Parallel Execution of BW Queries in ABAP Applications

Applies To:

SAP BW 3.5 and higher versions

ABAP 4.6 and higher versions

Summary

This paper discusses how you can execute BW queries in parallel in a BSP application using ABAP code.

By: Satyashilpa Bowalekar

Company: L&T Infotech (Powai), Mumbai - India

Date: 2nd February 2006

Background

With customer expectations at their peak, we strive hard to implement the latest technologies to achieve our goals. Furthermore, customer satisfaction has become the prime objective of every service organization. In this context, companies increasingly are working creatively, combining and leveraging familiar technologies, for example, by exploiting the close relationship between ABAP and BI.

SAP BI and ABAP are very tightly integrated indeed. Most of the enhancements done in SAP BI are done with ABAP. We use ABAP in update routines, transfer routines, and customer exits to enhance a data source or update a variable in BEx reports. Similarly, we can use ABAP to bring BW query data into an ABAP-based application, if that application needs BW data in its execution.

Scenario

The BW queries 'Zxxxx_Q001' and 'Zxxxx_Q002' need to be executed for product group 'PG1' **in parallel** in a BSP application. The query output needs to be displayed at the same instant. Users should see both BW reports in the BSP application at the same time. Furthermore, there should be no delay in viewing the output of either BW report.

Technical Approach

One approach to deal with the above scenario is to implement the following two steps:

1. Use an SAP-delivered ABAP Class to execute the BW queries
2. Execute the BW queries in parallel through asynchronous function calls (aRFC)

The main focus of this article is to understand the execution of BW queries in parallel using aRFC.

Use an SAP-delivered ABAP Class to Execute BW queries

To execute BW queries in ABAP we make use of the following SAP-delivered classes:

- CL_RSR_REQUEST
- CL_RSR_DATA_SET

One basic approach to execute the BW queries in our scenario would be to create two function modules for executing the respective queries. These function modules can then be called in the main program. The functions can be implemented in the following sub-sections:

1. Object creation for class 'CL_RSR_REQUEST'
2. Setting up filters for BW queries
3. Storing data for query view
4. Parsing data into customized table

Each of the above sub-sections are explained in detail below:

Object creation:

Create objects for the class CL_RSR_REQUEST by exporting the unique query id obtained as GENUNIID from table RSRREPDIR.

Setting up filters for BW queries:

Using method SET_FILTER, we set the filters (if required). In our example, we set the characteristic value of 'product group' as 'PG1'.

Storing data for query view:

The dataset contains the data for the query view. In our example, we have defined a class variable R_DATASET which is a TYPE REF of CL_RSR_DATA_SET. It is used to hold the data for query view. It is a deep structure with axis info, axis data, cell data, and text symbols.

In this example we have created:

1. 'wa_axis' using r_dataset->n_sx_version_20a_1-axis_data

   (The AXIS_DATA component describes the contents of the axis)

2. 'wa_cell' using r_dataset->n_sx_version_20a_1-cell_data

   (The CELL_DATA component describes the value cells)

Note:

The N_SX_VERSION_20A_1 is an attribute of CL_RSR_DATA_SET and is filled automatically. The data consists of both the metadata (a catalog of characteristics (AXIS_INFO) and so on) and the displayed values (AXIS_DATA and CELL_DATA).

Parsing data into a customized table

Each BW query 'ZXXXX_Q001' and 'ZXXXX_Q002' needs to be displayed in a particular format as per the customer requirement. This means we need to create a customized table having structure similar to our query results. The data obtained in 'wa_axis' and 'wa_cell' above now needs to be parsed accordingly in the customized table (ZZITAB in our example) using simple ABAP syntax.

Summing it up, to execute BW queries in ABAP, our function would need to import the unique GENUNIID for the BW reports and use a customized table to store the data for the query view using the above logical sub-sections.

Thus, we create two functions ‘ZCALL_1’ and ‘ZCALL_2’ which will execute BW queries ‘ZXXXX_Q001’ and ‘ZXXXX_Q002’ respectively.

A number of posts and weblogs are available on ‘Execution of BW queries in ABAP’ on SDN. You can explore to the minutest details for further information.

Execution of BW Queries in Parallel

Now that we know how to execute BW queries using classes in ABAP, let’s turn to executing queries in parallel. The key things to be noted in this are:

The functions that execute the BW queries (ZCALL_1 and ZCALL_2) need to be remote enabled. Remote Enabling would start the function module function asynchronously in a new session. In contrast to normal function module calls, the calling program resumes processing as soon as the function module is started in the target system. It does not wait until the function module has finished.

In Asynchronous remote function calls (aRFCs), the user does not have to wait for the function module to complete before continuing the calling dialog. There are few important characteristics of asynchronous RFCs like:

- When the caller starts an asynchronous RFC, the called server must be available to accept the request.
- The calling program can receive results from the asynchronous RFC

The main program that calls the BW queries ‘ZXXXX_Q001’ and ‘ZXXXX_Q002’ in parallel can be divided into four major sub-sections

1. Identification of the unique GENUNIID from table RSREPDIR for each query
2. Calling the remote enable functions asynchronously
3. Defining a ‘FORM’ routine for receiving the results from the RFC

4. Perform checks before the display of final query output. This is done using 'WAIT UNTIL', which interrupts the program execution as long as the result of the logical expression **`semaphore`** (in this case) is false.

Each of the above sub-sections is explained in detail below:

Identification of the unique GENUNIID from table RSREPDIR

Here the unique GENUNIID for each query is derived and transferred as import parameter (P1) to the functions. A code sample would look like :

```
select single GENUNIID from RSRREPDIR into P1  where COMPID = 'ZXXXX_Q001'.
```

Calling the remote enabled functions asynchronously

Call the functions 'ZCALL_1' and 'ZCALL_2' asynchronously meaning that they start independent processing of the respective queries and the main program proceeds ahead with further steps without actually waitng for ZCALL_1 and ZCALL_2 to complete execution. ' The code sample for calling function asynchronously would look like :

```
call function 'ZCALL_1'
  starting new task 'TASK1'
  destination 'NONE'
  performing RETURN_INFO_1 on end of task
  exporting
    P1 = P1.
```

RETURN_INFO_1' is basically defined to receive the results back and it's code is defined in the next section of this article

Defining 'FORM…' routine for receiving the results from RFC

A code is written to receive results back for 'RETURN_INFO_1'. The query output is stored in a particular structure and is updated in an internal table within the respective function modules. This table is then given back to the main program in this definition. Some flag, say 'SEMAPHORE_1', is set to indicate completion of the function call.The code sample would look like:

```
form RETURN_INFO_1 using TASKNAME.
  receive results from function 'ZCALL_1'
            tables zzitab = zzitab.
  SEMAPHORE_1 = 'X'.
endform.
```

Checks before final query output display

The final task now is to display the query results. The main program resumes when the flag is set after function execution (that means on completion of function call). It then displays the results from the table received from the remote enable function.

```
wait until SEMAPHORE_1 = 'X'.
loop at zzitab.
  write :/ zzitab-text1 , zzitab-text2 , zzitab-text3 , zzitab-text4 ,
zzitab-text5.
endloop.
```

Thus, finally our ABAP code could now execute BW queries for same product 'PG1' and in parallel and display results at the **same moment**.

Author Bio

Satyashilpa Bowalekar (also called Shilpa) is currently working with L&T Infotech. She has more than 2.5 years experience with SAP. During this period, she has worked extensively as a SAP BI Consultant on production support as well as on development assignments.

Parallel Execution of BW Queries in ABAP Applications

Disclaimer & Liability Notice

This document may discuss sample coding or other information that does not include SAP official interfaces and therefore is not supported by SAP. Changes made based on this information are not supported and can be overwritten during an upgrade.

SAP will not be held liable for any damages caused by using or misusing the information, code or methods suggested in this document, and anyone using these methods does so at his/her own risk.

SAP offers no guarantees and assumes no responsibility or liability of any type with respect to the content of this technical article or code sample, including any liability resulting from incompatibility between the content within this document and the materials and services offered by SAP. You agree that you will not hold, or seek to hold, SAP responsible or liable with respect to the content of this document.