Parallel Efficient Data Loading

Ricardo Jiménez-Peris[1], Francisco Ballesteros[2], Ainhoa Azqueta[3], Pavlos Kranas[1],
Diego Burgos[1] and Patricio Martínez[1]
[1]*LeanXcale, Campus de Montegancedo, Madrid, Spain*
[2]*Universidad Rey Juan Carlos, Madrid, Spain*
[3]*Universidad Politécnica de Madrid, Spain*

Keywords: Loading, Extract-Transform-Load (ETL), Scalable Databases, NUMA Architectures, Database Appliance, Scalable Transactional Management.

Abstract: In this paper we discuss how we architected and developed a parallel data loader for LeanXcale database. The loader is characterized for its efficiency and parallelism. LeanXcale can scale up and scale out to very large numbers and loading data in the traditional way it is not exploiting its full potential in terms of the loading rate it can reach. For this reason, we have created a parallel loader that can reach the maximum insertion rate LeanXcale can handle. LeanXcale also exhibits a dual interface, key-value and SQL, that has been exploited by the parallel loader. Basically, the loading leverages the key-value API and results in a highly efficient process that avoids the overhead of SQL processing. Finally, in order to guarantee the parallelism we have developed a data sampler that samples data to generate a histogram of data distribution and use it to pre-split the regions across LeanXcale instances to guarantee that all instances get an even amount of data during loading, thus guaranteeing the peak processing loading capability of the deployment.

SCITEPRESS
SCIENCE AND TECHNOLOGY PUBLICATIONS

1 INTRODUCTION

In this paper we discuss the parallel data loader architected and developed for the LeanXcale database. This data loader was motivated by the LeanXcale appliance being developed in cooperation with Bull-Atos in the Bull Sequana in the context of the CloudDBAppliance European project (CloudDBAppliance 2019). The Bull Sequana is a large parallel server than can reach 896 cores and 140 TB of main memory.

Extract-Transform-Load (ETL) process are commonly used at all enterprises and they play a key role in moving data across the different data management systems being used within the enterprise. One of the key performance issues is the speed of loading of the destination database.

Loading data in LeanXcale appliance adopting the traditional way, by means of a sequential thread, does not actually exploit the potential of the platform, since the thread doing the loading process become the bottleneck. Despite the LeanXcale appliance is able to load many millions of records per second, using a single loading thread results is simply using a small fraction of the insertion capacity of the appliance.

The loader addresses three key issues in the loading process:

1. The parallelism of the loader process that is required to exploit the full capacity of the appliance.
2. The efficiency of the process. Loading data does not have the same requirements as regular OLTP processing, and these lower requirements can be exploited to improve the efficiency of the process.
3. The parallelism on the database server side. In order to fully exploit the parallelism on the server side it is necessary to guarantee that all servers will receive a fraction of the load.

The first point have been addressed by creating a parallel loader that is multi-process and multi-threaded. This enables to use as many machines and cores as required to load data at the maximum rate that the LeanXcale appliance can process.

The second item has been addressed by exploiting the dual-interface provided by LeanXcale. The dual-interface supports key-value and SQL APIs. Both interfaces work over the same relational data. Since ingesting data does not require SQL, the loader leverages the key-value interface to save all the SQL overhead to ingest the data.

Jiménez-Peris, R., Ballesteros, F., Azqueta, A., Kranas, P., Burgos, D. and Martínez, P.
Parallel Efficient Data Loading.
DOI: 10.5220/0008318904650469
In *Proceedings of the 8th International Conference on Data Science, Technology and Applications (DATA 2019)*, pages 465-469
ISBN: 978-989-758-377-3
Copyright © 2019 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

The third bullet has been tackled by creating a data sampler that creates a histogram of the data to partition horizontally data across the servers before the loading process starts in order to guarantee that all servers will receive a similar amount of data, thus, reaching the maximum capacity of the appliance.

2 LeanXcale ARCHITECTURE

2.1 Layers

LeanXcale (LeanXcale 2019) is an ultra-scalable parallel & distributed database manager. It consists of three layers: storage engine, transactional manager, and query engine. The storage engine is actually a parallel-distributed relational key-value data store. The transactional manager is a parallel-distributed system with several components. The query engine is parallel-distributed as well and can scale both OLTP (each instance handles a subset of the transactions) and OLAP workloads (multiple instances cooperate to execute a large analytical query).

2.2 KiVi Storage Manager

KiVi is a parallel-distributed storage manager. One of its differential features is that it is optimized to run efficiently on many-core and NUMA architectures (Ricardo Jiménez-Peris, 2019). Basically, a different KiVi server is deployed at each of the cores that are dedicated to the storage layer.

Each table is horizontally partitioned into regions. Each region comprises a range of primary keys. The region is the distribution unit across servers. When a row is inserted it will hit the server managing the region where the row belongs (based on the primary key range of the horizontal partition).

Client applications access LeanXcale database through the SQL interface, via the JDBC driver. Internally, KiVi is accessed by the query engine subsystem. KiVi offers a key-value API. This API is internally used by the query engine to interact with the storage layer. However, this API is also available to be used directly by client applications. In this way, LeanXcale database offers a dual-interface, key-value and SQL.

This dual interface has the advantage that whenever it is convenient it becomes possible to avoid the overhead of SQL processing by directly accessing the key-value interface that is accesing the same relational data as SQL.

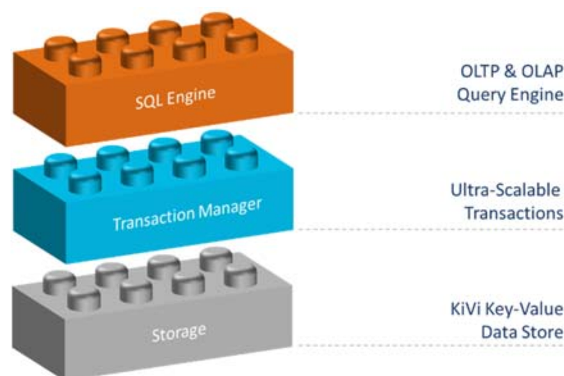

Figure 1: LeanXcale subsystems.

3 LeanXcale ARCHITECTURE

3.1 What Is LeanXcale Database

LeanXcale is an ultra-scalable operational Full SQL Full ACID distributed database (Ozsu and Valduriez, 2014) with analytical capabilities. The database system consists of three subsystems:

1. KiVi Storage Engine.
2. Transactional Engine.
3. SQL query Engine.

3.2 LeanXcale Subsystems

The operational database is a quite complex system in terms of different kinds of components. The operational database consists of a set of subsystems namely: the Query Engine (QE), the Transactional Manager (TM), the Storage Engine (SE) and the Manager (MG). Some subsystems are homogeneous and other heterogeneous.

Homogeneous subsystems have all instances of the same kind of role. Heterogenous subsystems have different roles. Each role can have a single instance or multiple instances. The transactional manager has the following roles: Commit Sequencer (CS), Snapshot Server (SS), Conflict Managers (CMs) and Loggers (LGs). The former two are mono-instance, whilst the latter two are multi-instance. The Storage Engine has two roles data server (DM) and meta-data server (MS), both multiple instances. The query engine is homogeneous and multi-instance. There is a manager (MNG) that is single instance and single-threaded. Many of these components can be replicated to provide high availability, but their nature does not change. Since replication it is an orthogonal topic, we do not mention anymore.

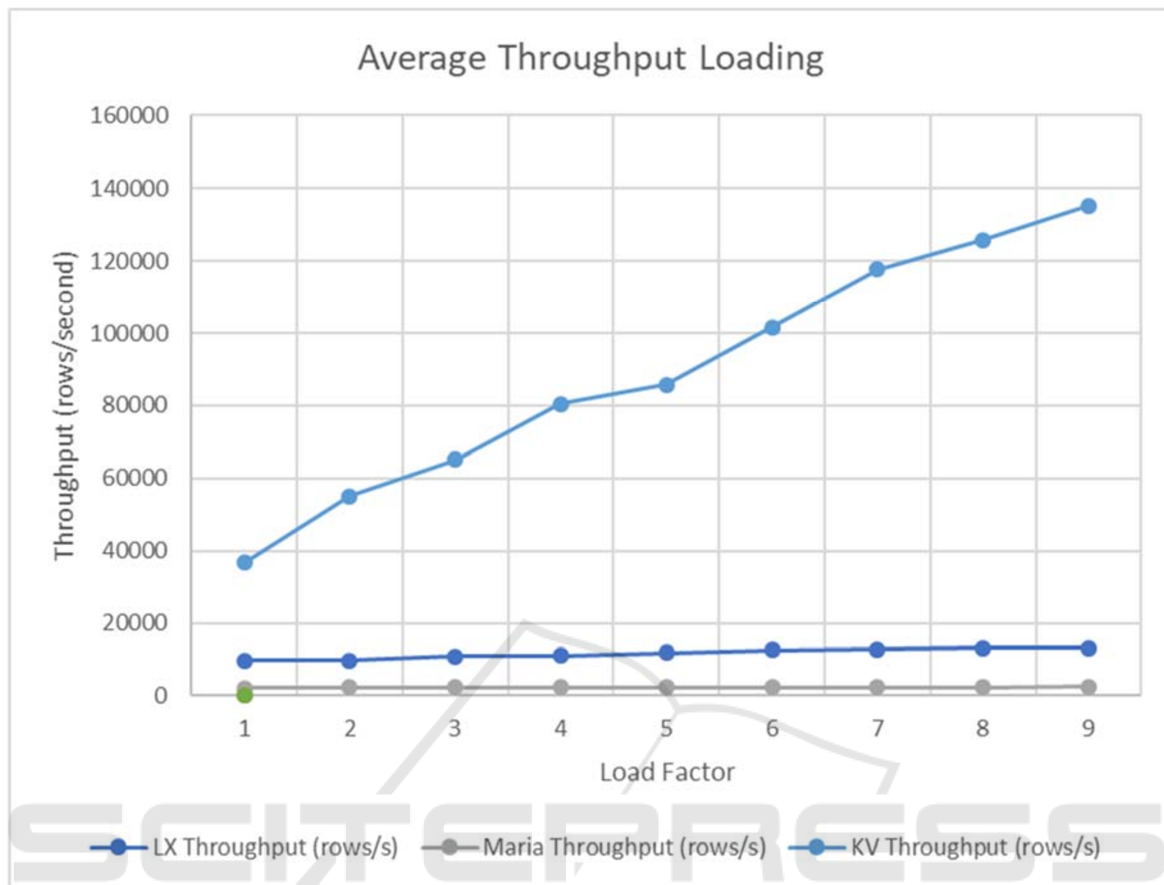

Figure 2: Loading Throughput.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

4 THE PARALLEL LOADER

As aforementioned it does not matter how powerful is a database or data store. Loading has to be done carefully to attain its maximum loading capacity.

The first aspect to be taken into account is the parallelism in the loader itself. Assume, there is a database able to handle 1 million insertions per second. Assume that the average latency of the insertion at this maximum throughput is 1 ms. If the loader is single threaded, it would do 1000 insertions per second. Why? Because it is limited by the latency of a single insertion that does not exploit the parallelism in the database. To reach the maximum throughput we will need 1,000 threads inserting data. A single computer might not be enough to generate this load what might require having the threads spread across multiple processes running on different computers.

This is why a loader should be parallel and this is key for the speed of the loading process, that is, to reach the maximum speed of loading.

However, the parallelism of the loader on the client side it is not enough. Despite the database server might be parallel or distributed, to reach its maximum potential data should be split evenly across the different server instances of the database server. This partitioning of data across server instances is not easy, especially because it has to guarantee balancing across them.

This even partitioning requires to know about the data distribution. Doing it accurately guarantees a perfect split of data across servers. However, the accurate splitting is too expensive. For this reason, we have developed a data sampler. The data sampler samples a subset of the data. The number of samples that are needed depend on the dataset to be loaded. We have run a number of experiments with different dataset sizes, different number of regions and different number of samples. We have measured the resulting imbalance across regions. We have seen that basically by multiplying by 10 the number of samples, the inaccuracy can be reduced by around 3 times. For a dataset of up to 600 million rows, a sam-

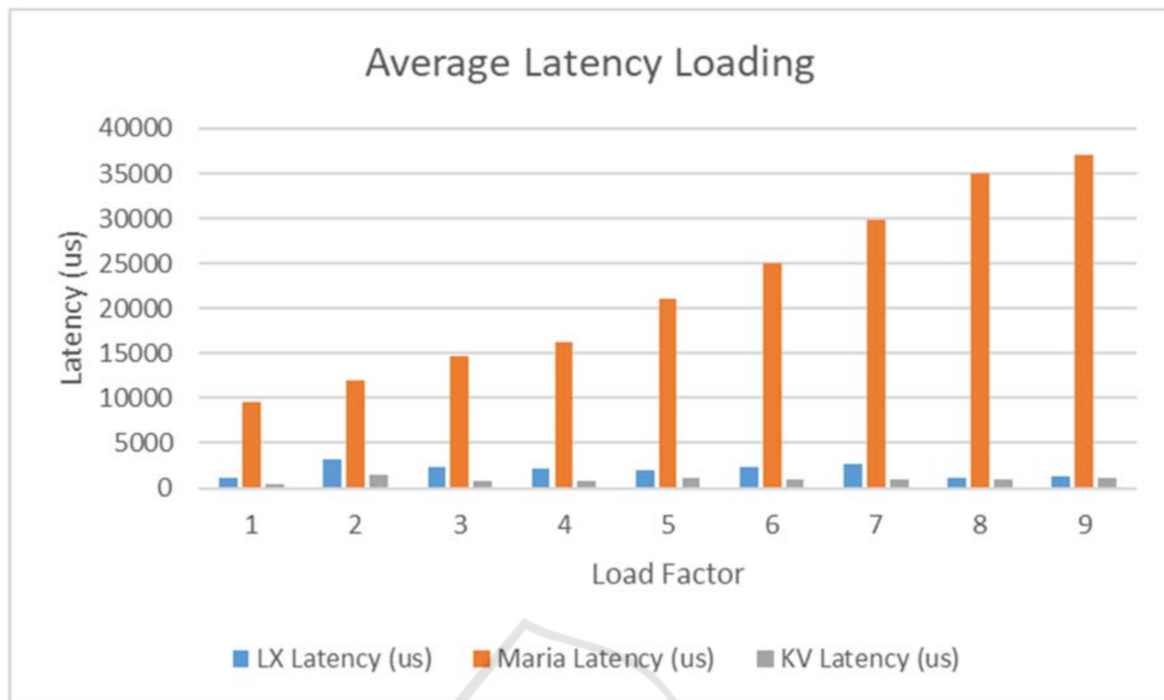

Figure 3.

ple of 10,000 rows provides a fairly good accuracy (Azqueta-Alzuaz, 2017).

After running the sampling process over the dataset to be loaded, then split points are set on the table that establish how data will be horizontally partitioned across the different servers.

5 EXPERIMENTAL EVALUATION

We have evaluated the parallel loader with LeanXcale database through its two interfaces, SQL and key-value. We compare the results with the parallel loading over a well-known open source database, MariaDB.

The evaluation has been performed over an INTEL quad computer with 8 GB of memory and 1 SSD.

The injection is made through the parallel loader with a load factor. The bigger the load factor, the higher the number of threads/processes in the parallel loader ingesting data into the database.

The results are shown in Figure 1 and Figure 2. Figure 1 shows the evolution of throughput for an increasing loading factor. MariaDB was able to reach a maximum of 2,628 rows/second. LeanXcale with the SQL/JDBC interface (LX) was able to reach 13,062 rows/second that is almost 5 times higher throughput. Interestingly the key-value interface (KV) reached a throughput of 135,220 rows/second, what means it was doing 10 times more than LeanXcale through the SQL interface and more than 50 times what MariaDB was able to do.

Figure 2 shows the results for latency. MariaDB has the poorest results, with a latency of 37 ms for the highest throughput it achieved. LeanXcale SQL interface was attained a latency of 1.3 ms under the maximum throughput. While LeanXcale KiVi interface had a latency of 1 ms for the maximum throughput figure.

6 CONCLUSIONS

The more performant and the scalable a database, the more careful it requires to perform the loading process. Otherwise, most of its capacity will be wasted.

In this paper, we have presented a parallel loader that is able to exercise the maximum throughput of LeanXcale database. It does so by having parallelism on the loader multi-process and multi-threaded to be able to inject data at the maximum rate that LeanXcale database can handle.

Secondly, a sampler has been developed that ena-

bles to set a priori the split points of the tables to be loaded to guarantee that the insertion load is distributed evenly across all server instances of LeanXcale.

Thirdly, the key-value of LeanXcale storage layer, KiVi, it is exercised to avoid the overhead of SQL processing.

A evaluation has been conducted and it has been found that the LeanXcale key-value interface is able to load data 10 times faster than the LeanXcale SQL database. Interestingly, it was able to load data 50 times faster than MariaDB.

ACKNOWLEDGEMENTS

This work has been partially funded by the European Commission under the H2020 project: CloudDBAppliance – European Cloud In-Memory Database Appliance with Predictable Performance for Critical Applications. Project number: 732051.

REFERENCES

Ainhoa Azqueta-Alzuaz, M. P.-M.-P. (2017). Massive Data Load on Distributed Database Systems over HBase. *CCGRID Proceedings*.

Ricardo Jiménez-Peris, F. B. (2019). NUMA-Aware Deployments for LeanXcale Database Appliance. *CLOSER Workshop*.

Özsu, T., P. Valduriez. *Distributed and Parallel Database Systems*. Computing Handbook, 3rd ed. 2014.

LeanXcale. http://leanxcale.com. 2019

CloudDBAppliance. https://clouddb.eu/ 2019.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS