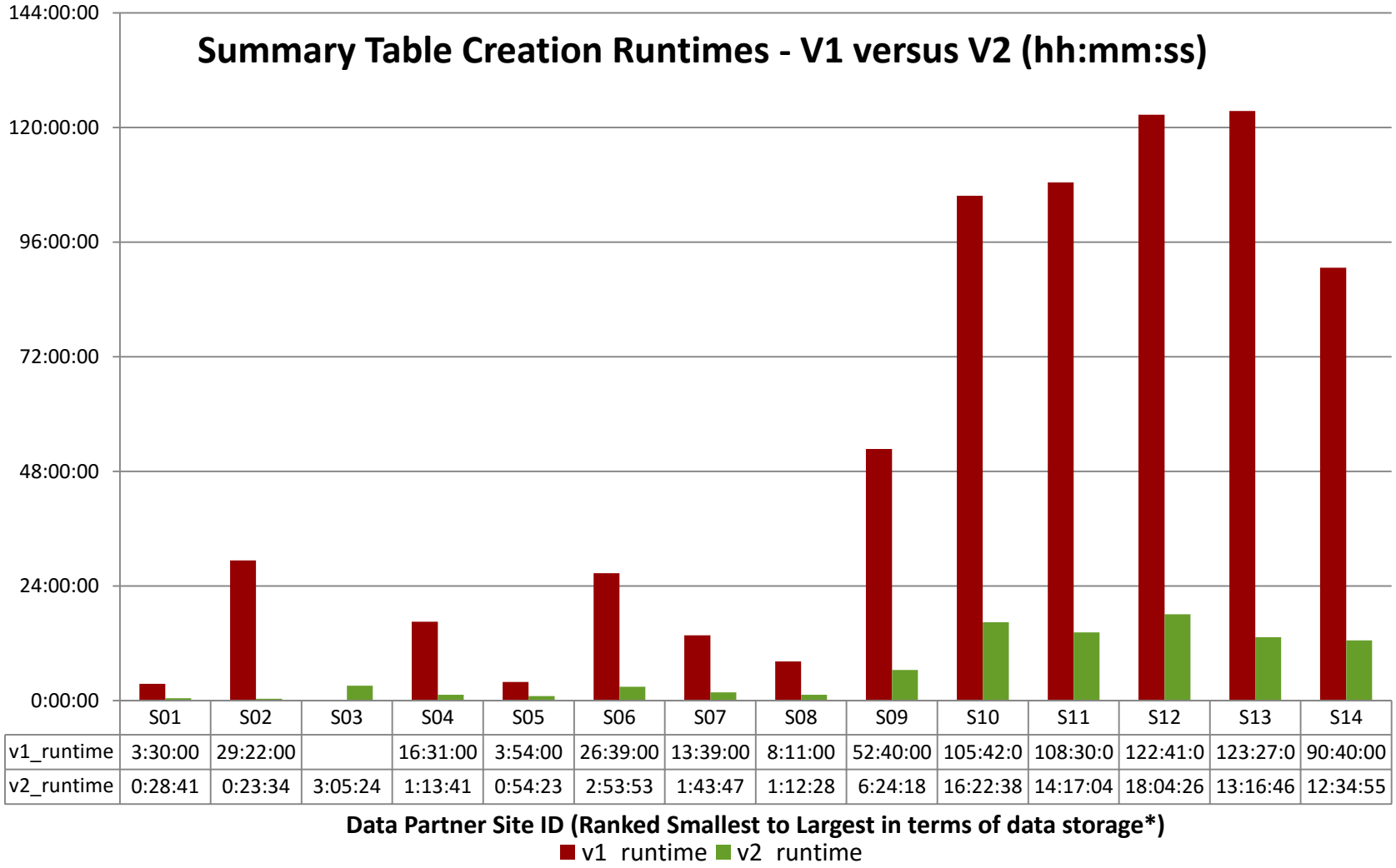
Enrollment

PatID	Enr_Start	Enr_End
2213	1/1/2010	12/31/2010
2213	7/1/2011	12/31/2014
Millions of Rows		
8089	5/1/2014	11/9/2014
9292	12/1/2014	6/30/2015

To something like …

AgeGroup	Sex	Dx_Code	Descr	Year	# Patients	# Events
0-1	M	250	Diabetes	2011	9,456	15,760
2-4	F	493	Asthma	2011	10,554	15,077
5-9	M	410	Acute myocardial infarction	2011	15,800	36,867
10-14	F	V22	Normal pregnancy	2011	7,400	19,733
15-18	M	581	Nephrotic syndrome	2011	9,600	67,200
19-21	F	250	Diabetes	2012	33,400	72,367
22-44	M	493	Asthma	2012	49,000	98,000
45-64	F	410	Acute myocardial infarction	2012	25,900	55,038
0-1	M	581	Nephrotic syndrome	2012	12,200	73,200
2-4	F	410	Acute myocardial infarction	2015	84,200	372,886

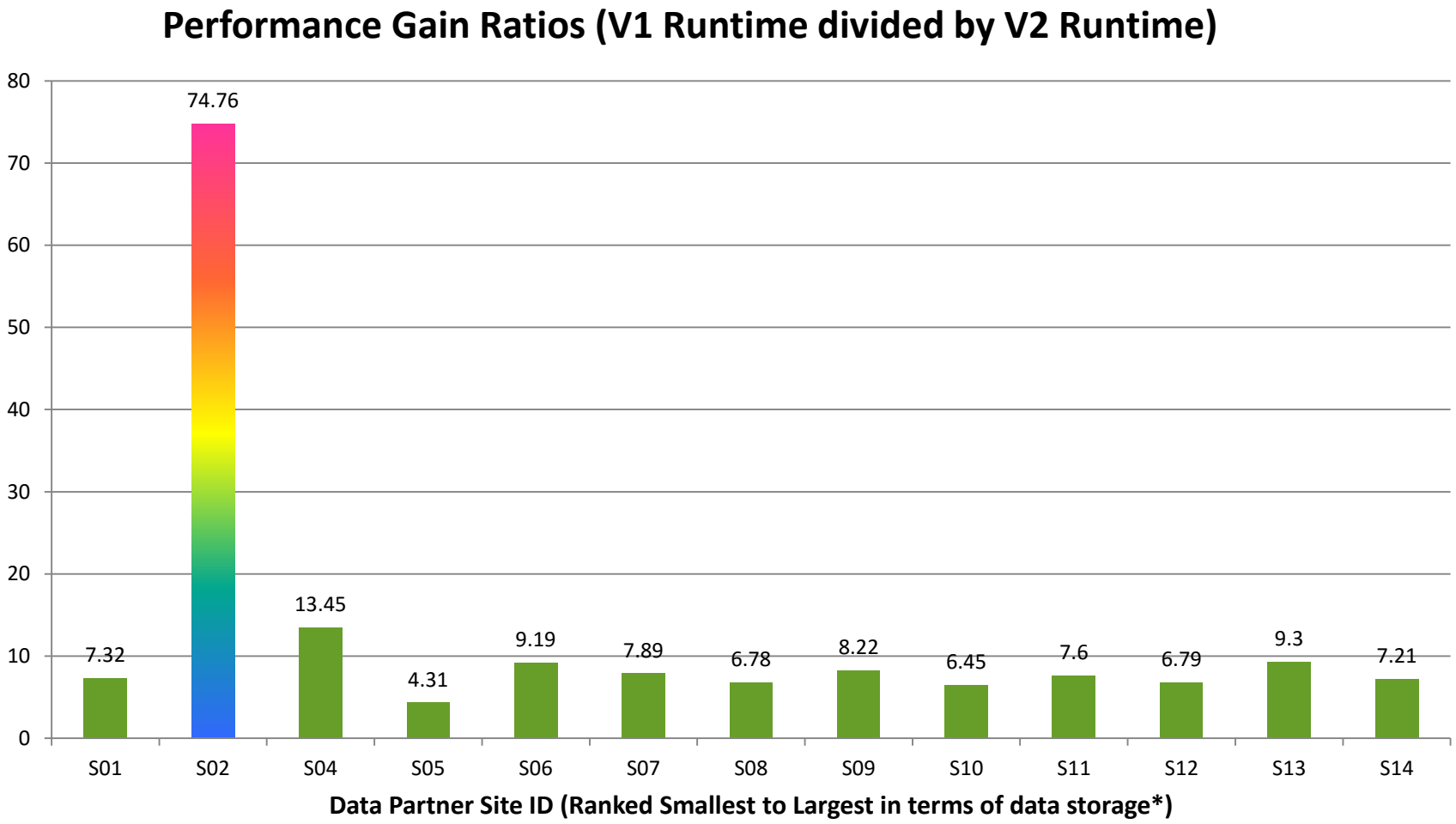
ST Creation - Results



	S01	S02	S03	S04	S05	S06	S07	S08	S09	S10	S11	S12	S13	S14
v1_runtime	3:30:00	29:22:00		16:31:00	3:54:00	26:39:00	13:39:00	8:11:00	52:40:00	105:42:0	108:30:0	122:41:0	123:27:0	90:40:00
v2_runtime	0:28:41	0:23:34	3:05:24	1:13:41	0:54:23	2:53:53	1:43:47	1:12:28	6:24:18	16:22:38	14:17:04	18:04:26	13:16:46	12:34:55

*Footnote: Rankings are based on gigabytes of computer storage. Storage can be impacted by many things such as years of data, type of data, type of enrollment, storage efficiency, # patients, etc.,

ST Creation - Results (Continued)



*Footnote: Rankings are based on gigabytes of computer storage. Storage can be impacted by many things such as years of data, type of data, type of enrollment, storage efficiency, # patients, etc.,

How did we do it?

- We redesigned the code to be “smarter” about
  - Computer system capabilities in general
  - Data volume differences in particular

Core Concepts

Computer System Performance

- A well functioning computer system can process jobs without resource bottlenecks or resource exhaustion

- 90% of the performance problems reported to the SAS Institute are due to improperly tuned or improperly sized systems

http://blogs.sas.com/content/sgf/2015/11/18/tips-to-keep-your-sas-system-humming/
http://support.sas.com/kb/42/197.html

Optimizing the SAS Configuration File

- Option settings recommended for SAS 9.2/9.3
  - MEMSIZE 2G → Limit on virtual memory available for SAS job/step
  - SORTSIZE 1G → Limit on memory for sort-related utility files
  - CPUCOUNT 4 → CPU "cores" available for multi-threaded procedures
  - BUFSIZE 64K → Page size used for SAS files (set when file is created)
    - → " " Utility files (used by multi-threaded procs)
    - → " " Index files
  - BUFNO 10 → Pages of data processed during a single IO operation
    - → " " Utility files
    - → " " Index files
  - UBUFNO 10
  - IBUFNO 10

- Location of Configuration File
  - Windows OS: !SASROOT\sasv9.cfg
  - UNIX OS: !SASROOT/sasv9_local.cfg

http://support.sas.com/kb/46/954.html
http://support.sas.com/resources/papers/Flexibility_by_Design.pdf

Optimizing OS & Hardware Configuration

Operation System (e.g. Windows, UNIX, etc.):

- Tuning guidelines are OS specific

CPU Cores:

- Current generation processing cores

Physical Memory (RAM):

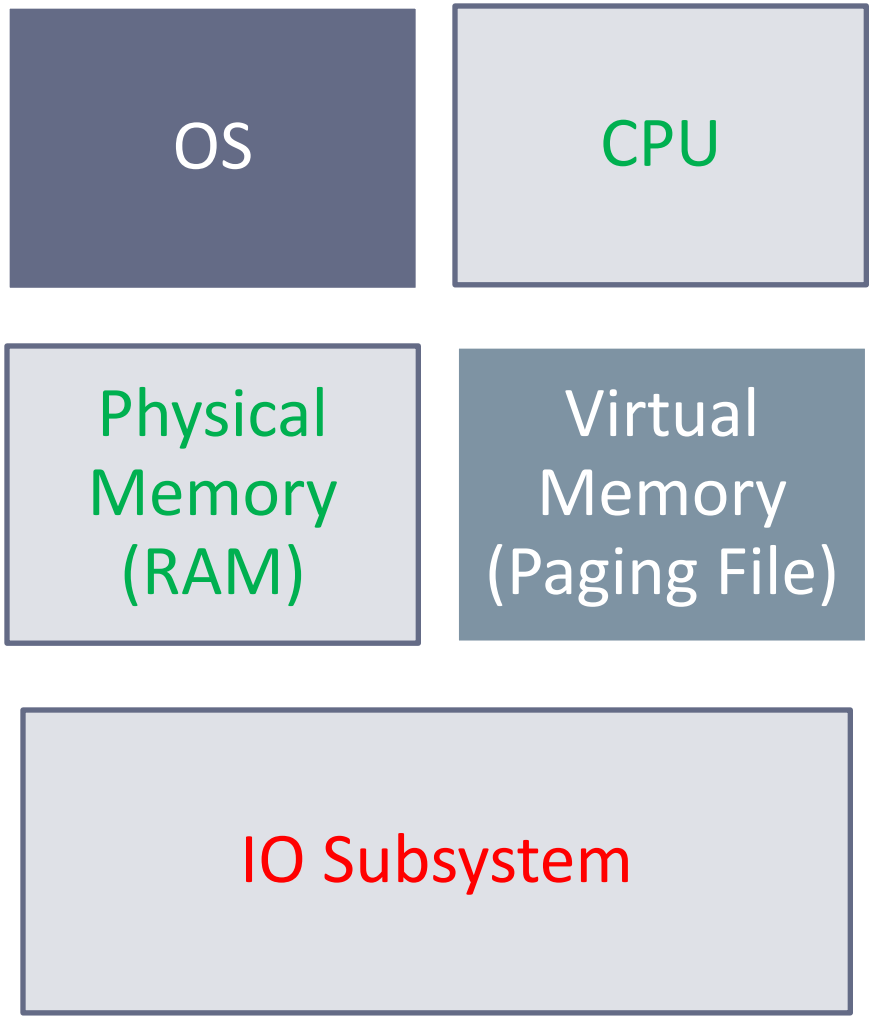
- >= 8GB RAM/***core***

Virtual Memory (Paging File):

- 1.5-2.0 x physical RAM

IO Subsystem:

- Overall throughput: >=100-125 MB/sec/***core***
- SAS WORK & UTILOC: >= 100 MB/sec/***core***
- SAS permanent data: >= 50-75 MB/sec/***core***
- **Use multiple, striped file systems if possible!**



http://blogs.sas.com/content/sgf/2014/10/08/configuring-sas-what-to-know-before-you-install/

Optimal Hardware Configurations Cost $$$

Optimally, a workstation with 4 cores might have:

- >= 32GB RAM (8GB RAM/core x 4 cores)
- ***>= 40-44 striped HDD for 3 file systems***
  - >= 16 HDD striped for WORK (4 HDD/core x 4 core)
  - >= 16 HDD striped for UTILITY (4 HDD/core x 4 core)
  - >= 8-12 HDD striped for permanent SASDATA (2-3 HDD/core x 4 core)

In reality, a workstation with 4 cores might have far less:

- 12 GB RAM (3GB RAM/core)
- 3 HDD non-striped (each HDD has ~25 MB/sec IO, which is ~ 4x slower/core than optimal)

IO Bottlenecks …

- When performance problems occur, insufficient IO throughput is often the diagnosis

- The SAS Institute provides many great resources to help you *tune and right-size* your system for better IO throughput …

- We will show you *coding techniques* that **reduce IO requests**

http://support.sas.com/kb/53/874.html

What is “Good” IO throughput?

- “Good” throughput occurs when the CPU are not kept waiting for IO
- The ratio of Real time to Total CPU time (user + system) will be close to 1
- Use FULLSTIMER option to report on Real time and Total CPU time

```
real time                  4:25.56
user cpu time              5:36.14
system cpu time            25.77 seconds
memory                     3354692.78k
OS Memory                  3622096.00k
Timestamp                  02/05/2016 11:02:30 PM
Step Count                              2142  Switch Count  0
Page Faults                             0
Page Reclaims                           2363082
Page Swaps                              0
Voluntary Context Switches              693505
Involuntary Context Switches            622
Block Input Operations                  1384008
Block Output Operations                 29158608
```

http://support.sas.com/kb/51/660.html

http://support.sas.com/rnd/scalability/papers/solve_perf.pdf

Why does IO throughput vary so much across SAS steps/jobs (on the same system)?

- SAS Code Efficiency – *Efficient vs. Inefficient*
- Dataset Sizes – *Small (< 2 GB) vs. Large*
- Physical Memory (RAM) – *Abundant vs. Limited*
- Multi-Threaded Steps – *Few vs. Many*
- Partitioned Parallel Processing – *Yes vs. No*
- Competing Jobs – *Few vs. Many*

➢ Even with a "light-weight" IO subsystem, it is still possible to squeeze out "good" IO throughput

Benchmark Test Environments

Benchmarking Environment #1

PC:

- 2 HDD; one with SAS installed, one devoted to WORK space
- 4Gbytes RAM
- 2-core CPU
- Windows 7

Benchmarking Environment #2

PC:

- 3 HDD; one for SAS, one for WORK, one for SASDATA
- 12Gbytes RAM
- 4-core CPU
- Windows 7

Individual Techniques

Individual Techniques

- Ordered here by skill/effort required
  1) Limit variables/rows to a minimum
  2) Reduce variable lengths to a minimum
  3) List GROUP BY vars from most # of distinct values to least
  4) When a PROC/DATA step reads *and* writes a "large" amount of data, turn that step into a view
  5) Aggressively delete temporary files
  6) Use in-memory table lookup techniques (difficulty varies)
  7) Replace long values with short surrogate key values
  8) Use horizontal table partitioning for scalability

Limit Variables/Rows to a Minimum

- Read and write **only:**
  - The variables you need
  - The rows you need

➢This reduces the number of IO requests and saves storage space

*Read and Write only the **Variables** you Need*

- Judiciously use KEEP and DROP statements throughout your DATA step processing

```
Data Many_Variables2;
     Set Many_Variables1;
     ... Processing ...
Run;
```

Gets replaced with…

```
Data Many_Variables2(keep= x1 x2...xn);
     Set Many_Variables1(drop= var1 var2...varn);
     ... Processing ...
Run;
```

*Read and Write only the **Rows** you Need*

- Judiciously use IF and WHERE expressions to include only those observations absolutely required

```
Data Some_Rows;
      Set LargeDataset;
      WHERE age between 0 and 18;
      ... Processing ...
Run;
```

Or…

```
Data Some_Rows;
      Set Many_Variables1;
      ... Processing ...
      IF NewVar in ('AB', 'CD', 'ZX');
Run;
```

LENGTHs of Variables

- SAS defaults the number of bytes in storage to contain a value
- Fewer bytes result in faster performance
- Separate default lengths for character vs. numerics
  - Numerics: Always defaults to 8 bytes
  - Character: Defaults to *1st* knowledge of the length of the values to be stored in a DATA step

```
Data KnowLength;
    set Whatever;
    if Y1 < 70 then Description = “123456789”;
    else if 71 <= Y1 <= 151 then Description = “123456”;
    else Description = “123”;
Run;
```

What is the length for variable Description?

9 bytes

LENGTHs of Variables (Continued)

Storage lengths for numerics in UNIX and Windows

Length in Bytes	Largest Integer Represented Exactly
3	8,192
4	2,097,152
5	536,870,912
6	137,438,953,472
7	35,184,372,088,832
8	9,007,199,254,740,992

- Dates: An integer representing the # of days since 1/1/1960
  - Today's date (March 23, 2016) = 20,536
  - So we need only 4 bytes instead of 8 (50% reduction)
- We won't need to go to 5 bytes until October 24, 7701

LENGTHs of Variables (Continued)

Comparisons*

- One dataset with 2 date variables set to 8 bytes each
- One dataset with 2 date variables set to 4 bytes each

Module	Time (8 bytes)	Time (4 bytes)	% Difference
Proc Freq (1 date variable)	0:09:55	0:08:25	15.2%
SQL Summarization (1 date variable)	0:26:33	0:24:31	7.7%

* Benchmarking environment #1 - using encounters dataset (680 million rows, 60GB)

- Syntax: **LENGTH VariableName length;**

e.g. LENGTH Boolean 3 LastName $25
BirthDate AdmitDate DischargeDate 4;

SQL GROUP BY - Variable Order

- Order GROUP BY variables (from left to right) in the from highest to the lowest number of distinct values

- Example – Group procedure events by 6 variables ordered two ways (low to high vs. high low):
  - Low-High: *EncType, PX_CodeType, PX, Provider, PatId, EncounterID*
  - High-Low: *EncounterID, PatId, Provider, PX, PX_CodeType, EncType*

SQL GROUP BY - Variable Order (Continued)

Comparison*

Method	Real Time	Total CPU Time
Group by: LOW-HIGH	66:57:01	35:34:00
Group by: HIGH-LOW	62:45.49	27:07:12
% Difference	**6.7%**	**29.6%**

* Benchmarking environment #2 - using procedures dataset (1.2 billon rows, 100GB)

- Multi-threaded procedures may hide inefficient coding practices
- "Very Large" datasets (e.g. >= 100GB) are difficult to cache effectively which often leads to "Poor" IO throughput

In-Memory Table Lookups

- Use Formats/Hash Tables to help
  - Reduce **IO Requests** (by avoiding data sorts)
  - Reduce in memory calculations (by using fewer statements)
  - Improve **Modularization** (by removing data from code)

```
/* "Get your data out of your code!" (… for improved modularization …) */
Data OutputDataSet;
        Set InputDataSet;
        If code="123" then Result="This";
        Else if code="369" then Result="Is";
        Else if code="159" then Result="Very";
        Else if code="753" then Result="Tedious";
        Else if code="456" then Result="Agree?";
        …
Run;
```

Gets replaced with…

In-Memory Table Lookups (Continued)

… with a *much better* technique

```
/* Example of Creating a SAS Format */
Proc format;
      Value $CodeToWord
            ”123”=”This”
            ”369”=”Is”
            ”159”=”Very”
            ”753”=”Tedious”
            “456”=“Agree?”
            …                    ;
Run;

/* Example of Using a SAS Format for In-Memory Table Lookups */
Data OutputDataSet;
      Set InputDataSet;
      Result = put(code, $CodeToWord.);
Run;
```

In-Memory Table Lookups (Continued)

Comparisons*

- Hardcoded IF-THEN-ELSE statements
- PROC FORMAT and PUT() function statement

Time (IF-THEN-ELSE)	Time (PROC FORMAT and PUT())	% Difference
1:03:09	0:30:27	51.8%

* Benchmarking environment #1 – using encounters dataset (680 million rows, 60GB)

Surrogate Keys for Long Values

- Long ID variables (10+ characters) can be replaced with numeric keys of many fewer bytes
- Long descriptions can be replaced with numeric keys of many fewer bytes

Examples

- A patient ID of 30 characters can be replaced with a numeric surrogate of only 4-5 bytes
  - With 300 million people in US, only 5 bytes are needed
  - You are likely to have fewer persons to manage!

- Long descriptions can be replaced with numeric keys of many fewer bytes
  - Drug description of 35 characters can be replaced with 3-4 bytes

Surrogate Keys for Long Values (Continued)

Managing surrogate keys requires the following overhead:

1. Discover the number of unique values
2. Determine # of bytes for storing each variable
3. Map the unique values to surrogate keys (1+)
4. Apply the surrogate keys to source rows
5. Manipulate (e.g., filtering, summarization) datasets containing surrogate keys
6. Reapply original values

Surrogate Keys for Long Values (Continued)

Using a table of 102mil outpatient drug claims

1. Discover the number of unique values = **43,331**
2. Determine # of bytes for storing each variable = **4 bytes**
3. Map the unique values to surrogate keys (1+)
4. Apply the surrogate keys to source rows
5. Manipulate (e.g., filtering, summarization) datasets containing surrogate keys
6. Reapply original values

Depending on your application, the overhead may be worth the effort

Surrogate Keys for (Enrollment) Status Flags

2004	2005	2006	2007	2008	2009
Y	Y	N	N	Y	Y

2004	2005	2006	2007	2008	2009
1	1	0	0	1	1

110011	Binary number

51	Decimal number

Only 3 bytes needed (Windows and UNIX)

- **For 15 years, maximum decimal value = 32,767**
- **Needs only 4 bytes, savings of 9 bytes, = 60%**

Surrogate Keys for Status Flags

- Compare use of two in-memory hash table lookup approaches, linking enrollment status flags for 5 million patients onto 320 million event records:*
  - "Separate status flags approach" (for speed)
  - "Binary decimal approach" (for packing more information into less memory)

Module	Time (6 separate flags)	Time (Disassemble Decimal Value)	% Diff
Hash Lookup: Status flags and event records	0:07:22	0:07:46	5.3%

* Benchmarking environment #1

Horizontal Table Partitioning

PatID	VisitDate	DX_Code
2213	2/12/2015	250.0
3338	3/18/2015	154
4152	4/16/2015	154.55
5344	5/26/2015	216.6
6235	6/7/2015	300.5
7218	7/2/2015	395.2
8089	8/11/2015	569.6
9292	9/17/2015	655.8
10115	10/18/2015	036.43
11264	11/14/2015	V90.32
12158	12/27/2014	E013.1
13329	1/4/2015	506.1
"Billions" of Rows		
14177	2/9/2015	853.05
15198	3/19/2015	V65
16126	4/18/2015	747.5
17266	5/8/2015	879.5
18317	6/14/2015	844
19179	7/6/2015	V90.32
20263	8/26/2015	923
21322	9/17/2015	506.1
22072	10/22/2015	671.13
23310	11/25/2015	967.9
24182	12/9/2014	378.2
25338	1/4/2015	719.82

PatID	VisitDate	DX_Code
2213	2/12/2015	250.0
3338	3/18/2015	154
4152	4/16/2015	154.55
5344	5/26/2015	216.6
"Millions" of Rows		
14177	2/9/2015	853.05

PatID	VisitDate	DX_Code
6235	6/7/2015	300.5
7218	7/2/2015	395.2
8089	8/11/2015	569.6
9292	9/17/2015	655.8
"Millions" of Rows		
15198	3/19/2015	V65

PatID	VisitDate	DX_Code
10115	10/18/2015	036.43
11264	11/14/2015	V90.32
12158	12/27/2014	E013.1
13329	1/4/2015	506.1
"Millions" of Rows		
16126	4/18/2015	747.5

...and many more

Horizontal Table Partitioning

Partitioning can be by:

- Random Selection
- Strata (e.g., Year, Region, Department, Category)
- Entity (e.g., Patient, Customer)

➢ *Analytic requirements drive partitioning strategy*

➢ *Data volumes (ideally) drive how many partitions to create*

For the summary table code, we partitioned by:

- Year → prevalent summary tables
- Patients → incident summary tables

HT Partitioning - Processing Methods

- **Method 1 – Sequential Processing**
  - Process partitions one at time through a set of steps
  - Improves performance through efficient resource use

- **Method 2 – Parallel Processing**
  - Process multiple partitions concurrently through a set of steps
  - Improves performance through efficient use of *more* resources (e.g. more CPU, RAM, and IO resources are concurrently used in order to reduce real times)
  - More flexible of the two methods (but harder to do)

Horizontal Table Partitioning Results

Table of 201 million rows
Summarizations by Year*

Module	Process	Average Time
Summarize Data (Full File)	PROC SQL summarization	0:14:04
Summarize Data (Yearly Partitions)	• Data step to partition (6 years) • PROC SQL summarization each year • Data step to concatenate results	0:07:58
	Benefit	**43.8%**

* Benchmarking environment #1

HT Partitioning – Why it Works

- In general, splitting tables **horizontally** and **looping** through the **partitions** allows us to
  - Regulate the maximum amount of resources consumed at every moment (i.e. we load balance within the job)
  - Tilt resource usage favorably (i.e. we effectively exploit the available RAM and CPUs and we minimize IO requests)
  - Improve the scalability of our solutions

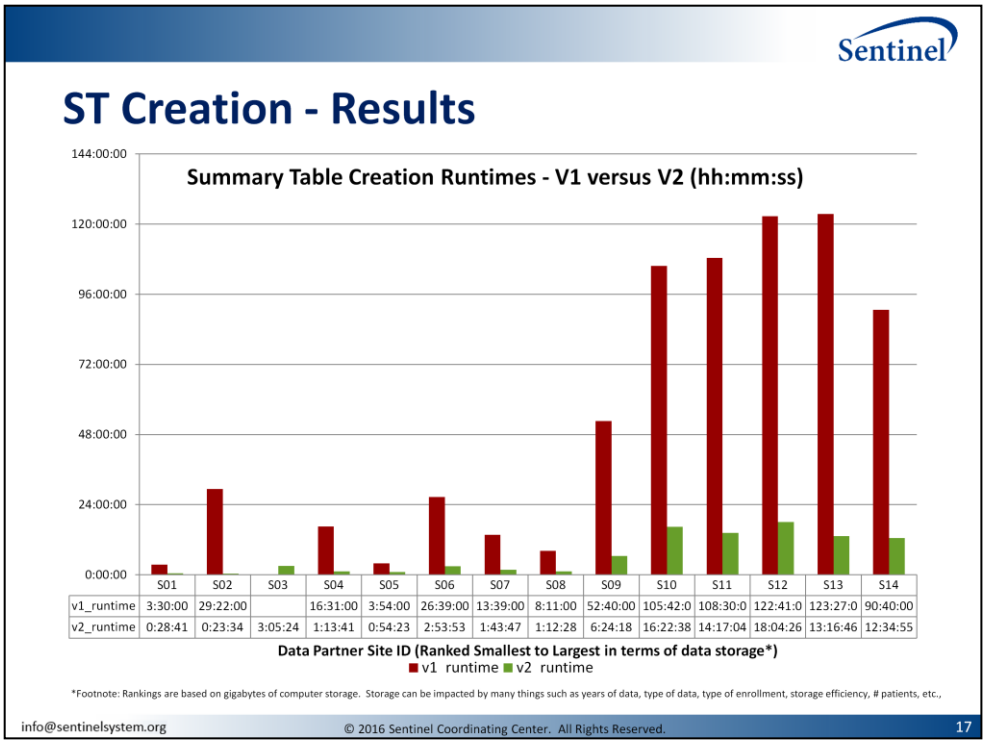
HT Partitioning – Why it Works (Continued)

- More specifically, we greatly improve the odds that
  - System resources (e.g. RAM & disk space) are never exhausted
  - Files are *effectively cached* in RAM (reduces IO, exploits RAM & CPU)
  - PROCs that sort/link/aggregate data are able to load *entire* files into RAM (reduces IO, exploits RAM & CPU)
  - Multi-threaded steps are able to fetch data pages directly from the cache (reduces IO, *fully exploits RAM & CPU*)

Acknowledgements

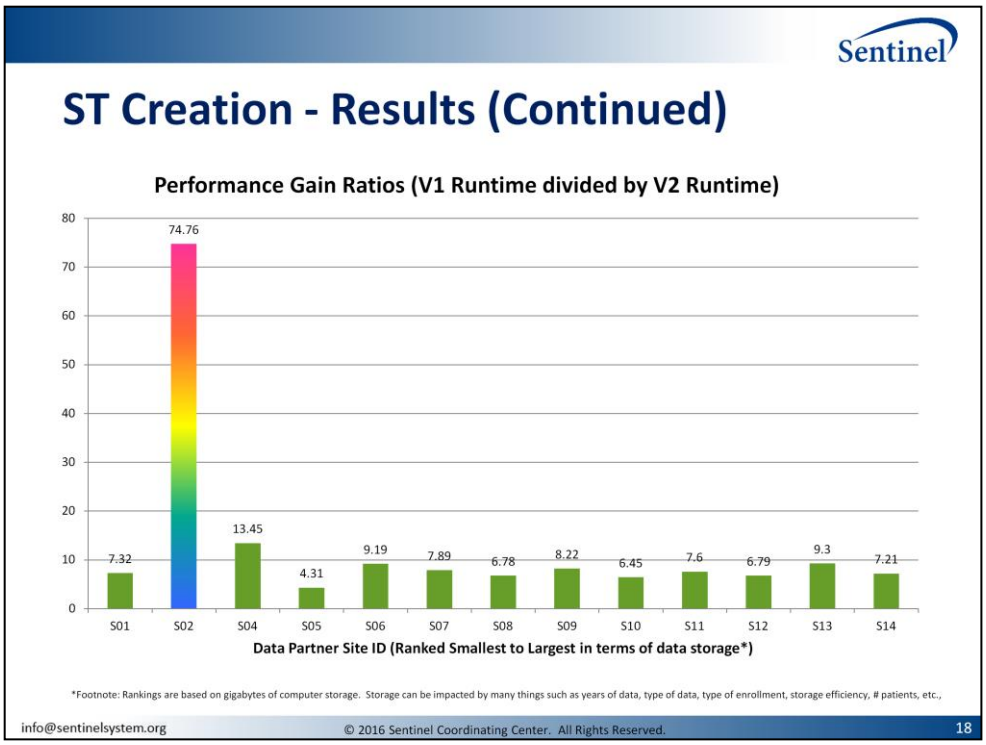
Contributing Partners

- Our early explorations into alternative approaches greatly benefited from the expertise and helpfulness of our Sentinel partners
  - Ron Johnson (Group Health)
  - Jack Hamilton (Kaiser P. of Northern California)
  - Steve Roloff, Eva Ng (Optum)
- These individuals (and organizations) provided in-depth feedback, troubleshooting, coding support, and benchmarking results, for which we are extremely grateful



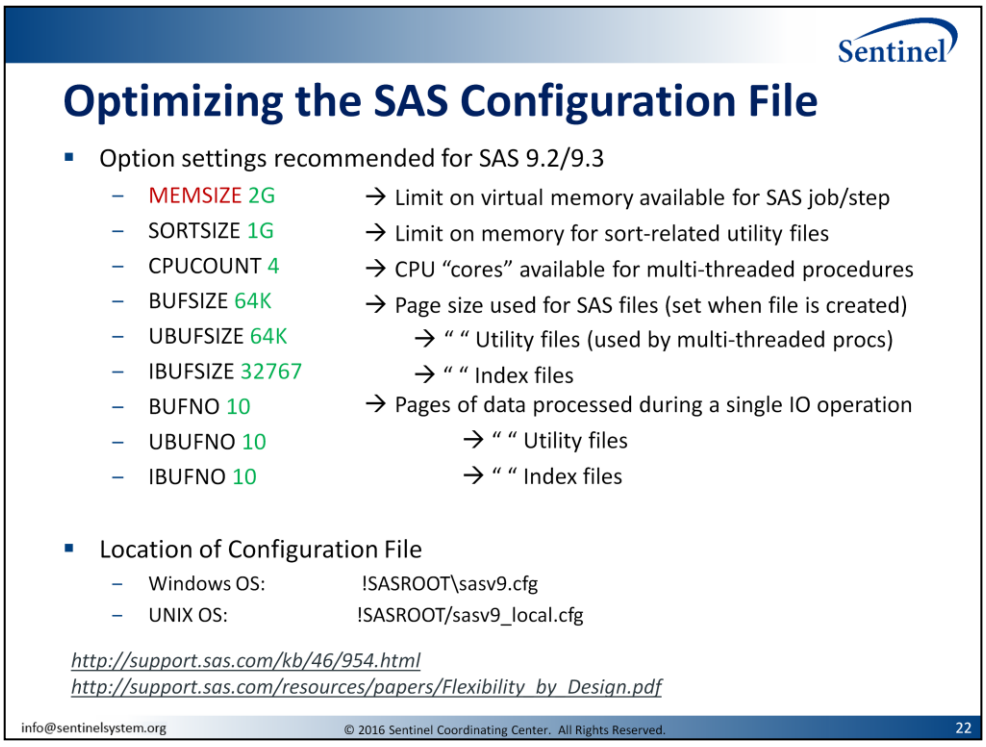
Talking Points

I love this chart because it shows that every site, even the large sites, will be able to produce results in under 24 hours. Most sites will be able submit queries in the morning and check results in the afternoon, while the largest sites can submit queries in the morning and check results the next morning – this is fantastic!



Talking Points

I love this chart because it shows (if you ignore the attention seeking outlier) remarkable consistency in performance improvement gains across all sites regardless of their size. This speaks volumes to the scalability of the approach.



http://support.sas.com/kb/46/954.html

http://support.sas.com/resources/papers/Flexibility_by_Design.pdf

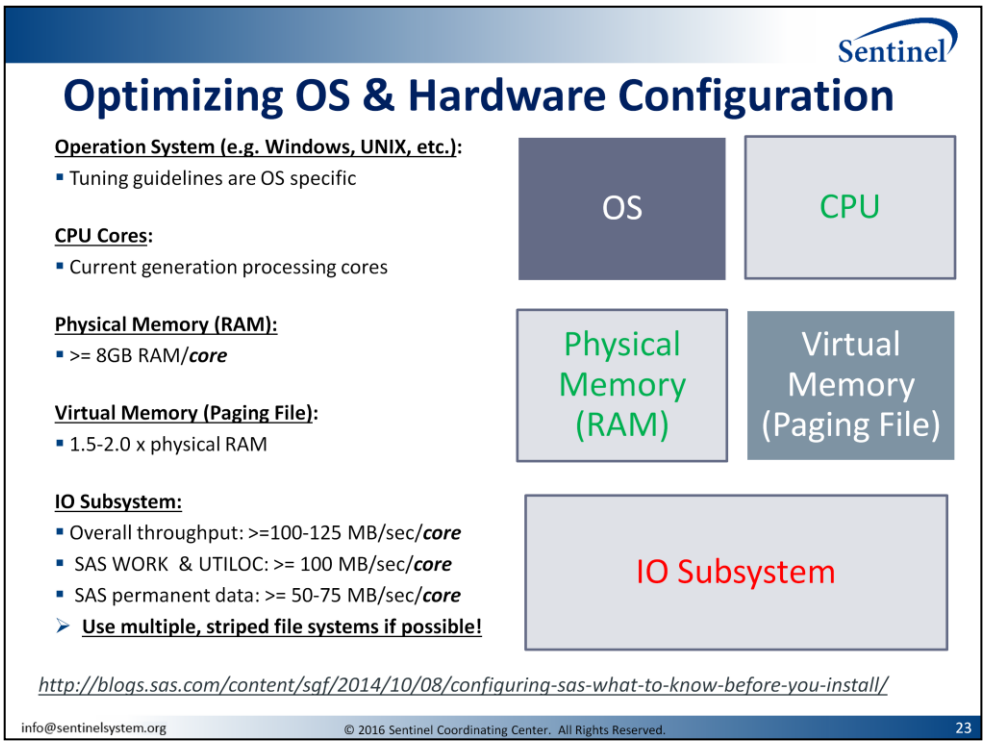
Talking Points

• MEMSIZE must be set when the SAS session is invoked, the other options may be set in the SAS code using an options statement

•Out of the box configuration SAS option settings are conservative (often too conservative) which led to the above recommendations from SAS Institute, but they should still be benchmarked.

• Starting with version 9.4, the default SAS option settings are less conservative, but they should still be evaluated/benchmarked.

• Conservative option settings may sometimes hamper job performance but the upside is that jobs are less likely to exhaust resources.



Talking Points

• Optimal configurations use three types of file systems – file system(s) dedicated to SAS WORK, file system(s) dedicated to SAS Utility files created by multi-threaded procedures, and file system(s) dedicated to permanent SASDATA

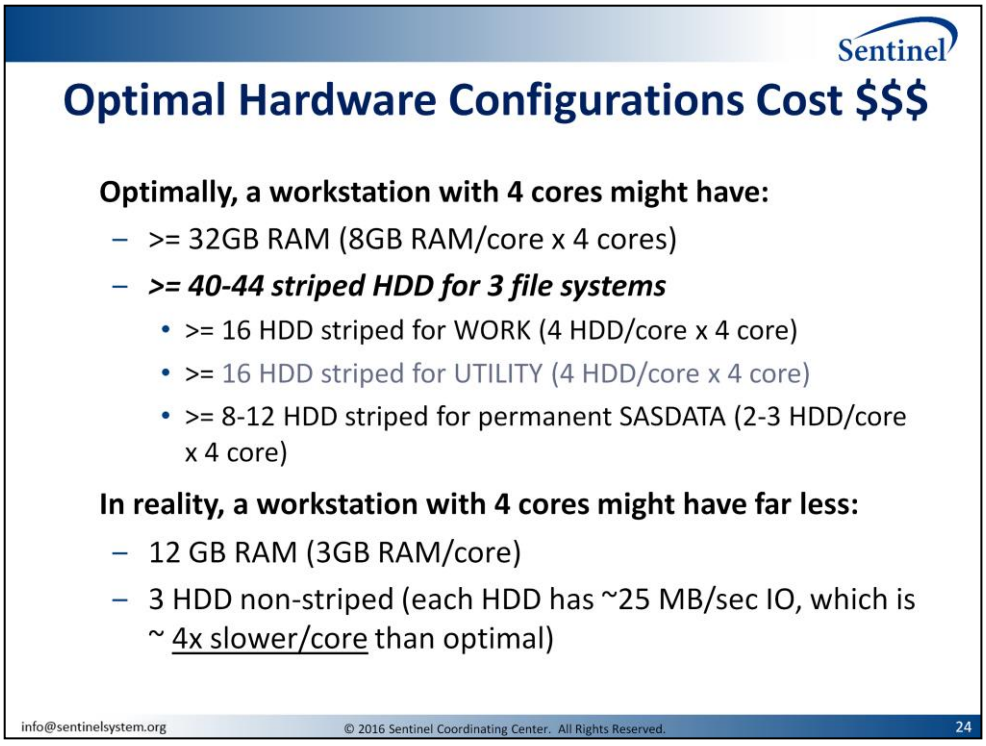
• A file system with 4 striped HDD yields ~ 100 MB/sec IO throughput

Definitions

• A "**core**" is an independent processing unit that execute instructions. (https://en.wikipedia.org/wiki/Multi-core_processor).

• **Data striping** is a method of improving IO by spreading data segments across multiple devices that can be accessed concurrently (https://en.wikipedia.org/wiki/Data_striping)

http://blogs.sas.com/content/sgf/2014/10/08/configuring-sas-what-to-know-before-you-install/

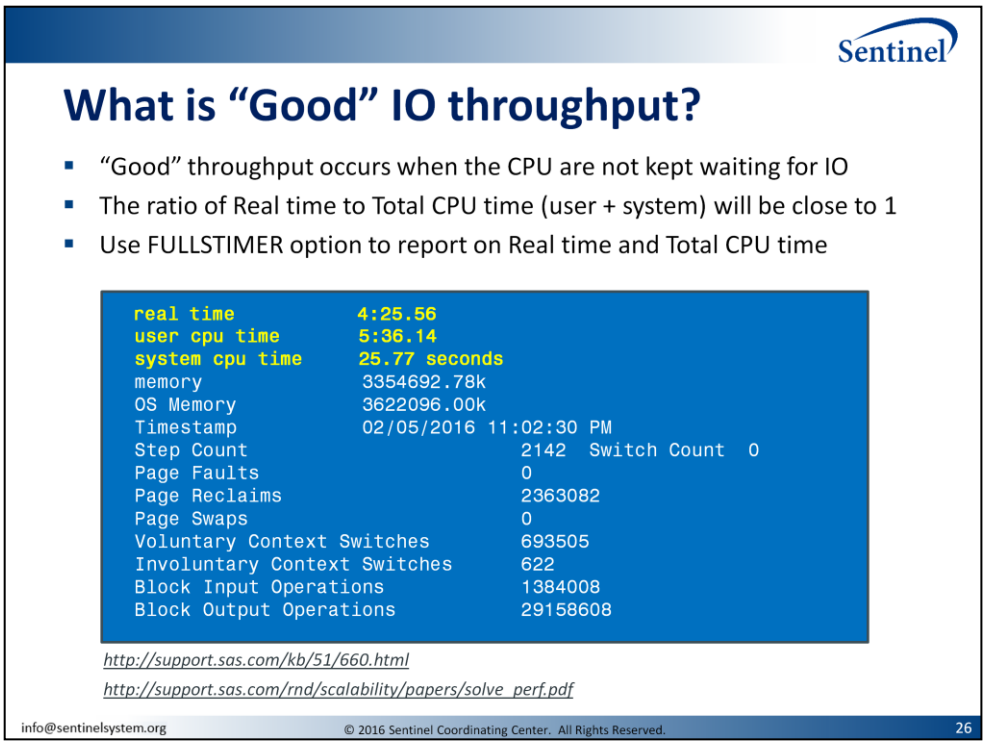


Talking Points

• Fewer devices are needed if SSD are used instead of HDD

• Optimally, 3 types of file systems are recommended, but 2 types of file systems (one for WORK, another for permanent SASDATA) may be adequate in which case the requirements would drop down to 24-28 striped HDD

• A striped SASDATA file system is often set up to trade some performance for a chance of recovery from a device failure

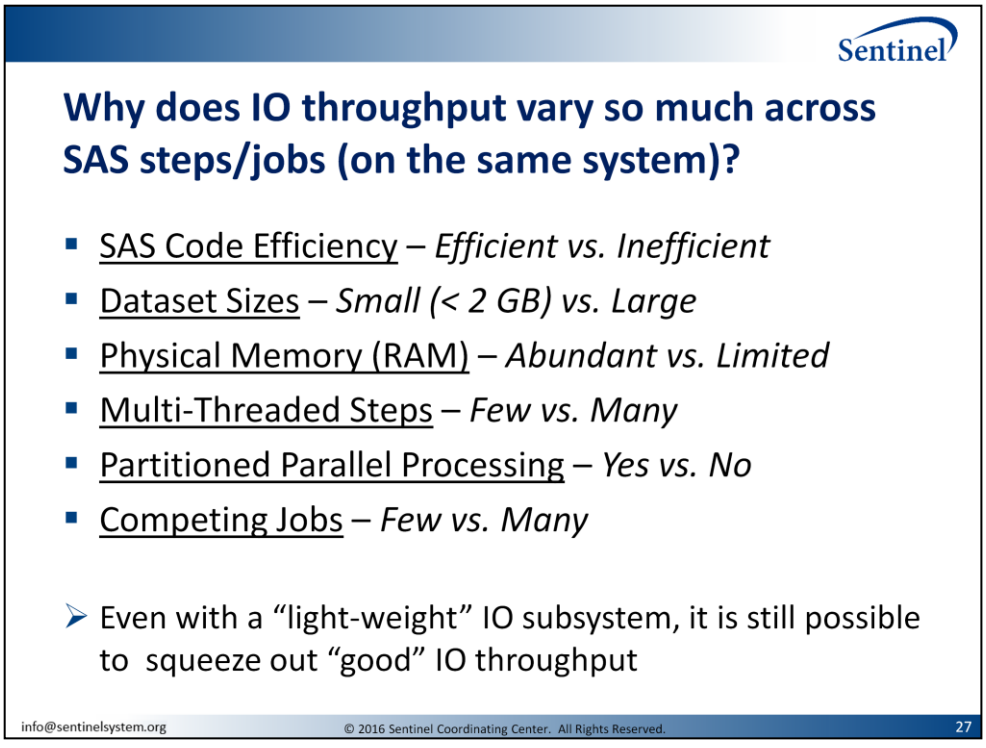


Talking Points

- A ratio significantly less than 1 indicates "*Really Good* " IO throughput.
- For example, the FULLSTIMER screen shot shows a SAS step where Real time is *less* than Total CPU time. This happened because the step used multiple CPU threads to read multiple chunks of data in parallel, which *increased* Total CPU time but *reduced* Real time.

Definitions

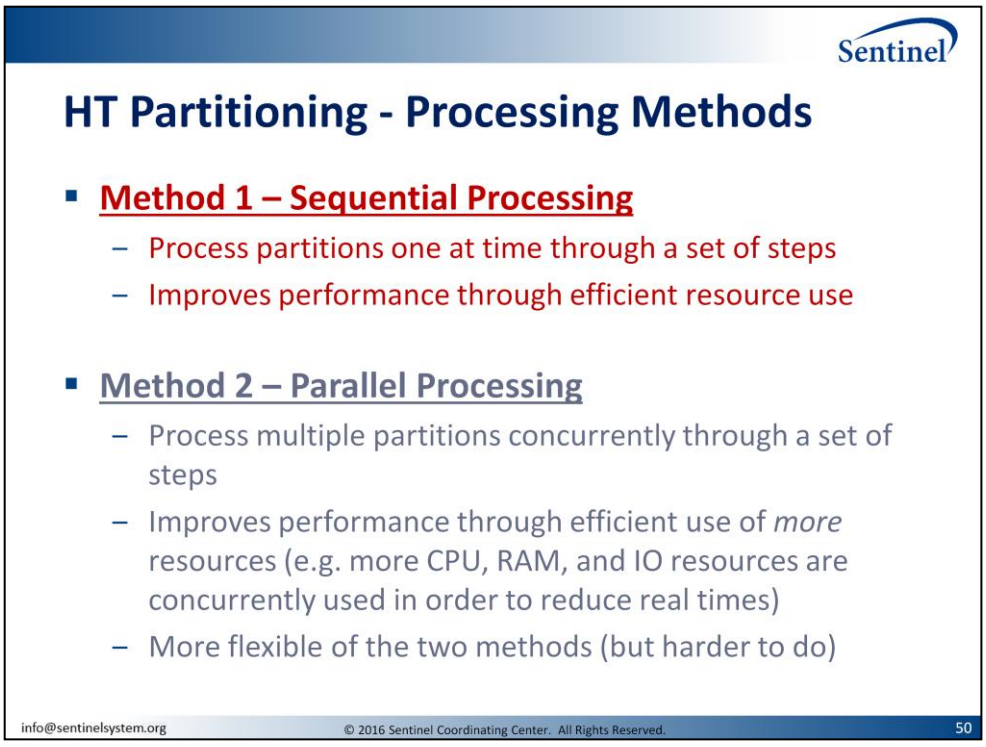
- **Page Faults**: Number of virtual memory page faults that occurred for this SAS job/step which required an IO operation to retrieve pages from disk
- **Page Reclaims**: Number of virtual memory pages retrieved for this SAS job/step from the page list awaiting reallocation (i.e. these pages are still in the memory cache and do not require and IO operation)
- **Page Swaps:** The number of times a process was swapped out of main memory. Consistently high values are evidence that memory resources should be looked into.
- **Involuntary Context Switches:** The number of times a process releases its time slice involuntarily. A consistently high values are evidence that the CPU should be looked into.
- **Block Input Operations:** The number of pages read for this SAS job/step that required an IO operation
- **Block Output Operations:** The number of pages written for this SAS job/step that required an IO operation



Talking Points

• Multi-threaded steps: Many SAS procedures can employ multiple reader threads to process data in parallel (e.g. SQL, SORT, MEANS/SUMMARY, etc.)

• Partitioned Parallel Processing: Multiple asynchronous SAS sessions are spawned, each processing separate partitions of data in parallel. This can be accomplished through the MP Connect SAS Module, or by using the SYSTASK and WAITFOR commands in Base SAS



Method 2 - Parallel Processing: Multiple asynchronous SAS sessions are spawned, with each SAS session processing separate partitions of data in parallel. This can be accomplished either through the MP Connect SAS Module, or by using the SYSTASK and WAITFOR commands in Base SAS.