Data Warehouse Population Platform

Jovanka Adzic, Valter Fiore

Telecom Italia Lab
Via G. Reiss Romoli, 274
10148 - Torino, Italy

{jovanka.adzic, valter.fiore}@telecomitalia.it

Abstract. The input data for a data warehouse, coming from operational systems, are not immediately ready for loading into data warehouse. Cleaning and integration could be necessary. Moreover, the input data must be transformed and translated into a format more suitable for analytical purposes. This paper presents a generalised platform for population of data warehouses named *Data Warehouse Population Platform* (DWPP), a set of modules whose aim is to resolve typical aspects arising during the transformation and loading vast amount of data into data warehouse. In Telecom Italia Lab we have realised several Population Systems based on DWPP. The Case Study section describes one of them: Population System for Mobile Network Traffic Data Warehouse.

1 Introduction

Data Warehousing is about technology and practice related to need to systematically integrate data from multiple distributed data sources and to use that data in suitable form to support business decision-making and enterprise management. In order to manage own business a telecommunication company have to collect vast amounts of detailed data, such as Call Detail Records (CDRs), coming from different source systems. These CDRs are used, first of all, for billing purposes and secondly, they are stored in decision support databases, very large databases, in order to answer different needs. For example Customer Relationship Management (CRM) have to deal with large volumes of data containing a lot of information about a company's business and behavioural patterns of the company's customers. The challenge is how to deal with this vast amount of data, how to store and analyse it, in order to discover, with low latency, valuable information about business and customers.

The input data from an operational system are not immediately ready for loading into data warehouse. Firstly, the input data may need cleaning and integration with other data. Secondly, the input data must be transformed and translated into a format more suitable for analytical purposes. In order to facilitate and manage this Data Warehouse operational processes, specialized tools are already available in the market under the general name Extraction-Transformation-Loading (ETL) Tools. The most prominent task of these tools include (a) extraction of relevant information at the source side; (b) transformation and integration of the data coming from multiple sources into a common format; (c) cleaning of the resulting data according to business rules and (d) loading and propagation of the data to the data warehouse and/or data marts.

In our experience, we had to deal with several context characterized by: lot of detailed data (CDRs, traffic alarm and/or measurements,..) coming from different sources in flat file format, strong performance requirements (time available from ETL activities from 1 to 4 hours) and simple and efficient day-by-day operational processes management (fast and consistent recovery).

The contribution of this paper is about our experience in developing and deploying different very large data warehouses based on our own tool for ETL activities. We have already evaluated some ETL tools available in the market, but we could not find one corresponding to performance (time and throughput constraints) requirements in transforming and loading very large amount of data (CDR Transformation & Loading). This paper presents a generalised platform for population of data warehouses named *Data Warehouse Population Platform (DWPP)*, a set of modules whose aim is to resolve typical aspects of transformation and loading vast amount of data into data warehouse. With DWPP we do not concentrate on extraction of data from different kind of source systems, but we assume to have input data already exported or produced in flat file format *DWPP* is tuned for data warehouses on Oracle DBMS and it is independent from the domain of data (CDRs, Customer Data, Transaction Data,…).

The rest of the paper is organized as follows: Section 2 describes DWPP principal features, such as dispatching input data, synchronizing application transformation units, loading output data into Oracle database tables, etc. Section 3 presents a scenario of a Data Warehouse Population System, based on DWPP, for Mobile Network Traffic Data Warehouse that we have deployed for an important mobile operator.

2 Data Warehouse Population Platform (DWPP) Features

DWPP has been used in deploying different data warehouse population systems on Unix S.O and Oracle 8i/9.i DBMS.

The *Data Warehouse Population Platform* (DWPP) is a set of C software modules organized in a shared library on Unix S.O, that include features such as:

- Identification, collection and distribution of files containing input data for the transformation units;
- Synchronization of data transformation units;
- Fast lookup functions for surrogate key valorisation;
- Bulk data loading (direct / conventional path) into database tables;
- recovery features;

The DWPP Modules are well structured: the first level modules wrap Operating System and Oracle DBMS (via OCI), the second level modules offer particular features for data warehouse population (ETL activities).

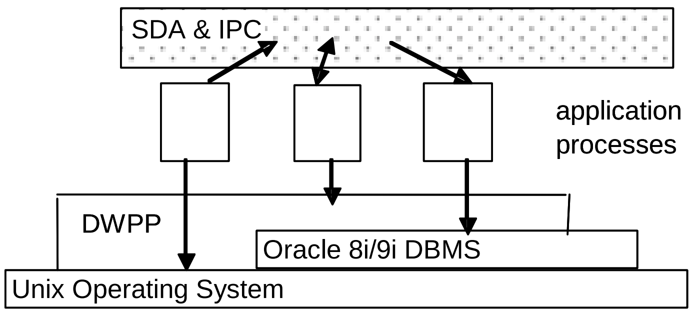
The typical architecture of a populating system based on DWPP contains the following jobs:

The **Master** process allocates shared resources used by all process units and takes care of their synchronization.

The **Listener** process collects data flows, coming from source systems and dispatches them towards appropriate data transformation units.

The **Shared Data Area** (SDA) is a main memory segment used by the application transformation units. The SDA is divided into many sub-segments whose access can be restricted to some units or be made public. The SDA offers some of the basic functions of an In-Memory Data Base allowing fast search in lookup tables that works as a cache for database tables. This feature is heavily used during surrogate key transformation, done by transformation units.

The **Appl. Units** can contain different application specific transformation units which transform the input data according to a specific application logic.



2.1 LSN Module – Input Data Dispatching

The Listener process takes care of collecting data-flows coming from local/remote source systems and dispatching them to data transformation ProcessUnits. Each flow consists of one or more set of files, each of them with associated pre-processing action (split and specific termination criteria). As soon as possible file or chunk are sent to destination ProcessUnit for transformation. DWPP identifies each flow assigned to a particular ProcessUnit by:

- The incoming directory (local o remote) where Listener looks for files;
- The Process Unit to which files must be sent;
- The file name pattern; listener handles both flat and master-detail files, fixed or variable record len;
- The waiting timeout;
- Pre-processing operation that Listener must compute (uncompress, split, header and footer skip); .
- The number of threads dedicated for pre-processing operation
- The flow termination criteria (maximum number of files, timeout, custom function).

The Listener process is a "declaration module": consists of a set of calls to DWPP library function to define the characteristics of data flows as above indicated and a control function that effectively manage the acquisition/dispatching process.

2.2 SDA Module – Shared Data Area

The *Shared Data Area (SDA)* is a main memory segment used by the application and system processes to maintain useful data. The DWPP module named SHM exposes an API to manage shared memory accesses. Allocation and de-allocation of SDA is usually done by Master process and all other processes just have to gain access to it.

One of the very useful features in a data warehouse population is the possibility to transform keys used by source systems into (surrogate) keys used inside the data warehouse. This transformation involves a fast search on a lookup table. The time consumed by this operation is critical in high volume population systems. DWPP allows the managing of lookup tables into segment of SDA, called *Rapid Search Area (RSA)*. The function *RSALoad* permits the loading of data set (result of specified query) from data base into *SDA* memory segment. In this way it's possible to initialize the lookup tables before the starting of any application ProcessUnits.

Each record of the lookup table is a structure containing all fields needed by the data transformation functions, so there is no need to access DBMS during transformation. All data necessary for transformation are cached in SDA from data base.

A lookup table can be organized for binary search (ordered and only for read access) or for hash search (not ordered and for read/write concurrent access). It is possible to define lookup table structures that are made of two tables in order to avoid locking the whole lookup table for every access. This lookup table is made of a Primary table and a Secondary table (both hash). The fast search of records in the Primary table is done without locking. Records in this area are read-only and loaded during the initialization phase. Only the accesses to the records in the Secondary table need the locking mechanism. New records can be added or old record can be updated only in the Secondary table.

The DWPP module named RSA (Rapid Search Area) exposes an API to manage lookup tables: definition, initialization, data retrieval and data insert/update/upsert.

In NearRT scenarios, is possible to retain SDA in memory, between a cycle and another, refreshing it only when necessary.

2.3 SYNC Module – Units Synchronization

The population system based on DWPP is made of elementary pieces, called *ProcessUnit*. The ProcessUnit can be Unix-process or thread implemented and multi-instantiated to increase parallelism. The Master process is responsible for activation and synchronization of all ProcessUnits during the population phase. Synchronization is defined at ProcessUnits level. If a ProcessUnits P_i depends on ProcessUnits P_k ($P_k \rightarrow P_i$) that means that all instances of P_i can begin working when all instances of P_k are terminated with a particular termination state. With the definition of dependency rules for ProcessUnits, it is possible to specify also the expected termination states of the ProcessUnits another ProcessUnits depends on. The SYNC module allows the definition of complex synchronization rules (AND / OR) that determine dependency graph.

Each instance of a ProcessUnit can reach different states:

- QUIET/DOWN: state during system startup prior to allocation and initialization of data structures;
- READY: state after the initialization phase; waiting for the termination of the ProcessUnits it depends on**;**
- STARTABLE: state when all instances of ProcessUnits it depends on are terminated; it has the permission to start;
- WORKING: state during the data transformation phase;
- SUCCESS: termination state of a successfully terminated instance;
- WARNING: termination state of an successfully terminated instance but with some warnings;
- FAILURE: termination state of an unsuccessfully terminated instance due to a failure that has interrupted transformation phase.
- TIMEOUT: termination state of an unsuccessfully terminated instance due to insufficient time to finish transformation.

Over the ProcessUnit structure and synchronization rules it is possible to define an aggregation level called *Component*. A Component is a logical application task characterized by atomicity. It has associated a special ProcessUnit – *RecoveryProcessUnit* with save/restore logic. The Component has fundamental role in recovery in case of failure during a regular population cycle and also gives a higher level to the structure of the ETL activities.

Component can contains one or more ProcessUnits, and have two kind of dependency: *normal* and *on recovery* dependency. A typical example of an on recovery dependency is a case of initialization component from which all the other components depends on.

When application runs day-by-day the SYNC Module evaluates the termination status of every components and tracks this information in historical status table. When a failure is detected, after understanding and removing cause of failure, we can run the application in “recovery” mode simply with "master -r". Based on information retrieved in historical tables, the SYNC Module automatically can detect the only Components need to re-run (terminated with failure) and launch the ProcessUnits part-of them. In this way, only the minimal part of the whole application (ETL Activities) will be re-run, automatically, in case of failure. The RecoveryProcessUnit is responsible for maintaining data base integrity. Recovery does not occur only when "application itself" fails: more even occurs when one of the source systems, or network, or other fails.

In NearRT scenarios, where the application needs run every hour or few minutes Sync support an similar techniques for recovery: a second instance of the master (run as "master -cr") looks at status tables periodically for failures of the primary application (Master operates in this case as a daemon which wakes up as defined in schedule clause) and, when a failure is detected, automatically retry the population cycle. Some simply rules define recovery depth and max retries: no other is necessary at code level

Like as a listener process, a Master process is simply formed of a set "declaration" (set of call to DWPP function defining processUnit, components, dependencies and so on) describing the structure of application ad a control function that effectively manage the syncronisations process.

2.4 LDR Module – Transforming & Loading Data

All DWPP application process use the loading module (LDR) to transform and load data into the Data Warehouse tables.

The LDR module is one of the most important and most performing module of DWPP, characterized by so-called "*Asynchronous Read/Transform/Write*" with complete parallelism of read, transform and write activities. The LDR module has an internal multi-threaded structure based on in-memory structure (buffer pools) and two kinds of specialized thread groups:

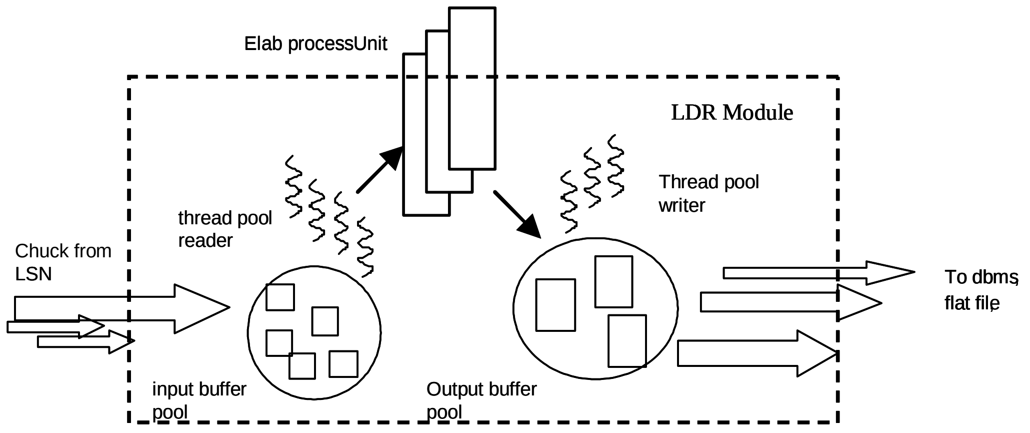
- Input Buffer Pool – memory segments used by Readers (from Readers Thread Pool) for storing input data;
- Output Buffer Pool – memory segments used by ProcessUnits to write transformed data ready to be loaded by Writers (from Writers Thread Pool);
- Reader Thread Pool - threads that open the input data files or chunks, once that the appropriate message has been detected from the Listener, and spread that data over the Input Buffer Pool;
- Writer Thread Pool – threads that move transformed data from Output Buffer Pool to database tables (in direct or conventional path) or flat files.

The ProcessUnit (thread) implements the transformation logic from over Input Buffer Pool data and move transformed data to Output Buffer Pool. DWPP offers lot of pre-defined functions/macros useful for transformation logic. Tipically, in this stage the input block is analyzed field by field, operates the key valorisation and other transformations and compose the row in the output block.

The sharing of the Input and Output Buffer Pool between different kind of ProcessUnits is managed by LDR Module structure: Input Channel – logical flow of data that comes in the ProcessUnit and Output Channel – logical flow of data that comes out of the ProcessUnit. One ProcessUnit can have one Input Channel and multiple Output Channels.

This kind of partitioning and pipelining of ETL activities, based on threads instead of processes and chunks instead of records, is very performing and all synchronization activities is done by LDR infrastructure.

LDR Module offer API interface for defining Buffer Pools, Channels and ProcessUnits.



2.5 DBA Module – Administration

In data warehouse environment “history” and “time” are unavoidable factors to consider at the Data Warehouse design time. Incremental loading, scheduled on a regular basis, into a target table that cumulates historical data (usually fact table) is typical situation in Data Warehouse environment. Data partitioning is therefore a key Data Warehouse design issue.

The DBA Module provide APIs for defining different type of partitioned tables, related indexes, time period that you have to keep the data in Data Warehouse and it is based on some strict naming convention. The DBA module provide useful API interface for “managing history” and common related DBA tasks in a Data Warehouse environment (index maintenance, add/drop partition, export for backup,..). DBA module also provides an original technique for merging partitions with different time slot (daily, monthly) maintaining free access and transactional consistency during whole process.

3 Case Study

The population system for Mobile Network Traffic Data Warehouse has been developed using the Data Warehouse Population Platform (DWPP) and deployed for one of the biggest mobile network operator. This Data Warehouse is aimed to store cellular network traffic data (CDRs), billing and other customer data in order to enable different kind of analysis, principally off-line fraudulent behavior analysis.

The Data Warehouse Population System elaborates, day by day, the entire mobile network traffic, customer data and data coming from billing system and interacts with on-line Fraud Detection and Management System. The Data Warehouse population system is able to transform and load about 80 million CDRs (about 20 columns/row) per hour for the entire process (ftp of files, uncompress, transformation and loading).

The Data Warehouse, based on Oracle8i, is made up of different federated logical Data Marts. There are two different sub-systems related to different customers segments: subscribers and pre-paid cards. The entire Data Warehouse is about 2 TB, 0.5 TB related to subscribers and 1.5 TB related to pre-paid cards. The database server is Oracle8i Enterprise Server on a Digital Unix Alpha Server GS160 (12 cpu's 567 MHz, 16Gb Ram). The system, during population and recovery is always "ready to analysis".

The Data Warehouse are organized in star schemas with dimensional and fact tables. The principal dimension tables are: Customer, Location, Direction, CDR Type, Time. The fact tables contain traffic detail data. All fact tables are partitioned by value and hash sub-partitioned due to:

- “managing history” needs - every day you have to add new partition and drop the oldest; you have to export for backup needs only the just loaded partition;
- performance needs - Parallel Query Servers work on different sub-partitions.

The main fact table, containing traffic detail data of all customers, has about 3 Billion of records. The Customer dimension table is also hash partitioned due to volumes of data (about 26 Million of Customers).

Data stored in the Data Warehouse are accessed by different kind of analytical tools in order to monitor traffic trends (risk directions) and discover patterns that describe or predict fraudulent behaviors. The OLAP tool is Oracle Discoverer; the Data Mining tool is SAS Enterprise Miner. Oracle Developer has been used to develop the “ad-hoc query” client application used for browsing detail data about customer.

4 Conclusions

DWPP was built to meet primarily the following objectives:

- user requirements in ETL are often characterized by specific needs, environments, particular conditions: so needs great adaptability and flexibility to meets this requirements;
- large volumes, or strictly timing deadlines or both of them, impose the maximum attention to performances issues;
- service and maintenance of the ETL system requires simple and reliable recovery techniques.

DWPP meets these objectives; in several contexts significantly reduce effort in building systems with well organized and comprehensible application code.

References

1. Ralph Kimball, et. al., “Data Warehouse Life Cycle Toolkit”, John Willey & Sons, Inc., 1998 ISBN: 0-471-25547-5
2. Dr. Yu Gong: “Partition Exchange Loading: A new Performance feature in Oracle Warehouse Builder, Release 3i”, An Oracle Technical White Paper, May 2001: