Paper 2500 - 2015

Integrating SAS® and the R Language with Microsoft SharePoint

Prasoon Sangwan, Piyush Singh, Shiv Govind
TATA Consultancy Services Ltd.

ABSTRACT

Microsoft SharePoint has been adopted by a number of companies today as their content management tool because of its ability to create and manage documents, records, and web content. It is described as an enterprise collaboration platform with a variety of capabilities, and thus it stands to reason that this platform should also be used to surface content from analytical applications such as SAS® and the R language. SAS provides various methods for surfacing SAS content through SharePoint.

This paper describes one such methodology that is both simple and elegant, requiring only SAS Foundation. It also explains how SAS and R can be used together to form a robust solution for delivering analytical results. The paper outlines the approach for integrating both languages into a single security model that uses Microsoft Active Directory as the primary authentication mechanism for SharePoint. It also describes how to extend the authorization to SAS running on a Linux server where LDAP is used. Users of this system are blissfully ignorant of the back-end technology components, as we offer up a seamless interface where they simply authenticate to the SharePoint site and the rest is, as they say, magic.

INTRODUCTION

It is becoming quite popular to create the web service reports and share them with the group of users. SharePoint is a very popular content management tool where users store and share their reports with their team in a secured way, by granting the access to SharePoint page.

In this paper we have used SAS Foundation for all SAS computation, R-Shiny for web service part and SharePoint for display/ store the reports. Though SAS Foundation has been powerful statistical tool, but creation of web services through SAS Integration Technology® is still bit complex (considering its multitier architecture concept and technology). We used R and Shiny to help basic users create web services. This solution is built for users who are using SAS Foundation and don't have enough exposure of SAS Integration Technology®. R (shiny) is open source, easily available and simplifies the creation of interactive web applications. Use of SAS Integration Technology® provides more features to users, but at a higher cost and user's learning.

ARCHITECTURE

The Integrated system architecture is designed in such a way that the complete setup gives a single point of entry to all the users. There are three basic building blocks to achieve the end to end result.

- SAS Foundation
- R (Shiny)
- Microsoft SharePoint

Security of integrated system is maintained through following

- Lightweight Directory Access Protocol (LDAP)
- Microsoft Active Directory (MS AD)

As explained in figure 1, SharePoint is used to display the request page of the web services created in Shiny/R. When a user accesses the web browser, the user id is validated against the set of permissible

users via AD security. Once that validation passes, the user is allowed to place the request. It helps in providing additional security by encapsulating the server details of R web service.

When the user sends a request, R web service captures the respective selected parameters and passes it to the SAS program for execution. This process of messaging to SAS is secured through OS security (LDAP). The SAS program executes as per the user's input parameters received through R web service. After successful execution of SAS, result is shared back to SharePoint for display in the specified layout.

.

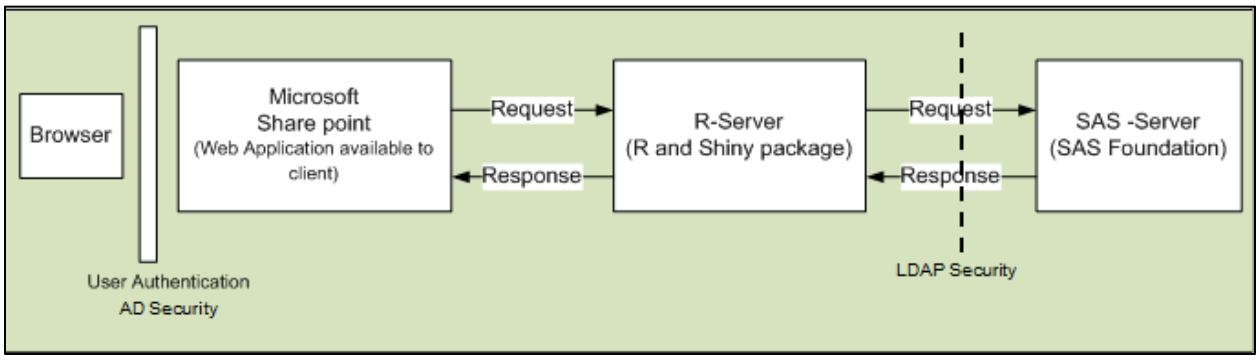


Figure 1. Architecture Diagram

APPROACH

There are three approaches to achieve the communication between SAS and R:

- SAS and R installed on different servers.
- SAS and R installed on different servers with a shared file system.
- SAS and R installed on the same server

SAS AND R INSTALLED ON DIFFERENT SERVERS

In the first approach SAS Foundation and R are installed on different servers. All input and output communications between SAS and R servers occur through Linux Shell Script. R web service captures input parameters in a file from the webpage at Share Point and then the file is passed on to SAS code residing on SAS server using SSH. SSH is the mechanism to send the input to code from R web service and then bring the SAS result back to R server to display (figure2).

It's always good to capture all input parameters required by SAS code in one file and then SSH it to SAS server. If number of SSH increases then total processing time to display the result increases. This is because of the additional message and response time; each time SSH is done between the servers. Usually, the result is displayed immediately after the request from user (interactive mode), but application can also be modified to facilitate the scheduling of the SAS jobs in batch mode. In case of batch jobs, there are various methods of notification which can be opted to send the notification once job is completed. The presentation of result on the web page can be controlled via R codes. Once SAS program is executed, the expected result generated on SAS server (to be displayed at SharePoint) is pulled on to R server.

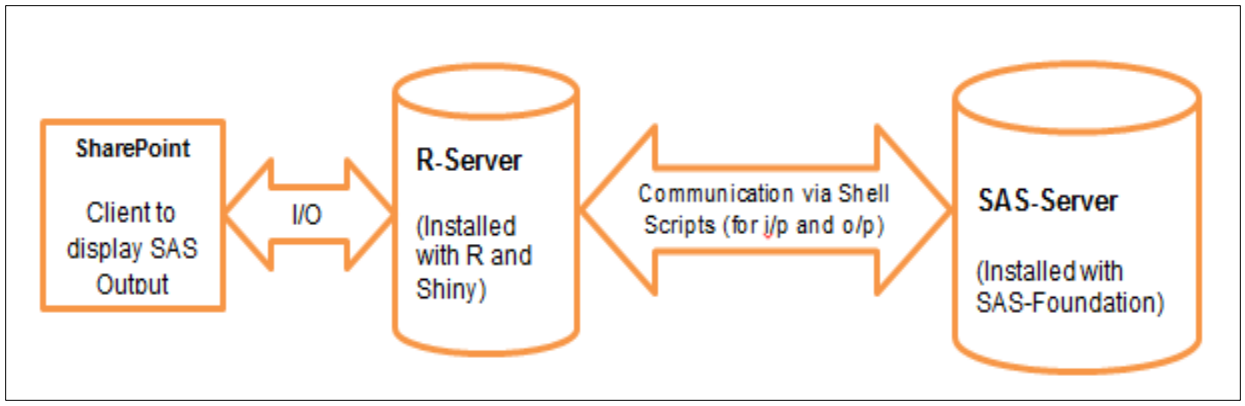


Figure 2. Architecture Diagram of R and SAS installed on different servers

The diagram below (figure 3) demonstrates the information flow between the various components of the application for the cited example; SERVER.R and UI.R as R scripts, which create the web service with the help of Shiny. R script invokes the Linux script. This Linux script executes the SAS program and provides the result back. Based on requirement, Linux scripts can be modified much more to fulfil the application specific requirement.

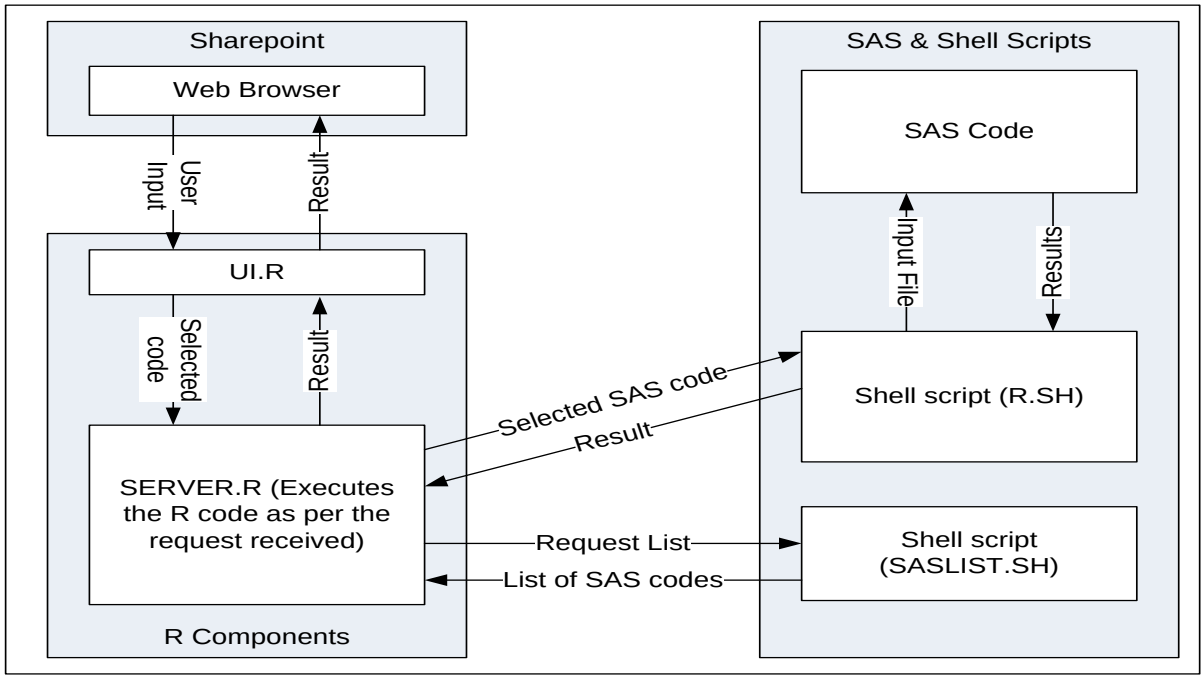


Figure 3. Flow Diagram between SharePoint, SAS, R and Shell Scripts

SAS AND R INSTALLED ON DIFFERENT SERVERS WITH A SHARED FILE SYSTEM

As given in below diagram (figure 4), R and SAS servers have shared file system. Use of shared file system, eliminates the process of SSH and sends the files between the servers. Once the input and output files are written as per given parameters, the files are available automatically for use through shared file system.

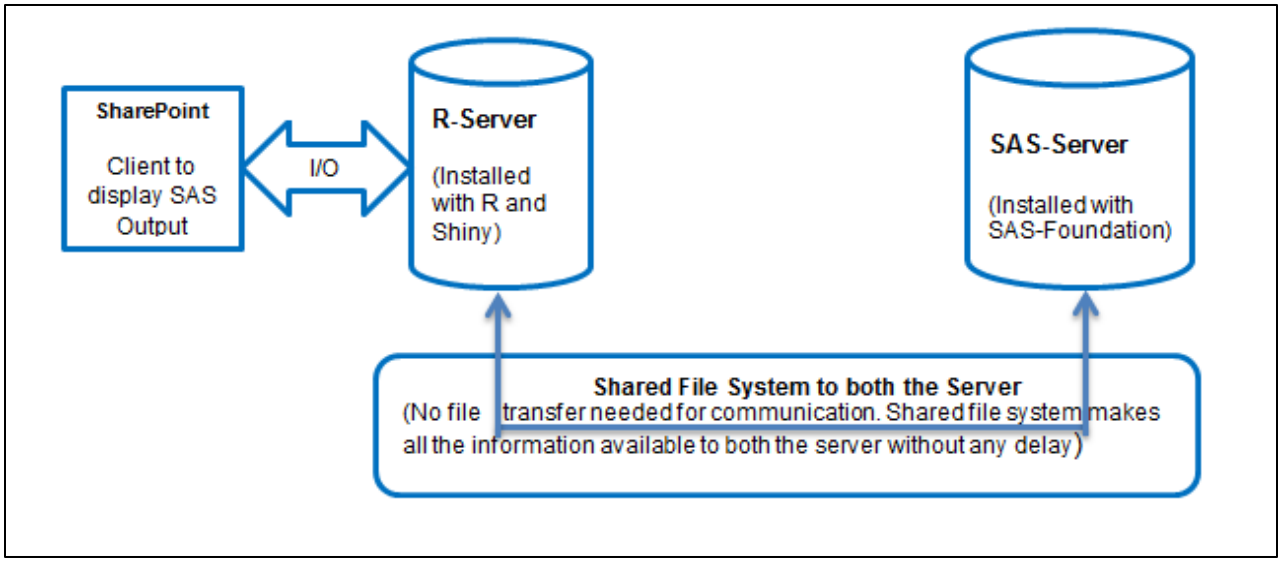


Figure 4. Flow Diagram for Shared File System for SAS and R Servers

SAS AND R INSTALLED ON THE SAME SERVER

Below figure 5, explains the architecture of R and SAS installed on same server. Since, both are installed on same server, it eliminates the process of SSH and sending the information between the servers. The file generated by one process is readily available to the other process without any intervention of any intermediate process.

The overall process explains that the process of communication happens with the help of writing the files and making it available for other processes.

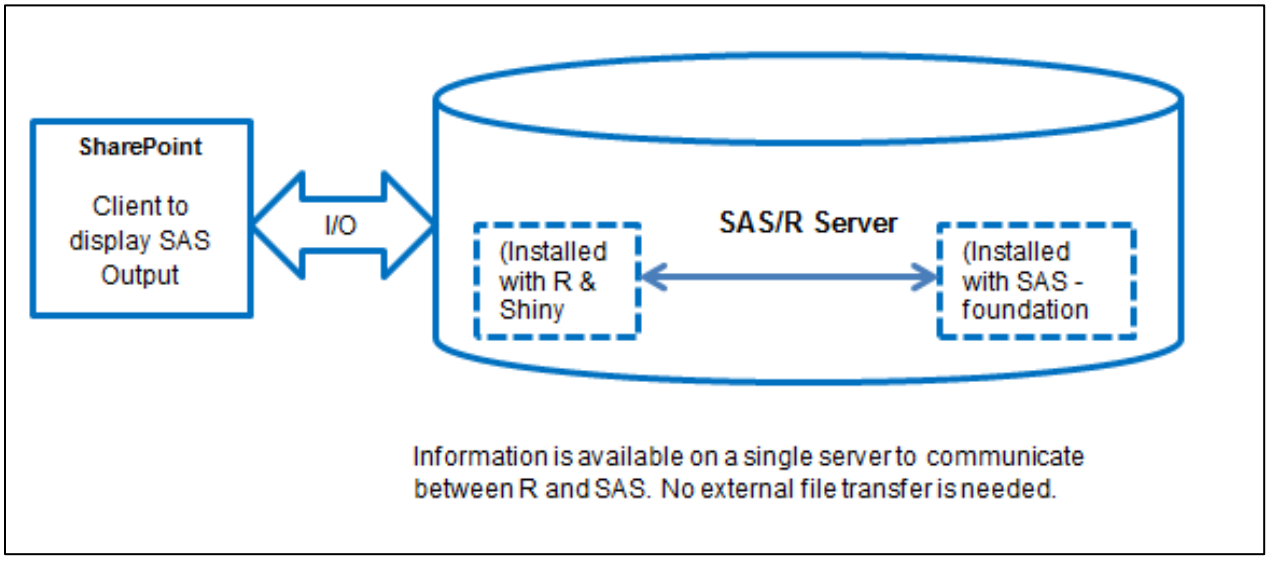


Figure 5. Flow Diagram for SAS and R installed on the same server

PROCESS FLOW

There are five basic steps to create end to end solution. But it totally depends on requirement and coding skill (scripting and R/SAS coding) to create dynamic and menu driven reports.

- Write SAS and R program
- Use Shiny to create web service and execute SAS programs at backend.
- Execute R and display the result in SharePoint through R web Service.
- Use Active directory to authenticate the user to access the web service.
- Use LDAP to secure the SAS program execution on SAS server.

This paper does not use any SAS BI technology to create, web portal. Only SAS Foundation is used to run SAS programs. Writing SAS and R programs is out of scope of this paper. Developers write their own SAS/ R code as per the business requirement. It is assumed that users have basic understanding of SAS, R and Shiny to achieve this functionality.

1. R SCRIPT – USER DISPLAY

ui.R -

This is the user interface script which is used to control the appearance and layout of the web page. It facilitates the page created, to automatically adjust to the dimensions of the user's browser window.

In the example below, UI.R gives an option to the user to select the SAS code to be executed through the drop down. And then displays the output; back on the side panel as per the response received from SERVER.R. The output can be in the form of table, charts, figures, graphs etc. depending upon the response results returned by the SAS code.

```
# ui.R
library(shiny)
shinyUI(pageWithSidebar(
  headerPanel("SAS Code Execution in R Web Service."),
  sidebarPanel( uiOutput("SAS_Pgm") ),
  mainPanel(
    list(tags$head(tags$style("body {background-color: #ADD8E6; }"))),
    h3("Report from SAS Server !", style = "color:blue"),
    h4("Below Report Is From The SAS Code Selected In Left Panael.",
style = "color:blue"),
    htmlOutput("inc")
  )
))
```

2. R SCRIPT – EXECUTE WITH SAS

server.R –

The server script controls the actions of the application as per the instructions defined in it. In the example, it requests the SAS server via a script to provide the list of available SAS codes and provides the list to UI.R to make it available for display at the webpage.

```
   # server.R
   library(shiny)
   shinyServer(function(input, output) {

   # First UI input (SAS Code)
# Please refer Appendix 1 for the code

   })
```

3. SHELL SCRIPT - LIST OF SAS CODES FROM SAS SERVER

Saslist.sh -

This script is being called by SERVER.R (R script). This script will be executed at SAS server and find all SAS codes from SAS server and bring the list back to R server. This list will be used to display the list of program, in the dropdown provided to the end user. User can select any of these programs to execute.

```
   #saslist.sh
   #!/bin/bash
   cd /home/ zixa897/SAS-R2
   ls *.sas > /home/ zixa897/SAS-R2/saslist.text
   scp /home/ zixa897/SAS-R2/saslist.text
   zixa897@rsasfusion:/opt/shiny- server/samples/sample-apps/R-SAS/
   FILE="/home/ zixa897/SAS-R2/saslist.text"
   while read line; do
   scp "$line" zixa897@rsasfusion:/opt/shiny-server/samples/sample-apps/
   R-SAS/
   done < $FILE
```

4. SHELL SCRIPT - EXECUTE SAS CODE

r.sh –

After selecting the SAS code, name of selected SAS program will be passed as parameter to this script. This shell script will be called by R Script after selecting the SAS code from UI. This script will execute the selected SAS code and return the output file to R server, so that it can be displayed into R web service.

```
   # r.sh
   #!/bin/bash
   cd /home/zixa897/SAS-R2
   saspgm=$(cat outfile.txt)
   sas $saspgm
   scp r-sas.lst zixa897@rsasfusion:/opt/shiny-server/samples/sample-apps/
   R-SAS
   scp bilingual-report.html zixa897@rsasfusion:/opt/shiny-
   server/samples/sample-apps/R-SAS
   scp univariate-report.html zixa897@rsasfusion:/opt/shiny-
   server/samples/sample-apps/R-SAS
   scp graph.html zixa897@rsasfusion:/opt/shiny-server/samples/sample-apps/
   R-SAS
   scp gchart.png zixa897@rsasfusion:/opt/shiny-server/samples/sample-apps/
   R-SAS
```

5. SAS CODE – PRODUCE REPORT

rsasunivar.sas -

This code is one of the codes listed as an option in the drop down on the web portal in the below stated example. The SAS code selected by the user is sent to the Linux script as an input parameter. Based on the input received the Linux script invokes the appropriate SAS code. The SAS code executes and returns the result back. Then the script transfers the result back to R server, from where the results are displayed.

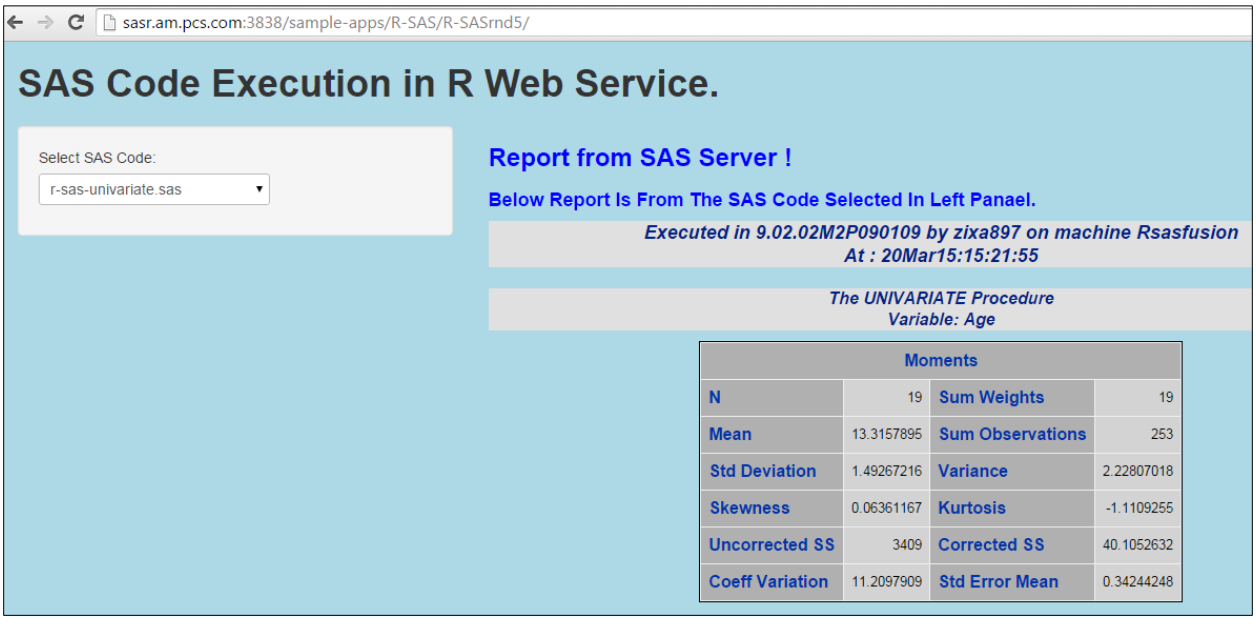
Here in this example, as the user selected to calculate the univariate, the SAS program RSASUNIVAR. SAS is called.

```
ods html file='univariate-report.html';
options locale=en_US;
title "Executed in &sysvlong by &sysuserid on machine &SYSTCPIPHOSTNAME";
title2 "At : %sysfunc(datetime(),nldatm.)";
footnote "%sysfunc(datetime(),nldatm.)";

proc univariate data=sashelp.class;
      var age;
run;

ods html close;
```

Below output (display 1) is from above SAS code in R web service.



Display 1. WebService Screen for End Users

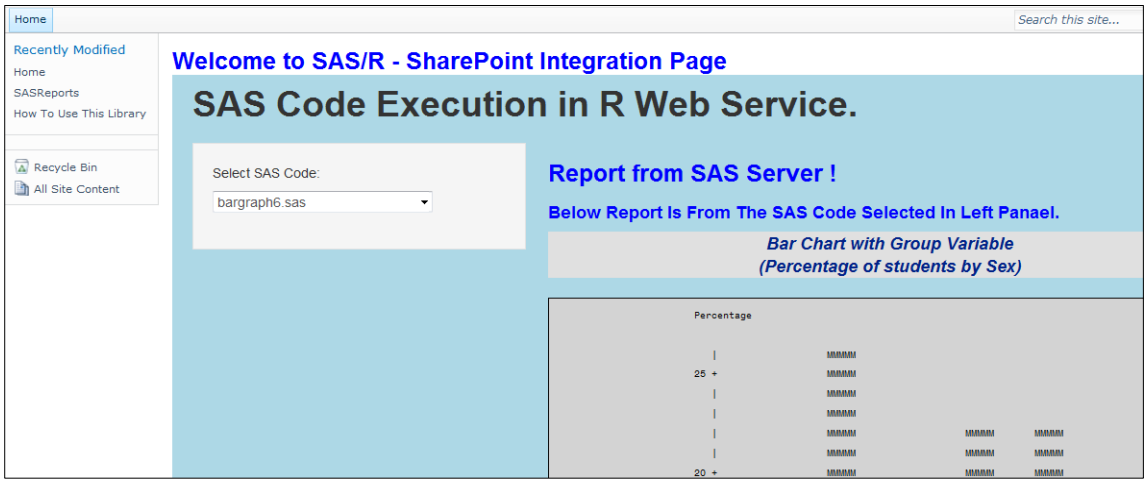
EXECUTE R AND DISPLAY THE RESULT IN SHAREPOINT THROUGH R WEB SERVICE

Once R web service is ready, it can be consumed by SharePoint web portal with the help of Page viewer web part. The page viewer web part has capability to consume data hosted in the other portals or web application (SAS). This will not only help to display the other portal/page content even SharePoint inbuilt security can be applied for that particular content or page.

In SharePoint, a page is designed to provide many options to the user like a button or link through which corresponding web service can be called. The page viewer web part provides the capability to display the content or web page in SharePoint; it allows placing content and controls the content's appearance on a SharePoint page in a synchronous mode. So, the host page (external web page/SAS) and the iframe (SharePoint page) are two different pages, the browser can load them in parallel, resulting in better performance.

Display 2. Main User Screen for Category Selection

In display2, there are three buttons created in a SharePoint page. Each button is associated with R web services. When a user clicks the button, the corresponding R web service is invoked. Page viewer is used here to incorporate the R web-services in SharePoint page.



Display 3. Sample Output from a SAS Code on User Screen

RESOURCE UTILIZATION AND CODE EXECUTION

Any installation (R or SAS) is out of scope of this paper. The use case presented in this paper has SAS and R installed on separate Linux servers. All SAS codes reside and execute on SAS server and R processing happens at R server. SharePoint page displays the results R web service.

When the web service is called, it executes the SAS jobs residing on SAS server. SAS job executes with available resources on SAS server. The example given in this paper executes the selected SAS code, from R web portal in SharePoint. Frequency of SSH between SAS and R server totally depends on the user's requirement and overall all I/O.

Performance and execution time of SAS code depends on SAS server capacity (provided the fact that SAS and R servers are separate server) However, the server resource and architecture can be bounded by the organization policies and decisions.

CONCLUSION

The architecture diagrams, explanations, sample code, and discussions offered in this paper help to understand the utilization of SharePoint and R to share/display the SAS report. This paper describes a prototype how SAS and R can work hand in hand and demonstrates the use of powerful functionalities of SAS/ R in a single application. This can be further enhanced based on the user's R/ Linux scripting capabilities.

ACKNOWLEDGEMENT

Authors would like to thank Greg Scott Nelson and Khushvinder Singh for their review and valuable feedback.

REFERENCE

http://en.wikipedia.org/wiki/R_(programming_language)

http://en.wikipedia.org/wiki/SAS_(software)

http://en.wikipedia.org/wiki/SharePoint#Configuration_and_customization

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Prasoon Sangwan	Piyush Singh	Shiv Govind
prasoon.sangwan@tcs.com	piyushkumar.singh@tcs.com	shivgovind.y@tcs.com
+1-317-488-0594	+1-317-487-9139	+1-317-350-0700

APPENDIX 1:

```
output$SAS_Pgm <- renderUI({
     selectInput("sas",
                 "Select SAS Code:",
                  choices = saspgm_list(),
                  #as.character(saspgm_list),
                  selected = input$sas,
                  multiple = FALSE
                  )
               }
   )

   saspgm_list <- reactive({
     a <- try(system("ssh zixa897@sasr /home/ zixa897/SAS-R2/saslist.sh",
                      intern = TRUE))
     setwd("/opt/shiny-server/samples/sample-apps/R-SAS")
     sas_list <- try(system("ls *.sas", intern = TRUE))
     return(sas_list)
     }
   )

   runsas <- reactive({
     if (!is.null(input$sas))
     wrtfile2()
     a <- try(system("ssh zixa897@sasr /home/ zixa897/SAS-R2/r.sh",
              intern = TRUE))
     }
   )

   getPage<-function() {
    runsas()
    setwd("/opt/shiny-server/samples/sample-apps/RSAS")
    a <- try(system("ls -lrt > test1.text", intern = TRUE))

    if (input$sas == "r-sas.sas")
    return(includeHTML("bilingual-report.html"))

    if (input$sas == "bargraph6.sas")
    return(includeHTML("graph.html"))

    if (input$sas == "r-sas-univariate.sas")
    return(includeHTML("univariate-report.html"))
     }

   wrtfile<-reactive({
     if (!is.null(input$sas))

     setwd("/opt/sihiny-server/samples/sample-apps/RSAS")
     fileConn<-file("/opt/shiny-server/samples/sample-apps
                    /RSAS/output.txt")
     writeLines(c("$sas"), fileConn)
     close(fileConn)
     }
   )
```

```
wrtfile2<-reactive({
  if (!is.null(input$sas))

  setwd("/opt/shiny-server/samples/sample-apps/RSAS")
  sink("outfile.txt")
  cat(input$sas)
  sink()
  try(system("scp /opt/shiny-server/samples/sample-apps/RSAS/outfile.txt
  zixa897@sasr:/home/ zixa897/SAS-R2", intern = TRUE))
  }
)

output$inc<-renderUI({getPage()})
```