



White Paper

High Performance XML Processing and Bulk Loading to Oracle System with Microsoft Integration Services

A common pattern when processing data electronically is parsing the data, applying certain rules, and then loading to destination systems. Hand coding a data processing solution in 3GL (C#, C++, VB, Java* and so forth) requires substantial coding effort to develop and maintain the system. In addition, handling large input files (XML or any other format) require careful considerations on efficiently processing the data.

Microsoft* Integration Services addresses this problem by providing a powerful environment for visually building the solution with minimal coding effort.

Another key factor that impacts scalability is the number of round trips to the database. One round trip per row to the database seriously limits the throughput of an application. Processing data in batches by sending several rows worth of data in a single database round trip allows an application to scale efficiently. However, bulk loading to Oracle* from Integration services framework today requires the use of third party components. In this white paper, we are going to look at a solution option for building high throughput application to parse large XML files and perform bulk load to Oracle database.

* Other names and brands may be claimed as the property of others.

XML Processing

Typical 3GL approach to XML processing would involve understanding the schema and parsing the elements of interest from the XML instance. While small-medium documents can be loaded as in-memory trees, large documents would require streaming or chunking to efficiently process the input. Writing the code in 3GL would require understanding these important details and hand coding an appropriate solution.

Integration services software on the other hand, hides the complexity involved in parsing an XML and exposes XML as in-memory relational table. Large XMLs are mapped to in-memory table by transparently allocating memory for the rows in use and freeing up the memory once the batch of rows is consumed.

Oracle* Bulk Loading

When an application needs to insert (and update) several rows of data, it is optimal to process several rows of data in one round trip. This bulk processing or fast load capability allows an application to scale efficiently.

Performing bulk operation, also known as Array Binding, is relatively straight forward with ODP.NET drivers as described in this [How to: Bind an Array](#) to an ODP.NET Database Command article.

Performing bulk operations in Oracle from Integration Services requires few additional steps and coding. The rest of this paper discusses throughput numbers and a code sample on how to perform bulk operation through integration services.

Unit and Bulk Performance Characterization

The "Customers" table available Oracle SH schema was used as a test bed for validating bulk insert using Integration Services. This table has 24 columns of mixed numeric and varchar data types.

* Other names and brands may be claimed as the property of others.

Input XML File

The data to be loaded to Customers table was in XML format with each XML file having several rows of data.

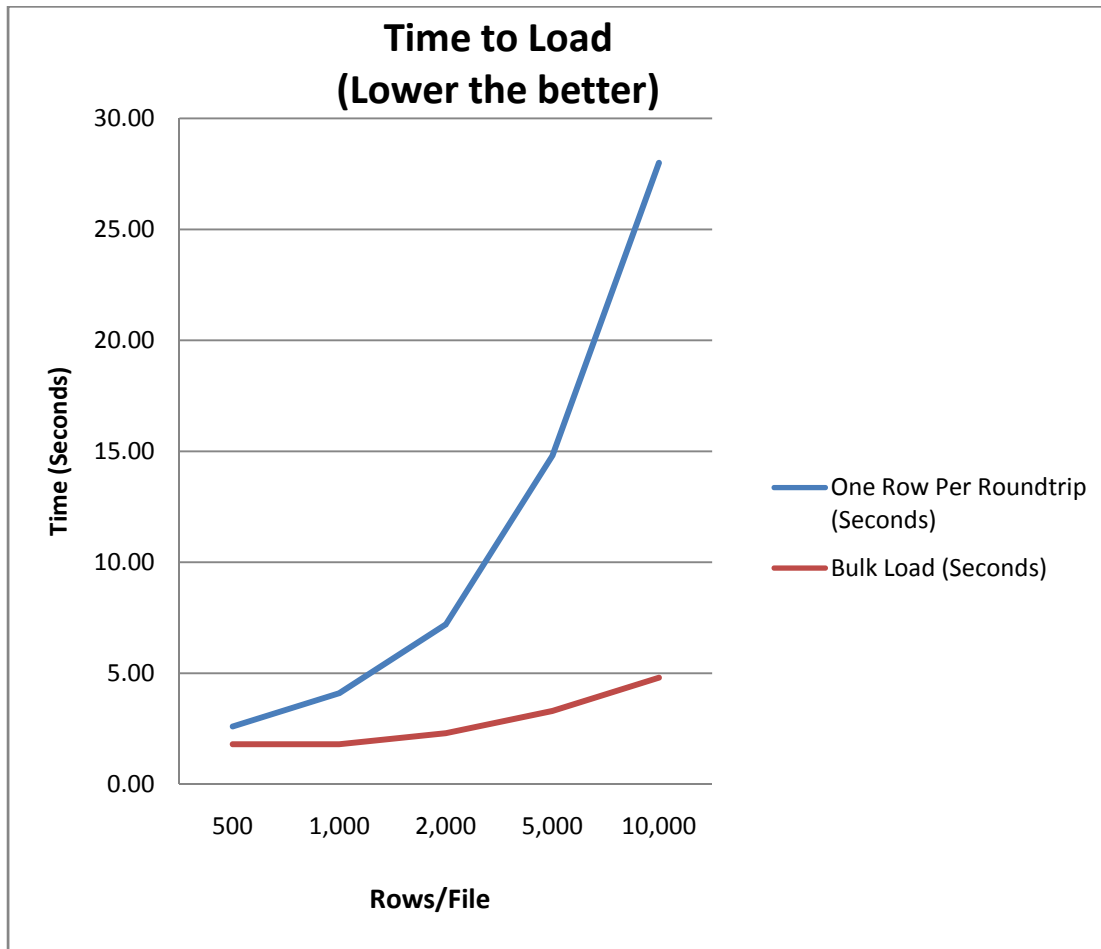
```
<?xml version="1.0" ?>
- <ROWDATA>
- <ROW>
  <C0>1</C0>
  <CUST_ID>1</CUST_ID>
  <CUST_FIRST_NAME>Abigail</CUST_FIRST_NAME>
  <CUST_LAST_NAME>Kessel</CUST_LAST_NAME>
  <CUST_GENDER>M</CUST_GENDER>
  <CUST_YEAR_OF_BIRTH>1946</CUST_YEAR_OF_BIRTH>
  <CUST_MARITAL_STATUS />
  <CUST_STREET_ADDRESS>7 South 3rd Circle</CUST_STREET_ADDRESS>
  <CUST_POSTAL_CODE>30828</CUST_POSTAL_CODE>
  <CUST_CITY>Downham Market</CUST_CITY>
  <CUST_CITY_ID>51396</CUST_CITY_ID>
  <CUST_STATE_PROVINCE>England - Norfolk</CUST_STATE_PROVINCE>
  <CUST_STATE_PROVINCE_ID>52591</CUST_STATE_PROVINCE_ID>
  <COUNTRY_ID>52789</COUNTRY_ID>
  <CUST_MAIN_PHONE_NUMBER>127-379-8954</CUST_MAIN_PHONE_NUMBER>
  <CUST_INCOME_LEVEL>G: 130,000 - 149,999</CUST_INCOME_LEVEL>
  <CUST_CREDIT_LIMIT>9000</CUST_CREDIT_LIMIT>
  <CUST_EMAIL>Kessel@company.com</CUST_EMAIL>
  <CUST_TOTAL>Customer total</CUST_TOTAL>
  <CUST_TOTAL_ID>52772</CUST_TOTAL_ID>
  <CUST_SRC_ID />
  <CUST_EFF_FROM>1/1/1998</CUST_EFF_FROM>
  <CUST_EFF_TO />
  <CUST_VALID>I</CUST_VALID>
</ROW>
+ <ROW>
+ <ROW>
+ <ROW>
</ROWDATA>
```

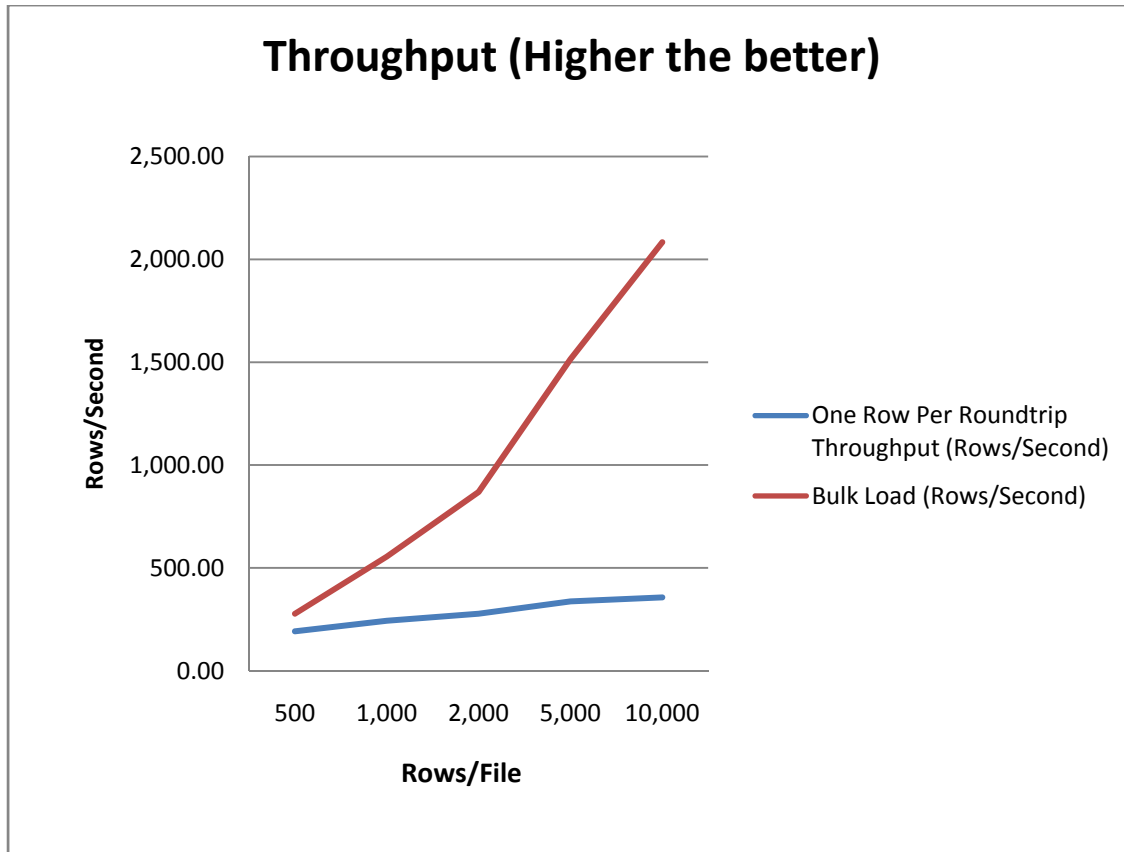
Performance

The table and chart summarizes the performance difference between row-by-row processing and bulk load processing in integration services with Customers Data set.

Please note that several factors influence the overall performance and depending on the number of columns and column length, appropriate batch size needs to be selected.

Rows/File	File Size (MB)	Row-By-Row Insert (Seconds)	Row-By-Row Throughput (Rows/Second)	Bulk Insert Time (Seconds)	Bulk Load Throughput (Rows/Second)
500	0.52	2.6	192.31	1.8	277.78
1,000	1.03	4.1	243.90	1.8	555.56
2,000	2.07	7.2	277.78	2.3	869.57
5,000	5.17	14.8	337.84	3.3	1,515.15
10,000	10.33	28.0	357.14	4.8	2,083.33





Code Approach

A custom destination Script Component available under Data Flow Transformations was used for implementing the bulk insert logic.

The destination script component, by default, provides three methods:

1. `public override void PreExecute()`
2. `public override void PostExecute()`
3. `public override void Input0_ProcessInputRow(Input0Buffer Row)`

All these methods are automatically invoked by the integration services data flow pipeline during appropriate events:

PreExecute - Preparing the custom destination component to consume data

Input0_ProcessInputRow - Invoked for every row that arrives in the pipeline

PostExecute - After all rows are processed, this method is invoked to perform cleanup.

At this point, all that needs to be done is allocate buffers within this script component class and populate the buffers whenever a new row arrives. Once, the number of rows reaches the defined batch size, stored procedure is invoked to pass the buffered data using array binding technique available in ODP.NET.

[Microsoft.SqlServer.Dts.Pipeline.SSISScriptComponentEntryPointAttribute]

```
public class ScriptMain : UserComponent
{

    private OracleConnection _conn;
    private OracleTransaction _tx;
    private int _rowCount;
    private int _rowsPerRoundTrip;
    private int _iterations;

    // Define buffers for all the columns
    private int[] countryId;
    private string[] custCity;
    private int[] custCityId;

    ....

    public override void PreExecute()
    {
        base.PreExecute();
        _rowCount = 0;
        _rowsPerRoundTrip = <read from configuration variable>;
        _iterations = 0;

        string connectionString = <read from configuration variable>;
        _conn = new OracleConnection (connectionString);
        _conn.Open();

        // Allocate space
        countryId = new int[_rowsPerRoundTrip];
        custCity = new string[_rowsPerRoundTrip];
        custCityId = new int[_rowsPerRoundTrip];

        ....
    }

    // Make sure that the last batch of rows are processed and loaded to database

    public override void PostExecute()
    {
        base.PostExecute();
        if (_rowCount > 0)
        {
            LoadData();
        }

        _conn.Close();
    }
}
```

```

// Process the incoming rows and copy to buffer

public override void Input0_ProcessInputRow(Input0Buffer Row)
{
    countryId[_rowCount] = Row.COUNTRYID;
    custCity[_rowCount] = Row.CUSTCITY;
    custCityId[_rowCount] = Row.CUSTCITYID;

    .....

    _rowCount++;

    if (_rowCount > _rowsPerRoundTrip - 1)
    {
        LoadData();
    }
}

// Function to invoke database stored procedure

private void LoadData()
{
    _tx = _conn.BeginTransaction();

    OracleCommand cmd = new OracleCommand("Stored Proc Name", _conn);
    cmd.CommandType = CommandType.StoredProcedure;
    cmd.ArrayBindCount = _rowCount;

    OracleParameter pcountryId = new OracleParameter("pi_country_id", OracleDbType.Int32);
    pcountryId.Direction = ParameterDirection.Input;
    pcountryId.Value = countryId;
    cmd.Parameters.Add(pcountryId);

    ....

    cmd.ExecuteNonQuery();
    cmd.Dispose();

    _rowCount = 0;

    _tx.Commit();
}

// Oracle Stored Procedure definition

CREATE OR REPLACE PROCEDURE proc_insert_customer
(pi_country_id      IN number,

```

```
    pi_cust_city    IN VARCHAR2,  
    ...  
  ) IS  
BEGIN  
  
insert into customers(cust_id, cust_first_name, cust_last_name,....)  
  values (pi_cust_id,pi_cust_fname,pi_cust_lname,...);  
  
END;
```

Conclusion

In this paper, we saw how custom Oracle Bulk Processing capability can be incorporated in to Integration Services while leveraging integration services for parsing XML data and preparing the data for backend systems.

This mix provides an efficient mechanism to build scalable applications on Intel® Platforms while reducing the time-to-market.

About the Author

ChandraMohan Lingam (Chandra) is an Architect at Intel where he works for the Technology Development Group developing solutions to address Intel's manufacturing needs. Chandra has a MS degree from Arizona State University, Tempe and a Bachelors Degree in Engineering from Thiagarajar College of Engineering, Madurai, India.