DOE GMLC Project 1.4.18 “Computational Science for Grid Management”

Example Implementation of the Computational Framework Design for Power Grid Analysis

November 2016

Example Use Case: Dynamic Security Assessment with Uncertainty

This use case is to conduct dynamic security assessment considering the impact of uncertainties brought by forecasts of energy and loads. High performance computing (HPC) and smart sampling techniques will be applied to better analyze the impact of uncertainties in forecasts. This use case is expected to provide a predictive capability for assess system’s dynamic security and largely improve situational awareness and decision making in power grid operation and planning.

Implementation Overview

The connections of all technical elements through GOSS+GridPACK™ framework are shown in Figure 1. Details of the technical elements are in the following section. Implementation highlights are:

- All inputs and outputs of applications can be archived in GOSS database. They can be assessed through GOSS application programming interfaces (APIs).
- Smart sampling is written in R. It is integrated with GOSS through a GOSS R adapter. It reads forecast and actual values and a contingency list, creates a representative set of scenarios with a reduced set of contingencies as inputs for power flow (PF)/DSA solvers through GridPACK.
- GridPACK is written in C++. It reads all tasks created by smart sampling methods and utilizes a task manager to perform parallel computation of PF/DSA scenarios.
- The post processing script is written in R. It reads all PF/DSA outputs, conducts statistical analysis on the outputs, and computes the probability density functions (PDFs) of interest.
- A web-based visualization tool is written in java/java script. It subscribes to GOSS topics and channels. Once data is available, the visualization tool will consume the data and automatically display the information.
- All applications are connected to GOSS though GOSS client APIs.
- If an application needs to be replaced, or another application needs to be inserted, it can be done conveniently though GOSS client APIs with proper definition of data interfaces among applications.

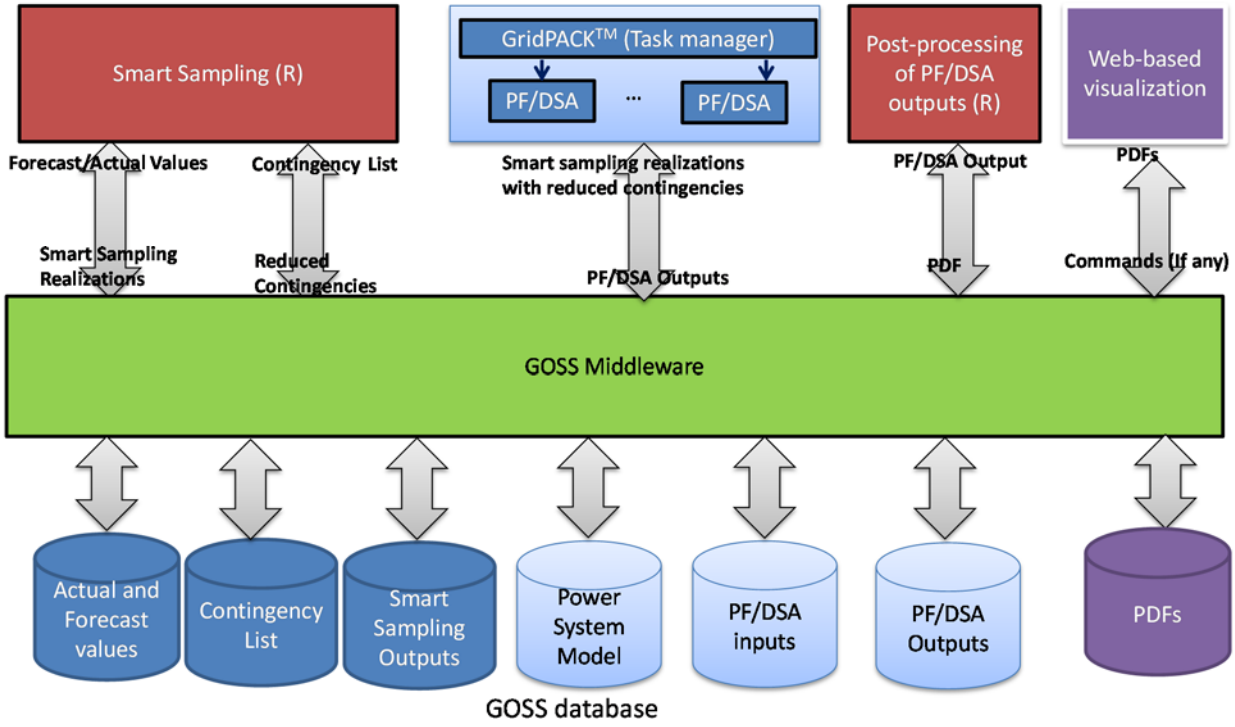

Figure 1: Connections among all technical elements based on GOSS+GridPACK framework

Implementation of Technical Elements

This use case includes following main technical elements:

1. **Smart Sampling Data Analysis**

   Smart sampling techniques have better coverage and less redundancy than the traditional Monte Carlo sampling method. The traditional Monte Carlo sampling method requires significant model simplifications or significant computational resources to achieve reliable results. Smart sampling techniques are more efficient, without compromising accuracy.

   Because of the nature of uncertainty sources in the power grid such as wind turbines and smart loads, system data and forecast data often have internal dependencies and correlations. To incorporate the data dependence structure as well as uncertainties associated with variable generation in the predicted multiple scenarios, a statistical time-series forecasting method is needed. An auto-regressive integrated moving average (ARIMA) based method can be used to develop models that fit historical time-series data so the model can be used to predict the future points in the series. Meanwhile, spatio-temporal data-dependency structure analysis will be integrated into the smart sampling method to reduce the number of scenarios for the overall forecast and probabilistic approaches used.

   The overall smart sampling data analysis framework is shown in Figure 2, with the following inputs and outputs:

   Inputs: historical actual and forecast load and wind generation.

   Outputs: multiple realizations of wind generation and load forecast values based on the smart samples of the load/wind forecast errors. These outputs are used as inputs for Dynamic Security Assessment (DSA) analysis. Because the realizations are independent of each other, they can be

treated as individual cases. These cases can be easily handled in parallel by high-performance computing techniques to reduce computational time.

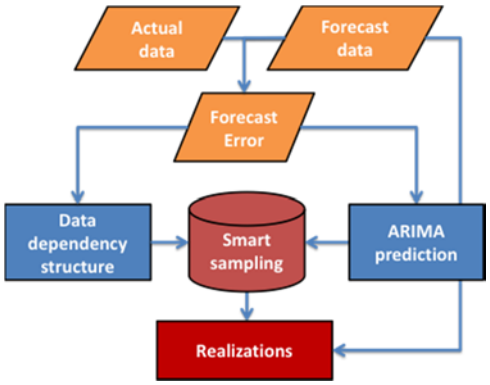

Figure 2. Smart Sampling Data Analysis Framework

2. **High Performance Computing Techniques**

GridPACK™ can be used to improve computation speed through high-performance computing techniques. After reading basic system and model information and realizations, GridPACK™ can perform DSA for each realization through a task manager.

A two-level parallelization mechanism has been developed in GridPACK™. At each realization level, GridPACK™ can allocate the realizations to different groups based on their availability. In each group, multiple cores will be used to perform DSA studies to reduce computational time.

GridPACK can be integrated with the GOSS middleware to allow data transfer between GridPACK and other applications through GOSS client APIs.

The outputs of smart sampling methods, i.e. realizations with a reduced set of contingency list, are the inputs for GridPACK to conduct powerflow and DSA computations. The outputs of power flow and DSA for each realization will be analyzed by statistical analytics methods to study the impact of uncertainties on system's power flow and dynamic behaviors.

3. Statistical Analytics Methods

Based on the outputs of DSA for all realizations, statistical analytics methods will be applied to compute the probability density function of DSA outputs that users are interested in and provide predictive capability to users. The outputs can be transferred to other applications, such as web-based visualization tools or other applications that leverage this output through GOSS middleware.

4. **GOSS Middleware**

GOSS is a middleware platform that facilitates deployment of applications for future power grid. Its architecture is shown in Figure 3. GOSS separates data sources from applications and provides a unified API to seamlessly fetch data from multiple data sources. GOSS also allow data exchange and communication among applications. GOSS is built on top of Apache ActiveMQ messing middleware. This allows synchronous, asynchronous and event based communication channels between application and data sources. Using GOSS, power grid applications developed using different

software languages and platforms can communicate with ease. GOSS is designed to enable complex application workflows and allows data pipelines to be built to link multiple applications to produce results. Using GOSS, applications can access data from multiple sources; transform data into desired format; publish data for other applications; store data in available data sources; and subscribe to data coming from other applications. There are three ways an application can connect with GOSS and leverage its capabilities: client API, web services and web sockets. GOSS provides a customizable authentication and authorization layer that takes care of identity management as well as fine-grain access control.

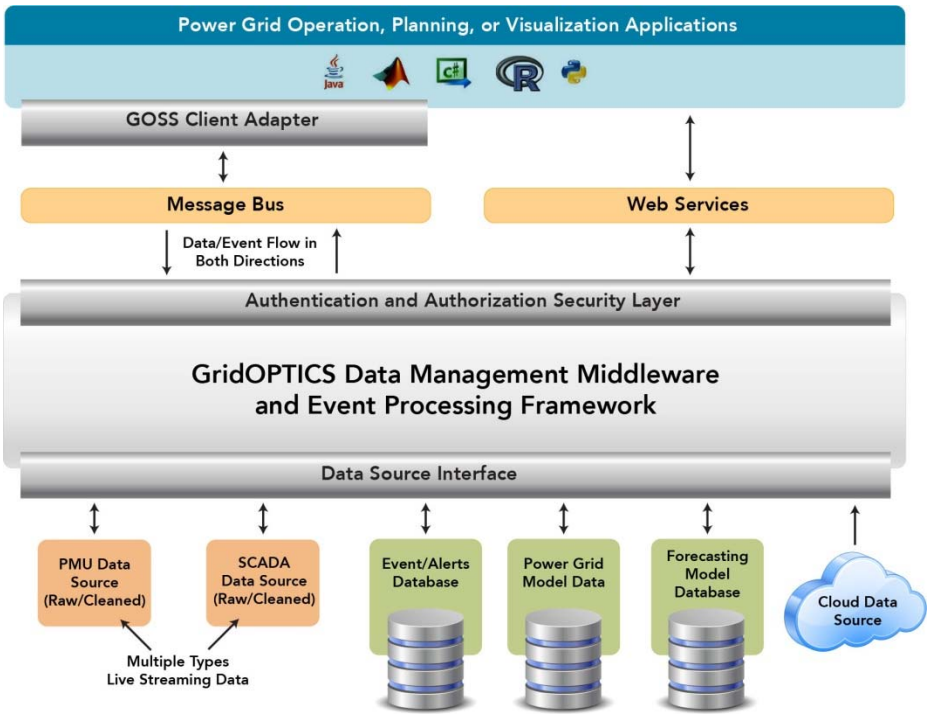

Figure 3. GOSS architecture

GOSS can be integrated with different applications, including, but not limited to, GridPACK, advanced statistical methods, and web-based visualization tool. Currently, GOSS support applications in following languages: Java, C++, C#, VB, R, MATLAB, and Python.

5. GOSS-GridPACK Integration

GridPACK is written in C++ and uses ActiveMQ-CPP library to connect with GOSS. Once the library is built and included in GridPACK, GOSS connection details are added in GridPACK input file as shown in Figure 4.

```
<channelTopic> goss/gridpack/dynamic_simulation/output </channelTopic>
<channelURI> tcp://server:port </channelURI>
<username> user</username>s
<password> pswd</password>
```

Figure 4: GOSS connection input in GridPACK

ChannelTopic is the channel name where GridPACK publishes its results to GOSS that can be subscribed by other applications. ChannelURI contains the IP address and port details of the machine where GOSS is deployed and running. The username and password is the account information to connect with GOSS. With all this information GridPACK is ready to use the API to connect with GOSS in order to receive and publish data. Figure 5 described the pseudo code in GridPACK that creates the connection with GOSS using information provided above.

```
connectionFactory = new ActiveMQConnectionFactory(channelURI)
connection = connectionFactory.createConnection(username, password)
connection.start()
session = connection.createSession(Session::AUTO_ACKNOWLEDGE)
```

Figure 5: Pseudo code in GridPACK for creating GOSS connection

Once the connection is done, GridPACK can subscribe to data from other application or publish its results on the channelURI mentioned above. Figure 6 describes the pseudo code for publishing GridPACK results. These results published on the channel can be subscribed by other applications such as web-based visualization tools to display the results in a graphical manner to users.

```
destination = session.createTopic(channelTopic)
producer = session.createProducer(destination)
message = session.createTextMessage(message))
producer.send(message.get())
```

Figure 6: GridPACK pseudo code to publish results

6. Web-Browser-Based Visualization Tool

The web-browser based visualization tool can display outputs of applications such as such as GridPACK and R in this use case, once they are ready through subscribing GOSS topics/channels. It is cross-platform and very convenient for users to analyze program outputs. It also allows Windows users to stay in Windows environment to perform analysis without knowing any HPC or other Linux-based applications, which minimize the learning curve for new HPC applications.

The overall flowchart of this use case is shown in Figure 7.

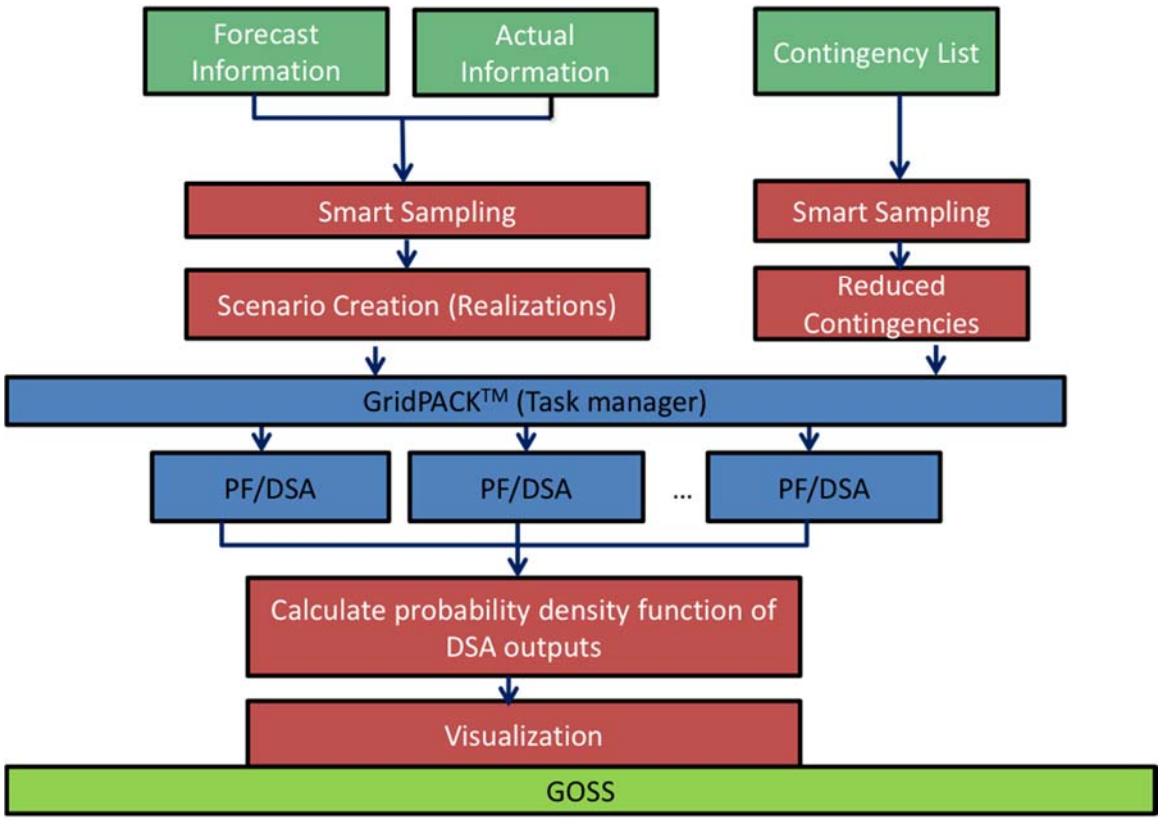

Figure 7: The overall flowchart of the use case