High Speed Data Loading for Large Sized RDBMS Tables

Prabin R. Sahoo and Chetan Phalak

Tata Consultancy Services, Yantra Park, Thane,
Maharashtra, India
{prabin.sahoo,chetan1.phalak}@tcs.com

Abstract. Data loading into RDBMS is a common phenomenon over decades. Over the years RDBMS has dominated in the enterprise world. Conventional file systems are seen to be replaced with popular RDBMS such as Oracle ®, UDB ®, and Sybase ® etc. This has necessitated the need for data loading while migrating from file systems to RDBMS. In some cases data loading is an everyday activity, for example, dataware housing projects involving data gathering, mining and analysis. No matter what type of data loading it may be, the goal is common i.e. how to achieve high speed in data loading for large sized table to minimize the down time. In this paper we are demonstrating how we can achieve high speed data loading into large RDBMS tables.

Keywords: RDBMS, Loader, Partitions, Parallelism, Queries, Performance, sqlldr.

1 Introduction

High speed data loading is required to reduce the down time of business operations. While data loading involves a lot of preparation such as data gathering, transformations, mining which requires substantial time but in this paper we will be discussing on the data loading activity only. Our focus is to achieve high speed data loading for large sized tables. We assume that data is ready for our loading procedure to take off. We have used Oracle 10g ® as our target database and source is a comma separated data file. We are also using Oracle sqlldr utility [1, 2, 4] for our data loading activity. This utility provides several options as command line parameters for speed up of data loading. Parallel loading is one of the features for high speed data loading. In general when a table is loaded in parallel, DBMS may try to load the data from the loaders synchronizing with lock mechanism especially when the table is not partitioned. This can affect the performance of loading. We have demonstrated in this paper how we can achieve high speed data loading using parallelism.

2 Literature Review

High speed data loading into RDBMS using sqlldr [2] is not much being discussed directly. In [3] the authors have discussed about data load procedure for DB2 database

V.V. Das, E. Ariwa, and S.B. Rahayu (Eds.): SPIT 2011, LNICST 62, pp. 282–286, 2012.

and have cited the reason for using alternative methods for load and unload. In addition the authors have mentioned that the regular migration methods cannot be used "*when a large amount of data needs to be transferred from one system to the target system, when there is a big release differences between them and when time is limiting factor*" [3]. We have referred this article as we are also using standard load utility of Oracle and our aim is to load large database tables. However, our implementation approach is different and our focus is on the high speed data loading to reduce the down time. In [5] the author mentions various options for data loading. We have used the SQL* Loader with direct path option in our experiment. The author mentions "*For all of its awkwardness, SQL*Loader still seems to be about the fastest and most efficient way to get flat file data into Oracle. By default, SQL*Loader uses what it calls conventional path loading–bulk inserts, basically. The performance is not phenomenal, and there are faster alternatives. However, with a simple "direct=true" on the command line, you can invoke "direct path" loading. In a direct path load, SQL*Loader writes rows directly into new data blocks above the table's high water mark"* [5]. Though direct option is efficient way to go, but using this option is not pretty straight forward for large files specially while loading into a table for optimal loading. We have shown in our model how to use it effectively.

3 Model and Architecture

In our model we are assuming our input is having comma separated text file and each table in the target database has one text file. In Fig.1 "F" represents the file corresponding to a table in the target database. For parallel loading into a table we need the following conditions to be true in our high speed load model; a) Table structure should be such that loaders should be able to load concurrently these files. If tables are not large, there is no need to go for partitions. However, in this paper we are dealing with large file, so we are discussing about partitioning of tables [9]. Oracle provides partition option in the loader control file [8]. We assume that loading into partitioned table reduces the locking possibilities which can improve more parallelism in data loading. We are using range partition feature in our model. We do not claim on any other types of partitions. We further assume that as per reference [10] Oracle allows more than one partition loading at a time, and table size is large enough to qualify for partitions and direct option is applicable. b) Split files (F2, F3) which can be loaded in parallel. The table level file "F" can be split into multiple files. The split is not just straight forward. The split is based on a rule. For example if we are dealing with employee data, we can split the big file "F" into smaller files with a range of employees, employee staring with employee number 1 to 10000 in one file F1, 10001 to 20000 in F2, 20001 to 30000 in F3 and so on. It is based on the number of partitions [6] required for a table for optimal query performance with respect to the business need. However our focus in this paper is on data loading. So as a thumb rule we will split the file as per the number of partitions. If 5 partitions required for a table, then 5 split files are required. This we call the partition level of split, and for simplicity we would refer these as partitioned file. c) Addressing the Memory