Big-ETL: Extracting-Transforming-Loading Approach for Big Data

M. Bala[1], **O. Boussaid**[2], **and Z. Alimazighi**[3]
[1]Department of informatics, Saad Dahleb University, Blida 1, Blida, Algeria
[2]Department of informatics and Statistics, University of Lyon 2, Lyon, France
[3]Department of informatics, USTHB, Algiers, Algeria

Abstract— *ETL process (Extracting-Transforming-Loading) is responsible for (E)xtracting data from heterogeneous sources, (T)ransforming and finally (L)oading them into a data warehouse (DW). Nowadays, Internet and Web 2.0 are generating data at an increasing rate, and therefore put the information systems (IS) face to the challenge of big data. Data integration systems and ETL, in particular, should be revisited and adapted and the well-known solution is based on the data distribution and the parallel/distributed processing. Among all the dimensions defining the complexity of the big data, we focus in this paper on its excessive "volume" in order to ensure good performance for ETL processes. In this context, we propose an original approach called Big-ETL (ETL Approach for Big Data) in which we define ETL functionalities that can be run easily on a cluster of computers with MapReduce (MR) paradigm. Big-ETL allows, thereby, parallelizing/distributing ETL at two levels: (i) the ETL process level (coarse granularity level), and (ii) the functionality level (fine level); this allows improving further the ETL performance.*

Keywords: Data Warehousing, Extracting-Transforming-Loading, Parallel/distributed processing, Big Data, MapReduce.

1. Introduction

The widespread use of internet, web 2.0, social networks, and digital sensors produce non-traditional data volumes. Indeed, MapReduce (MR) jobs run continuously on Google clusters and deal over twenty Petabytes of data per day [1]. This data explosion is an opportunity for the emergence of new business applications such as Big Data Analytics (BDA); but it is, at the same time, a problem given the limited capabilities of machines and traditional applications. These large data are called now "big data" and are characterized by the four "V" [2]: *Volume* that implies the amount of data going beyond the usual units, the *Velocity* means the speed with which this data is generated and should be processed, *Variety* is defined as the diversity of formats and structures, and *Veracity* relates to data accuracy and reliability. Furthermore, new paradigms emerged such as *Cloud Computing* [3] and *MapReduce* (MR) [4]. In addition, novel data models are proposed for very large data storage such as *NoSQL* (Not Only SQL) [5]. This paper aims to provide solutions to the problems caused by the big data in a decision-support environment. We are particularly interested in the very large data integration in a data warehouse. We propose a parallel/distributed ETL approach, called Big-ETL (ETL Approach for Big Data), consisting of a set of MR-based ETL functionalities. The solution offered by the research community, in this context, is to distribute the ETL process on a cluster of computers. Each ETL process instance handles a partition of data source in parallel way to improve the performance of the ETL. This solution is defined only at a process level (coarse granularity level) and does not consider the ETL functionalities (fine granularity level) which allows understanding deeply the ETL complexity and improve, therefore, significantly the ETL process. To the best of our knowledge, Big-ETL is a different and original approach in the data integration field. We first define an ETL process at a very fine level by parallelizing/distributing its core functionalities according to the MR paradigm. Big-ETL allows, thereby, parallelization/distribution of the ETL at two levels: (i) ETL functionality level, and (ii) ETL process level; this will improve further the ETL performance facing the big data. To validate our Big-ETL approach, we developed a prototype and conducted some experiments.

The rest of this paper is structured as follows. Section 2 presents a state of the art in the ETL field followed by a classification of ETL approaches proposed in the literature according to the parallelization criteria. Section 3 is devoted to our Big-ETL approach. We present in Section 4 our prototypical implementation and the conducted experiments. We conclude and present our future work in Section 5.

2. Related work

One of the first contributions on the ETL field is [6]. It is a modeling approach based on a non-standard graphical formalism where ARKTOS II is the implemented framework. It is the first contribution that allows modeling an ETL process with all its details at a very fine level, i.e. the *attribute*. In [7], authors proposed a more holistic modeling approach based on UML (Unified Modeling Language) but with less details on ETL process compared to [6]. Authors in [8] adopted BPMN notation (Business Process Model and Notation), a standard notation dedicated to the business process modeling. This work was followed by [9], a modeling framework based on a metamodel in MDD (Model Driven Development) architecture. [7] and [8] are top-down approaches and allow, therefore, modeling

sub-processes in their collapsed/expanded form for more readability. Authors in [10] proposed a modeling approach which consists of a summary view of the ETL process and adopt the Reo model [11]. We consider that this contribution could be interesting but is not mature enough and deserves Reo customization model to support the ETL specifics.

Following the big data emergence, some works tackled interesting issues. [12] is an approach which focuses on the performance of ETL processes dealing with large data and adopts the MapReduce paradigm. This approach is implemented in a prototype called ETLMR which is a MapReduce version of the PygramETL prototype [13]. The ETLMR platform is demonstrated in [14]. [15] shows that ETL solutions based on MapReduce frameworks, such as Apache Hadoop, are very efficient and less costly compared to ETL tools market. Recently, authors in [16] proposed CloudETL framework. CloudETL uses Apache Hadoop to parallelize ETL processes and Apache Hive to process data. Overall, experiments in [16] shows that CloudETL is faster than ETLMR and Hive for large data sets processing. [17] demonstrates the P-ETL platform. P-ETL (Parallel-ETL) is implemented under the Apache Hadoop framework and provides a simple GUI to set an ETL process and the parallel/distributed environment. In the batch version, P-ETL runs thanks to an XML file (*config.xml*) in which the same parameters should be set. In the P-ETL approach, the mappers (Map step) are in charge of standardizing the data (cleasing, filtering, converting, ...) and the reducers (Reduce step) are dedicated for merging and aggregating them. To the best of our knowledge, there are no works having tackled the ETL modeling issue intended to the big data environment and more precisely to the parallel/distributed ETL processing. We focus in this paper on the parallelization/distribution issue to improve the performance of the ETL. The classification proposed in Tab.1 is based on the parallelization criteria.

Table 1: Classification of ETL works

Approach	Purpose	Classification
[6]	Modeling	Centralized approach
[7]	Modeling	Centralized approach
[8]	Modeling	Centralized approach
[13]	Performance	Centralized approach
[12]	Performance	Distributed approach
[10]	Modeling	Centralized approach
[15]	Performance	Distributed approach
[16]	Performance	Distributed approach
[17]	Performance	Distributed approach
Big-ETL	Performance	Distributed approach

a) Centralized ETL process approach: In this paper, the ETL process approach is defined as centralized (or classical) when (i) the ETL process runs on an ETL server (one machine), (ii) in one instance (one execution at the same time), and (iii) the data are with moderate size.

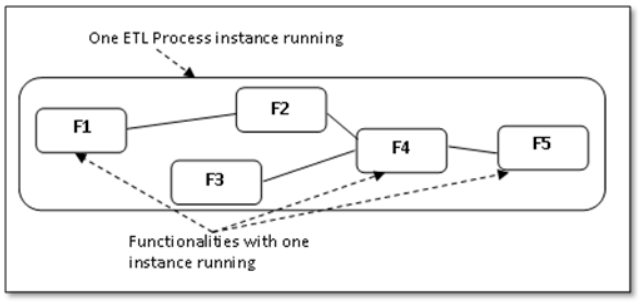

Fig. 1: Centralized ETL Process approach.

In this context, only the independent functionalities can be run in parallel way (both the ETL functions and the machine, on which it will be run, should be multithreaded). An ETL functionality, such as Changing Data Capture (CDC), Surrogate Key (SK), Slowly Changing Dimension (SCD), Surrogate Key Pipeline (SKP), is a basic function that supports a particular aspect of an ETL process. In Fig. 1, we can see that (*F1* and *F3*) or (*F2* and *F3*) can be run in parallel way.

b) Distributed ETL process approach: The well-known solution to cope with big data is the "parallelization/distribution" of the data and the ETL process on a cluster of computers. The MR paradigm, for instance, allows splitting large amounts of data sets where each partition will be subject to an instance of the ETL process.

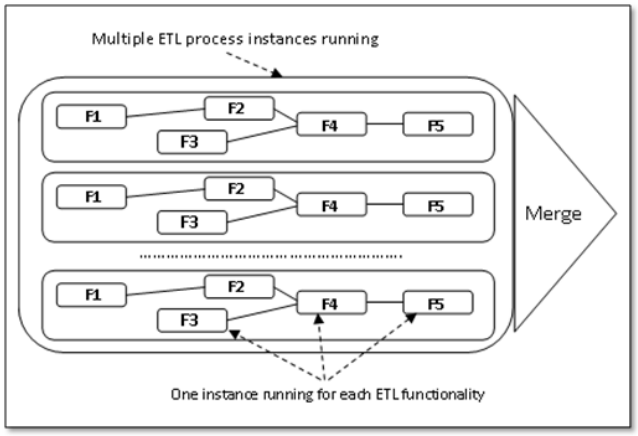

Fig. 2: Distributed ETL Process approach.

As depicted in Fig. 2, multiple ETL process instances run in parallel way where each one deals with its data partition in the Map step. The partial results produced by the mappers are merged/aggregated in the Reduce step and then loaded into the DW. All approaches proposed with MR paradigm, [12] and [15] for instance, apply the distribution

only at the process level. Big-ETL applies MR at two levels: (i) Process level (coarse granularity level), and (ii) functionality level (fine granularity level). We believe that the ETL, in the context of technological change having affected both data and processes, still presents some scientific problems such as big data modeling considering its different characteristics (volume, variety, velocity, veracity,...), data partitioning, parallel processing in its various forms (processes parallelization, process components parallelization, pipeline parallelization,...), etc. Functionalities as the core ETL functions deserve a most in-depth study to ensure, at a very fine level, robustness, reliability and optimization of the ETL process. Our Big-ETL is a parallel/distributed ETL approach based on two distribution levels (Process and functionalities) and two distribution directions (Vertical and horizontal).

3. ETL Approach for Big Data

We present in this section our Big-ETL approach. We deployed it on many ETL functionalities such as Changing Data Capture (CDC), Data Quality Validation (DVQ), Surrogate Key (SK), Slowly Changing Dimension (SCD), Surrogate Key Pipeline (SKP). Among all these ETL functionalities, we chose to present, in this paper, CDC to illustrate our Big-ETL approach.

3.1 Big-ETL principle

Our Big-ETL process is functionalities-based approach which exploits the MR paradigm. For each of these functionalities, we apply the same principle adopted at a process level in the "distributed ETL process approach" as depicted in Fig. 2.

3.1.1 Key Concepts

a) ETL functionality: In order to control the complexity of the ETL process, we define it thanks to a set of core functionalities. The ETL functionality is a basic function that supports a particular ETL aspect such as Changing Data Capture (CDC), Data Quality Validation (DQV), Surrogate Key (SK), Slowly Changing Dimension (SCD), Surrogate Key Pipeline (SKP), etc. The ETL task, however, is an instance of an ETL functionality. Let *SK1* and *SK2* be two ETL tasks that generate a surrogate key for inserting tuples in *PRODUCT* and *CUSTOMER* dimensions respectively. *SK1* and *SK2* are two different tasks but both are based on SK. Thus, the SK is the ETL functionality where SK1 and SK2 are its instances. In the follows, we describe an ETL process in terms of its functionalities.

b) Elementary process/function: When an ETL functionality is not atomic (aggregate functionality) in terms of processing, we consider that it is in charge of several separated elementary processes where each one is affected for an elementary function. An elementary process is an atomic unit of processing which is synchronized with other elementary processes to ensure the ETL functionality. Each one of the elementary processes is implemented as an elementary function. Thus, we consider that the aggregate functionality is a set of synchronized elementary functions. For example, the functionality CDC which is responsible to identify the changes (INSERT, UPDATE, DELETE) having affected the data in a particular source, can be decomposed in three elementary functions where each one is in charge of identifying INSERT, UPDATE, DELETE respectively.

3.1.2 Vertical Distribution of Functionaltilies (VDF)

As shown in Fig. 3, the ETL process runs in one instance, while each of its functionalities runs in multiple instances. For example, the functionality *F4* (oval) that runs in three instances (fragments separated by dashes), received its input data from *F2* and *F3*. These inputs are partitioned and each of the three partitions is subject to an instance of *F4* (mapper). Partial results produced by the three mappers are merged by reducers to provide the final *F4* outputs. This is a novelty in the parallel/distributed ETL approaches based on MR paradigm as all other approaches does not consider the parallelization/distribution at an ETL functionality.

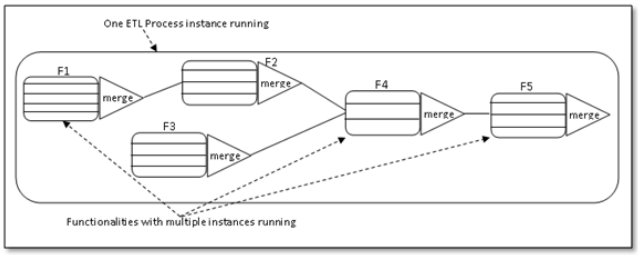

Fig. 3: VDF Approach.

3.1.3 Vertical Distribution of Functionaltilies and Process (VDFP)

In case where VDF presents low performance (particularly if the ETL process contains much sequential functionalities), the designer should set the ETL process to be run in several instances. This is an hybrid approach that takes the principles of the "distributed ETL process" and VDF approaches at the same time as shown in Fig. 4.

It should be noted that the VDFP approach requires more resources (*cluster nodes, HDD space, RAM, Cache, LAN bandwith* ...). When the ETL process runs in the VDF approach and reaches *F4*, it will require three tasks as *F4* runs in three instances. The same process executed in the VDFP approach will require thirty parallel tasks if it runs in ten instances in addition to the three instances of *F4*.

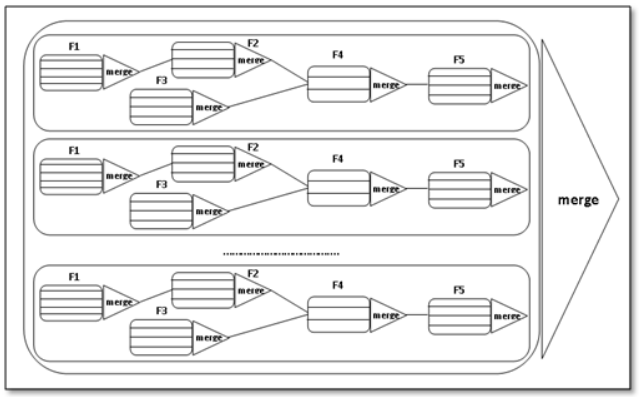

Fig. 4: VDFP Approach.

3.1.4 Horizontal Distribution of Functionaltilies (HDF)

Some ETL functionalities operate several elementary processes on source data. In this case, these functionalities are not atomic and can thereby be decomposed into elementary functions where each one is in charge of a particular process unit. Let F be a functionality in an ETL process which operates some elementary processes units $T_1, T_2, ..., T_n$ on the source data.

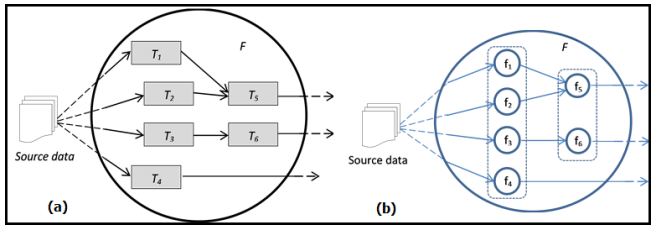

Fig. 5: Elementary processes (a) and functionalities (b).

We can decompose F into elementary functions noted $f_1, f_2, ..., f_n$ where each f_i is in charge of T_i. FIG. 5 (a) shows an ETL functionaly F which operates six elementary processes $T_1, T_2, ..., T_6$. We note that T_1, T_2, T_3, T_4 can be run in parallel way since no dependencies exist between them. In the same way, T_5 and T_6 can be, also, run in parallel. Thus, we can decompose F into six elementary functions noted $f_1, f_2, ..., f_6$ which are in charge of $T_1, T_2, ..., T_6$ respectively (FIG. 5 (b)). Unlike the VDF approach which distributes the ETL functionality by instanciation, the HDF approach distributes the ETL functionality by fragmentation. In a distributed environment, the schema depicted in FIG. 5 (b) allows, in a first phase, distributing F in four fragments f_1, f_2, f_3 and f_4 which run in parallel way. In the second phase, F is distributed into f_4 and f_5 that can be run in parallel way and allow providing the final output of F.

3.1.5 Pipeline Processing Distribution (PPD)

Some ETL functionalities process the source data tuple-by-tuple in a sequential way called pipeline processing. Since all the tuples pass by the pipeline, we propose a synchronization schema in order to process a subset of tuples in parallel way. The number of tuples present in the pipeline should be equal to the number of functionalities. Indeed, when a particular tuple is being processed by the last functionality in the pipe, its successors should be also processed according to the order of the functionalities as defined in the pipe. Let P be a pipe in which is defined a set of functionalities F_1, F_2, ..., F_n.

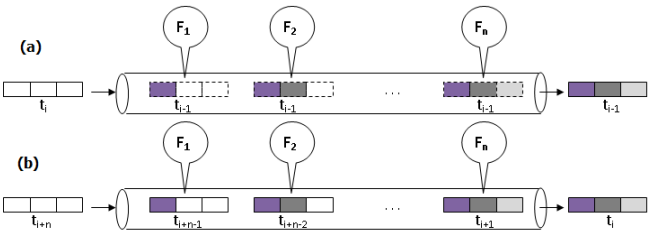

Fig. 6: Sequential (a) and parallel (b) pipeline.

When the tuples of data $t_1, t_2, ..., t_m$ should pass by a sequential pipe P, the tuple t_i is moved in the pipe P when the tuple t_{i-1} is completely processed by all the functionalities F_1, F_2, ..., F_n (FIG. 6 (a)). Thus, only one tuple can be present in the pipe at the same time. In order to improve further the performance of the ETL process, we propose to parallelize the pipe. In this way, several tuples can be processed simultaneously in the pipe where each one is handled by a functionality according to the order defined in the pipe. Thus, when the tuple t_i is being processed by F_n, the tuple t_{i+1} is processed, in the same time, by F_{n-1}, the tuple t_{i+2} is processed by F_{n-2} and so on. In this way, the number of tuples being processed in the pipe is equal to the number of functionalities defined in the pipe (equal to n). Indeed, when a tuple is moved out the pipe after a complete process, another tuple (first in the queue) is moved in and so on until a complete process of the data partition. FIG. 6 (b) depicts the pipe P in the parallel approach.

3.2 Changing Data Capture (CDC) in Big-ETL approach

Our Big-ETL approach is applied on many ETL functionalities such as CDC, SCD, SKP, etc. Seeing the paper space constraint, we illustrate Big-ETL with the CDC functionality. The ETL functionality CDC is considered as the main functionality in the E step of ETL. It identifies the data affected by changes (INSERT, UPDATE, DELETE) in the source systems. These latter is then extracted and processed for the DW refresh [18]. The rest of data (unaffected by changes) is rejected since it is already loaded in the DW. The most common technique used in this field is based on snapshots [18]. In the classical algorithms of CDC, the changes between two corresponding tuples are detected by comparing them attribute-by-attribute. Furthermore, tuples

contain hundreds of attributes in data warehousing systems. In order to improve the CDC performance and make its cost lower, we adapted the well-known hash function CRC (Cyclic Redundancy Check) which is widely used in digital data transmission field [19] and internet applications [20]. We adapted CRC function in the CDC context as follows. Let *tuple1* and *tuple2* be two tuples stored in *ST* and *STpv* respectively. If *tuple1* and *tuple2* satisfy the two equations 1 and 2, it means that they are similar. In this case, the tuple1 will be rejected by the CDC process as no changes have occurred. However, if only the equation 1 is satisfied, it means that *tuple1* has been affected by changes and will be extracted by the CDC process as UPDATE.

$$tuple1.KEY = tuple2.KEY \quad (1)$$

$$CRC(tuple1) = CRC(tuple2) \quad (2)$$

To propose a CDC schema in the Big-ETL environment, we consider that both *ST* and *STpv* tables contain large data, the CDC functionality will run on a cluster of computers, and we adopt the MR paradigm. The classical scheme of CDC will be supplemented by new aspects namely (i) data partitioning, (ii) lookup tables, (iii) insert and update data capture process and (iv) delete data capture process.

3.2.1 Data partitioning

To deal with large data, we adopt the rule of "divide and conquer". In the context of CDC, the system should, firstly, sort *ST* and *STpv* on the column KEY, and then split them to obtain usual volumes of data. The partitioning of *ST* allows processing the generated partitions in parallel way. *STpv* is partitioned to avoid searching *ST* tuples in a large volume of data.

3.2.2 Lookup tables

To avoid searching a tuple in all *ST* and *STpv* partitions, we use Lookup tables denoted *LookupST* and *LookupSTpv* respectively. They identify the partition that will contain a given tuple. Here, are some details on the use of lookup tables:

- *LookupST* and *LookupSTpv* contain the min and max values of keys (*#KEY*) for each *ST* and *STpv* partitions respectively;
- For a T_i tuple in *ST*, it consists of searching the $Pstpv_k$ partition of *STpv* that satisfies the expression 3 in *LookupSTpv*;
- For a T_j tuple in *STpv*, it consists of searching the Pst_k partition of *ST* that satisfies the expression 4 in *LookupST*.

$$LookupSTpv.KEYmin \leq T_i.KEY \leq LookupSTpv.KEYmax \quad (3)$$

$$LookupST.KEYmin \leq T_j.KEY \leq LookupST.KEYmax \quad (4)$$

3.2.3 INSERT-UPDATE data capture (IUDCP) and DELETE data capture (DDCP) Processes

We propose two parallel processes in the new CDC scheme that support (i) INSERT and UPDATE data capture (*IUDCP*), and (ii) DELETE data capture (*DDCP*).

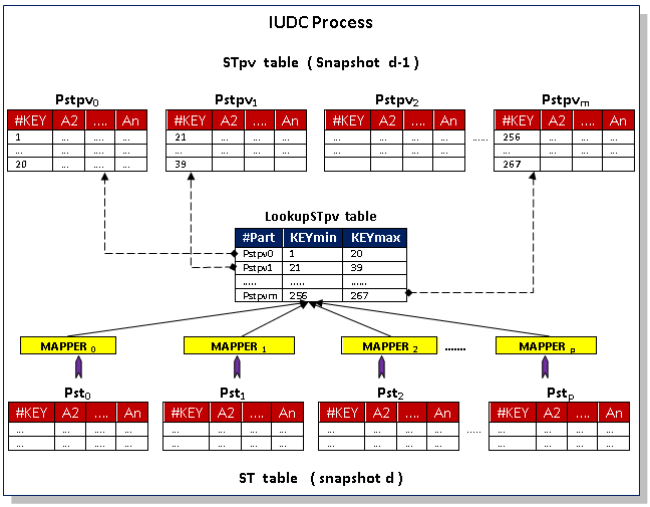

Fig. 7: IUDC process architecture.

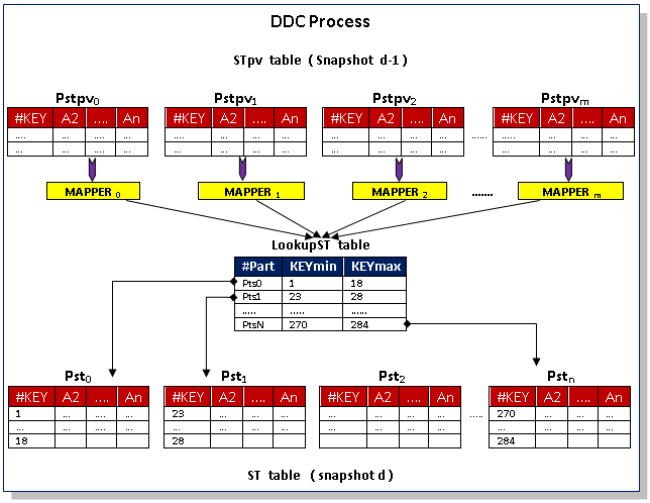

Fig. 8: DDC process architecture.

Each process runs into multiple parallel instances. Each instance of *IUDCP* and *DDCP* handles *ST* and *STpv* partition respectively. Fig. 7 depicts *IUDCP*. Each partition Pst_i is assigned to a Map_i task which is responsible for checking the existence of each its partition tuples in *STpv*. To this end, the mapper looks up in *LookupSTpv* the partition $Pstpv_k$ that may contain the tuple. Once the $Pstpv_k$ partition is identified, three cases can arise: (1) #KEY value is nonexistent in $Pstpv_k$; this means an insertion (INSERT), (2) #KEY value exists and identifies a similar copy of the tuple in $Pstpv_k$; the tuple is rejected (3) #KEY value exists in $Pstpv_k$ with

change in at least one attribute between the two tuples; this is a modification (UPDATE).

Algorithm 1 IU_MAP(Pst)

Input: Pst, LookupSTpv, tuple1: ST record, tuple2: STpv record
Output: CHANGES

```
1:  while not eof (Pst) do
2:    read(Pst, tuple1)
3:    Pstpv ← lookup(LookupSTpv, tuple1.KEY);
4:    if found() then
5:      tuple2 ← lookup(Pstpv, tuple1.KEY);
6:      if found() then
7:        if CRC(tuple1) ≠ CRC(tuple2) then
8:          extracting tuple1 as UPDATE;
9:        end if
10:     else
11:       extracting tuple1 as INSERT;
12:     end if
13:   else
14:     extracting tuple1 as INSERT;
15:   end if
16: end while
17: return (CHANGES);
```

As shown in Fig. 8, DDCP operates on the same principle as IUDCP but in the opposite direction. In DDCP, we focus exclusively on the case where the tuple does not exist (DELETE). In order to have an approach about how to process the mixing of these multiple operations, we propose a main program of CDC called *CDC_BigData*. At this level, the *ST* and *STpv* tables are sorted and then partitioned, the *LookupST* and *LookupSTpv* tables are generated respectively from *ST* and *STpv*, and finally the parallel *IUDCP* and *DDCP* processes are invoked. Algorithm 1 is responsible for capturing insertions and updates in *ST* table. A Pst_i partition will be processed by an instance of iu_map () function. Line 3 looks up in $LookupSTpv$ for a $Pstpv$ partition which may contain the tuple readed in line 2. Lines 4-12 describe the case where the $Pstpv_k$ is located. Line 5 looks-up in Pstpv the tuple. Lines 6-9 treat the case of tuple affected by changes (UPDATE) by invoking CRC hash function. Lines 10-12 treat the case where the tuple does not exist in the partition $Pstpv_k$ and is thereby captured as an insert. Lines 13-15 treat the case where the tuple does not match with any partition in the lookup table $LookupSTpv$ and thereby it is captured as an insert.

4. Implementation and experiment

We developed an ETL platform called P-ETL (Parallel-ETL) which provides: (i) data distribution, and (ii) parallel and distributed ETL processing. P-ETL is implemented in Apache Hadoop environment and uses mainly two modules (1) HDFS for distributed store and high-throughput access to application data, and (2) MapReduce for parallel processing. We defined two levels for our experiments: (i) ETL process level (coarse granularity level) and (ii) ETL Functionality level (fine granularity level). We present in this section the results for the first scenario. To evaluate our P-ETL platform, we proposed an ETL process example applied on students' data gathered at the Education Ministry. The data source contains student identifier (*St_id*), his enrollment date (*Enr_Date*), his cycle (*Bachelor, Master or Ph.D.*), his specialty (*medicine, biology, computer , ...*) and finally we find information about *scholarship* (if the student received *scholarship* or not) and about *sport* (if he practices *sport* or not). We developed a program to generate csv source data. In this experiment, we generated 7 samples of source data that vary between $244 * 10^6$ and $2,44 * 10^9$ tuples where each one has 44 bytes of size. The ETL process configured to process the data is as follows. The first task is *projection* which restricts the source tuples to an attributes subset by excluding *Scholarship* and *Sport*. The process presents in the second task a *restriction* which filters tuples and rejects those having *Null* value in *Enr_Date*, *Cycle*, and *Specialty*. The third task in the process is *GetDate()* which retrieves year from *Enr_Date*. The last task is the aggregation function *COUNT()* which computes the number of students grouped by *enrolment year*, *Cycle* and *Specialty*.

We considered the P-ETL scalability by varying the "data source size" and the "number of tasks". The test environment is a cluster made up of 10 machines (nodes). Each machine has an intel-Core i5-2500 CPU@3.30 GHZ x 4 processor with 4GB RAM, 20 GB of free HDD space. These machines operate with Ubuntu-12.10 and are interconnected by a switched Ethernet 100 Mbps in a LAN. The framework *Apache Hadoop 1.2.0* is installed on all the machines. One of these 10 machines is configured to perform the role of Namenode in the HDFS system and JobTracker in the MapReduce system. However, the other machines are configured to be HDFS DataNodes and TaskTrackers. Overall, we can see, in FIG. 9, that the increasing of tasks improves the processing time. Indeed, we further analyzed the results and discovered some interesting aspects. Seeing the paper length constraint, we can not present all the experiment results.

FIG. 10 shows the "time saving" by increasing tasks. The "time saving" is calculated as the difference between "processing time" corresponding to different "number of tasks". We can see that the time saving to handle $2,2 * 10^9$ tuples (FIG. 10 (a)) decreases when we configure more than "5 tasks". Also, to handle $2,44 * 10^9$ tuples (FIG. 10 (b)), the time saving after "8 tasks" becomes not significant. To sum up our experiment, we note that the "number of tasks" is not the only parameter to be set in order to speed-up the process. Our cluster must be extended in terms of nodes, memory

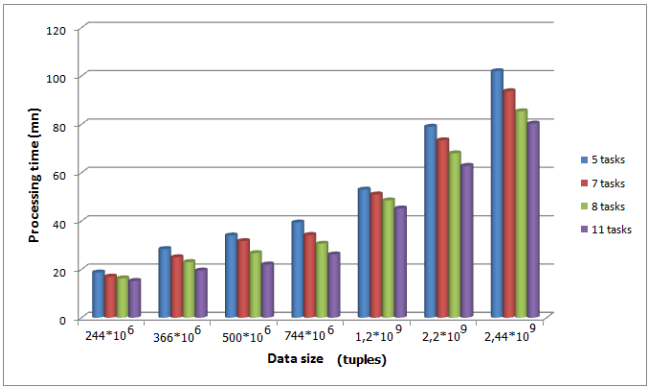

Fig. 9: Proc. time (min.) by scaling up data (tuples) and increasing tasks.

space (RAM, cache), LAN bandwidth, etc. The cluster used for this experiment is a small-sized infrastructure. The HDD space is very low (20 GB per node). Thus, trying to increase tasks, for example, to more than eight while staying on the same resources in terms of HDD, RAM ..., will not make Handoop able to improve performance of the process if HDD space or memory is, already, completely consumed by the eight tasks.

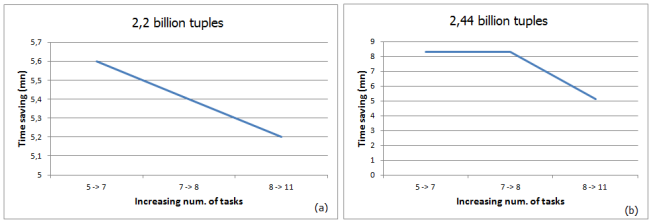

Fig. 10: Time saving (min.) by increasing tasks.

5. Conclusion

The ETL is the core component of decision-support system since all the data dedicated for analysis pass through this process. It should be adapted following the new approaches and paradigms to cope with big data. In this context, we proposed a parallel/distributed approach for ETL process where its functionalities run in parallel way with MR paradigm. In the near future, we plan to finish our experiments on a larger scale both in ETL process level and ETL functionality level. A complete benchmark in which we compare the four approaches (centralized ETL process approach, distributed ETL process approach, Big-ETL approach, Hybrid approach) is an interesting perspective.

References

[1] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[2] S. Mohanty, M. Jagadeesh, and H. Srivatsa, *Big Data Imperatives: Enterprise Big Data Warehouse,BI Implementations and Analytics*. Apress, 2013.

[3] B. Sosinsky, *Cloud computing bible*. John Wiley & Sons, 2010, vol. 762.

[4] J. Dean and S. Ghemawat, "Mapreduce: a flexible data processing tool," *Communications of the ACM*, vol. 53, no. 1, pp. 72–77, 2010.

[5] J. Han, E. Haihong, G. Le, and J. Du, "Survey on nosql database," in *6th international conference on pervasive computing and applications (ICPCA), 2011*. IEEE, 2011, pp. 363–366.

[6] P. Vassiliadis, A. Simitsis, and S. Skiadopoulos, "Conceptual modeling for etl processes," in *Proceedings of the 5th ACM international workshop on Data Warehousing and OLAP*. ACM, 2002, pp. 14–21.

[7] J. Trujillo and S. Luján-Mora, "A uml based approach for modeling etl processes in data warehouses," in *Conceptual Modeling-ER 2003*. Springer, 2003, pp. 307–320.

[8] Z. El Akkaoui and E. Zimányi, "Defining etl worfklows using bpmn and bpel," in *Proceedings of the ACM twelfth international workshop on Data warehousing and OLAP*. ACM, 2009, pp. 41–48.

[9] Z. El Akkaoui, E. Zimànyi, J.-N. Mazón, and J. Trujillo, "A model-driven framework for etl process development," in *Proceedings of the ACM 14th international workshop on Data Warehousing and OLAP*. ACM, 2011, pp. 45–52.

[10] B. Oliveira and O. Belo, "Using reo on etl conceptual modelling: a first approach," in *Proceedings of the sixteenth international workshop on Data warehousing and OLAP*. ACM, 2013, pp. 55–60.

[11] F. Arbab, "Reo: a channel-based coordination model for component composition," *Mathematical Structures in Computer Science*, vol. 14, no. 3, pp. 329–366, 2004.

[12] X. Liu, C. Thomsen, and T. B. Pedersen, "Etlmr: a highly scalable dimensional etl framework based on mapreduce," in *Data Warehousing and Knowledge Discovery*. Springer, 2011, pp. 96–111.

[13] C. Thomsen and T. Bach Pedersen, "pygrametl: A powerful programming framework for extract-transform-load programmers," in *Proceedings of the ACM twelfth international workshop on Data warehousing and OLAP*. ACM, 2009, pp. 49–56.

[14] X. Liu, C. Thomsen, and T. B. Pedersen, "Mapreduce-based dimensional etl made easy," *Proceedings of the VLDB Endowment*, vol. 5, no. 12, pp. 1882–1885, 2012.

[15] S. Misra, S. K. Saha, and C. Mazumdar, "Performance comparison of hadoop based tools with commercial etl tools–a case study," in *Big Data Analytics*. Springer, 2013, pp. 176–184.

[16] X. Liu, C. Thomsen, and T. B. Pedersen, "Cloudetl: scalable dimensional etl for hive," in *Proceedings of the 18th International Database Engineering & Applications Symposium*. ACM, 2014, pp. 195–206.

[17] M. Bala, O. Mokeddem, O. Boussaid, and Z. Alimazighi, "Une plateforme etl parallèle et distribuée pour l´intégration de données massives," *Revue des Nouvelles Technologies de l'Information*, vol. Extraction et Gestion des Connaissances, RNTI-E-28, pp. 455–460, 2015.

[18] R. Kimball and J. Caserta, *The data warehouse ETL toolkit*. John Wiley & Sons, 2004.

[19] D. V. Sarwate, "Computation of cyclic redundancy checks via table look-up," *Communications of the ACM*, vol. 31, no. 8, pp. 1008–1013, 1988.

[20] M. P. Freivald, A. C. Noble, and M. S. Richards, "Change-detection tool indicating degree and location of change of internet documents by comparison of cyclic-redundancy-check (crc) signatures," Apr. 27 1999, uS Patent 5,898,836.