Use of HPC-Techniques for Large-Scale Data Migration

Jan Dünnweber, Valentin Mihaylov,
René Glettler, Volker Maiborn, and Holger Wolff

MaibornWolff GmbH, München, Germany
{Jan.Duennweber,Valentin.Mihaylov,Rene.Gletter,
Volker.Maiborn,Holger.Wolff}@mwea.de

Abstract. Any re-design of a distributed legacy system requires a migration which involves numerous complex data replication and transformation steps. Migration procedures can become quite difficult and time-consuming, especially when the setup (i. e. , the employed databases, encodings, formats etc.) of the legacy and the target system fundamentally differ, which is often the case with finance data, grown over decades. We report on experiences from a real-world project: the recent migration of a customer loyalty system from a COBOL-operated mainframe to a modern service-oriented architecture. In this context, we present our easy-to-adopt solution for running most replication steps in a high-performance manner: the *QuickApply* HPC-software which helps minimizing the replication time, and, thereby, the overall *downtime* of the migration. Business processes can be kept up and running most of the time, while pre-extracted data already pass a variety of platforms and representations toward the target system. We combine the advantages of traditional migration approaches: transformations, which require the interruption of business processes are performed with static data only, they can be made undone in case of a failure and terminate quickly, due to the use of parallel processing.

1 Introduction

The ongoing re-design of a multi-organization customer loyalty system demands a fast large-scale data migration. The system, which was originally launched in the mid-90ies, now holds financial data for over 18 million customers. Whenever a customer performs a transaction involving a business partner, such as subscribing to a certain newspaper, using a certain credit card or shopping at a partner mall, the system credits "bonus points" to the customer. Points can be honored by free weekend getaways, rental cars, CDs, books or other goodies. For facilitating the miscellaneous rewards, the loyalty system is connected to multiple distributed servers and huge databases which must provide high availability. However, until the end of the last decade, the core customer data still persisted on tapes of a 60ies-style mainframe computer, operated by one monolithic COBOL program (as it is still quite common in financial software).

M. Alexander et al. (Eds.): Euro-Par 2011 Workshops, Part I, LNCS 7155, pp. 408–415, 2012.

The company in charge of the core system recently decided to migrate its software to a state-of-the-art *service-oriented architecture* (SOA), i. e. , a scalable network of loosely coupled Web services which can be exchanged or extended on demand. Such a migration scenario is not specific for our loyalty program but very common, whenever IT-landscapes of big financial or actuarial organizations are fused together or modernized. The main difficulties of such a migration are:

1. How to cope with the masses of data which are constantly changing and stored in entirely different formats in the source and in the target system?
2. How to keep the business-critical processes permanently alive, or, at least, absent for only a very short *downtime*, during the data used by these processes is transferred, transformed, re-arranged or re-formatted?

The paper is structured as follows: Section 2 compares popular migration approaches. Section 3 introduces our HPC-software called *QuickApply* and points out where it is superior to traditional migration software, especially in financial applications. Section 4 concludes the paper and summarizes our contributions.

2 Popular Approaches toward Migrating Finance Data

Developing a custom migration software is a challenging task for vendors like us [1], who aim at building efficient and reliable solutions for *single use* (i. e. , once the migration is done, the software is no more needed) in time and budget, and, for researchers as well, who face the problem of mapping well-defined, but static models (e. g. , graphs or fixed networks [1]) to highly dynamic finance software [2].

Let us review the three most popular approaches to data migration and identify their shortcomings for financial applications. All these approaches have an **e**xtract, a **t**ransform, and, a **l**oad step forming the *ETL process* [3].

2.1 The Big Bang Approach

The big bang approach is illustrated in Fig. 1 [3]. Vertical lines represent boundaries between network sites and horizontal lines represent levels of abstraction: *business level* processes are, e. g. , orders and reversals which, on the *tool level*, correspond to work units, i. e. , sequences of jointly committed statements, such as `INSERT`, `DELETE` and `UPDATE`, processed record-by-record on the *database level*.

In step ①, the source DB is disconnected from the legacy system which is shut down to extract all its contents. Big bang migration software focuses on transforming static data which changes no more during the migration (see question 1 of the introduction).

In step ②, the data is transformed, whereby the formatting and arranging of data is performed in a *pipeline-parallel* [7] way, for keeping the downtime as

[1] MaibornWolff[et al] GmbH - `www.mwea.de` - is a software and IT-consulting company, based in Munich and Frankfurt serving some of the European leading providers of tourism, telecommunication and finance services.

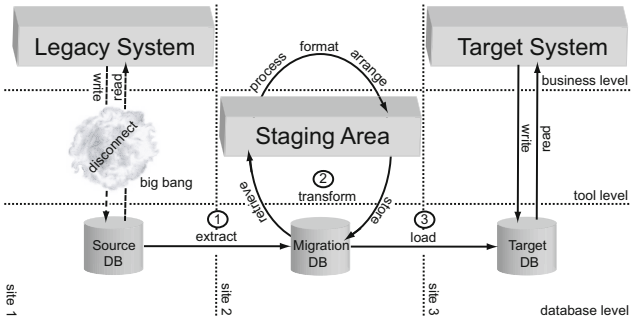

Fig. 1. Big Bang ETL

short as possible (see question 2). Successive transformation steps follow after the data is stored in the migration DB, such that step ② becomes cyclic, as shown in Fig. 1.

In step ③, the extracted data is loaded into the target DB using a mechanism like *batch insert*, *external tables* or Oracle's *SQL*Loader* (see [8] for a performance comparison of these loading mechanisms for ETL).

Advantages of the big bang approach are: new software must be developed and installed only on the staging area; the correctness of all data can be verified completely, once before and once after the migration; and, the big bang approach is *fault-tolerant*, i. e. , a fallback is possible by relaunching the legacy system. Obviously, the big bang approach has the disadvantage of a noticeable downtime of all involved business-level applications.

2.2 The On-The-Fly Approach

The on-the-fly migration [4] is illustrated in Fig. 2. This approach introduces a *data access layer* to the ETL process which has one component on the source side and one component on the target side.

In step ①, the source-side data access layer *captures* δ-data in the source DB. In the δ-data, each record reflects the data that is actually written plus the write statement type (`UPDATE`, `DELETE` or `INSERT`), a timestamp, a sequence number and a work unit identifier. Once the δ-data capturing has started, the source DB's original schema is fully extracted and transferred to the migration DB. In the successive phases, δ-data is extracted analogously.

In step ②, a full dump of source DB is transformed as in the big bang approach and then successively the contents of the δ-data schema, which were captured and extracted during the previous transformation phase are transformed. Provided that the amount of δ-data is reduced in each phase, the migration terminates with a transform phase during which no new δ-data is stored in the source DB (e. g. , during a regular system maintenance interval).

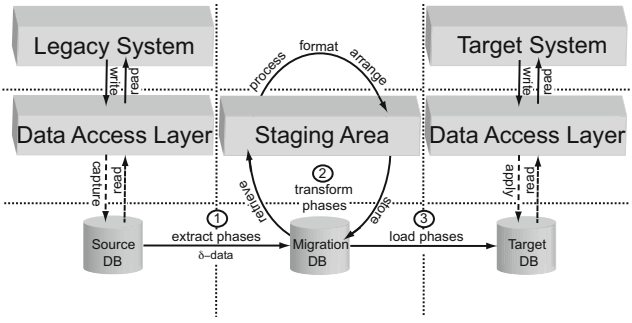

Fig. 2. On-The-Fly ETL

In step ③, the transformed data is loaded. Source-side work units are mapped onto target-side work units which are committed to the target DB. If , e. g. , the δ-data contain a work unit X which corresponds to a reversal and is composed of an `UPDATE` statement x_1 and a `DELETE` statement x_2 in the source DB, it is mapped onto the work unit X' for the target system reversal process, composed of the statements $x'_1..x'_n$ which may differ from the source DB statements regarding their types, affected tables and quantity.

The legacy system is never shut down until the target system is launched. This answers question 2 of the introduction even more satisfactorily than above. In trade-off, there is no convincing answer to question 1, since the work unit mapping for different business processes in step ③ is not always combinatorially decidable. Consider, e. g. , a commit sequence that may correspond either to process X or process Y in the source system. Should the target-side data access layer run process X' or process Y'? Moreover, when steps ② and ③ (δ-data apply plus process mapping) are not faster than step ① (δ-data capture), δ-data will not be reduced and the ETL process will never terminate.

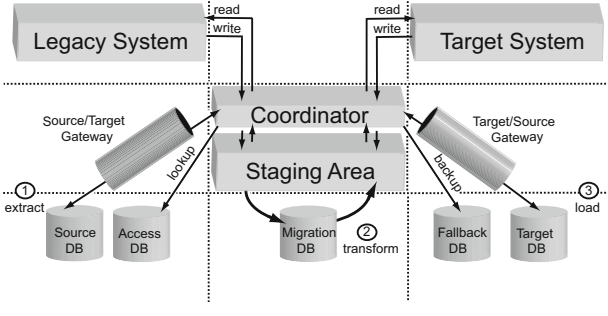

Fig. 3. Parallel Business ETL

2.3 The Parallel Business Approach

In the *parallel business* migration, sketched in Fig. 3, the legacy system is kept in operation even *after* the target system has been launched [5].There is no direct connection between the databases and the business level systems. Any data access in steps ① and ③ is routed through the central *coordinator* which provides the legacy system with target system data and, vice versa, the target system with legacy system data using appropriate *gateways*. The format and location of the most recent version of each record is stored in an *access DB* and before each transformation (step ②) the original data is stored in a *fallback DB*. Downtime is never required, irrespective of the processing load. However, the gateways perform complex but redundant conversions (already performed on the staging area) and any data access is routed across network boundaries, which may result in serious performance drawbacks for this approach.

3 Introducing the QuickApply HPC-Software

For our novel approach to large-scale finance data migration, we developed a parallel software called QuickApply that combines the advantages of the big bang and the on-the-fly migration (Fig. 4).

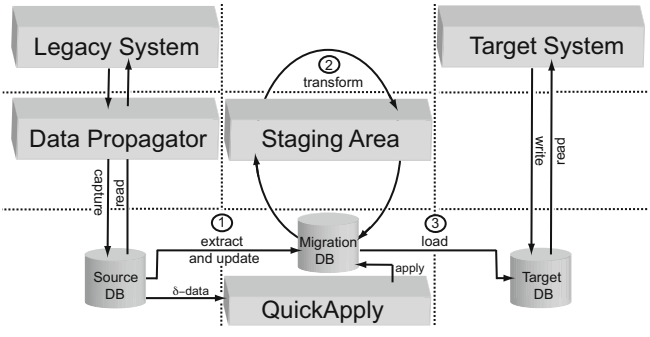

Fig. 4. QuickApply for Parallel ETL

Step ① captures recent changes in the source DB. Contrary to the on-the-fly approach, no target system component depends on the data capturing, and, therefore, a standard tool can be used for this purpose: the *data propagator* [9] for IBM mainframe databases [6].

Step ② makes use of QuickApply to update the migration DB record-by-record until it is in-sync with the source DB. Static data from the source DB is extracted and transferred in advance (i. e. , before the downtime) and the downtime (in case that a noticeable period remains) is fully exploited for running transformations.

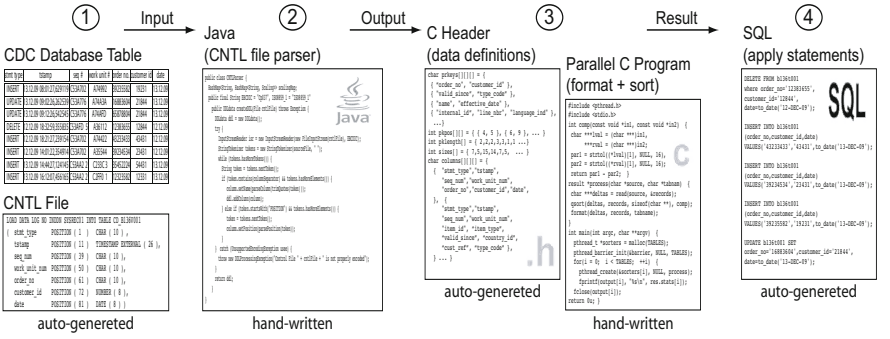

Fig. 5. Overview of the code and data pieces for QuickApply

Step ③ loads the transformed data which, in contrast to the big bang approach, is up-to-date and which, in contrast to the on-the-fly and parallel business migration, must neither be mapped onto processes nor converted.

3.1 The Implementation of QuickApply: Combining Java, C & SQL

QuickApply is a parallel δ-data apply software that is portable across different operating systems, hardware platforms and across SQL-compliant databases as well. As shown in Fig. 5, our implementation makes use of Java, C and multiple input pieces. QuickApply works on the database level and only runs on one network site (the staging area). It processes the **c**hange **d**ata **c**apture tables (CDC) which IBM's data propagator uses for capturing δ-data in a δ-data schema.

The tiny code and data excerpts in Fig. 5 (not intended to be read) illustrate the relation between auto-generated and hand-written code: In step ①, the input data are CDC tables and CNTL-Files (table structure definitions) which are generated by the data propagator and parsed by a hand-written Java program that also provides the user interface. In step ②, the output generated by the Java program is a C-header file (`.h`), wherein the CDC-data is encoded in arrays. In step ③ a hand-written C-program sorts and formats the δ-data. In step ④, SQL-statements, which the C-program generates as result, are applied to the migration database. As shown in Fig. 6, for each CDC-table, the C-program explicitly starts a new POSIX thread which sorts the corresponding array, according to the elements' sequence numbers.

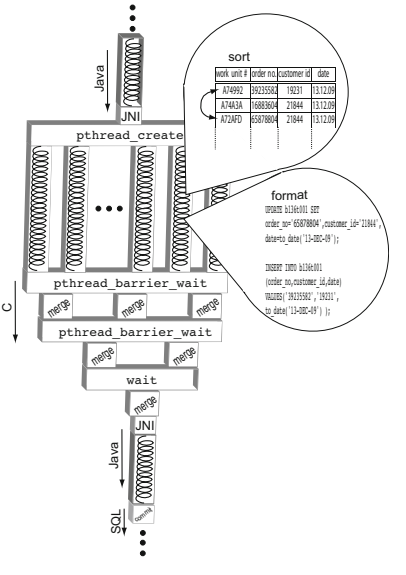

Fig. 6. Execution Model

After sorting the threads join at a barriers where the statements, sorted table-wise, are merged together, such that statements forming a common work unit are committed sequentially (to prevent concurrency anomalies like *lost updates* [11]).

Due to the fact that QuickApply runs below the database level, it is significantly faster than a replication tool which applies work units directly to the database using , e. g. , JDBC.

3.2 Case Study: Time Needs For Migrating the Loyalty Program

Our example data stock is ≈ 3 TB large. Our migration DB (and the target DB) are approximately 400 kilometers away from the legacy system. The periodical updates processed via QuickApply (i. e. , the δ-data volume) vary: Fig. 7 shows the runtimes (in sec.) for the Java-based part and the multithreaded C-segment of QuickApply for 906672 (≈ 350 MB of δ-data), 3485068 (≈ 1.4 GB) and 6095741 SQL statements (≈ 4.2 GB).

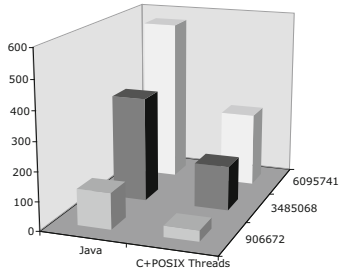

Fig. 7. Java vs. C Part

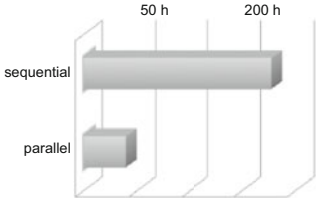

Fig. 8. Full Transform

Our execution platform (the staging area) comprises two 24 CPU Servers with 8 cores per CPU, both running the Java HotSpot^{TM} 64-Bit Server VM under Linux 2.6.18. QuickApply helps saving valuable downtime, which is fully usable for running transformations. Fig. 8 and Fig. 9 show the parallel runtimes for the transform step and the clearance steps, following the replication, each compared to the sequential runtime.

Each bunch of δ-data was captured by running the data propagator for 24h. We use a dedicated line which has an average transfer rate of ≈ 30 MBit/s, i. e. , transferring 2–4 GB δ-data takes only 6–12 minutes. When QuickApply is used, the final transfer carries δ-data, not the full load, and this is the only transfer which we perform during the downtime.

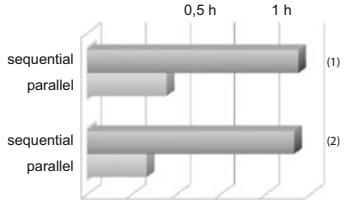

Fig. 9. Data Cleansing (1) & Functional Clearance (2)

In experiments with 1–8 processors, the data replication scaled almost linearly and the transformation speedup ranged from 3 to 5. The average efficiency for was $> 50\%$.

4 Conclusion

Technically, the presented work is in the tradition of component-based *grid computing*, which also combines multiple technologies (and, potentially, multiple programming languages) [12].

Compared to other research in the data migration realm, our HPC-approach has proven to be more efficient than a big bang and, which is especially important in finance, more reliable than on-the-fly or parallel business. Some big bang implementations migrate audit data separately for reducing downtime. Other implementations, additionally, keep the legacy DB running in *read-only* mode permanently. Such optimization can be combined with QuickApply, since the data propagator keeps the source DB always up-to-date and a selection of DB contents is possible via filtering. From our experiences with QuickApply, we conclude that properly implemented HPC-methods, like Pthreads, JNI and partially auto-generated code, are no risky adventure but a tremendous improvement for applications processing finance or other sensitive data.

References

1. Hall, J., Hartline, J., et al.: On Algorithms for Efficient Data Migration. In: Proceedings of ACM Symposium on Discrete Algorithms, Washington, DC, USA (January 2001)
2. Allgaier, M., Heller, M.: Research Challenges for Seamless Service Integration in Enterprise Systems. In: Proceedings of IE4SOC: Workshop on Industrial Experiences for Service-Oriented Computing, Stockholm, Sweden (November 2009)
3. Brodie, M.L., Stonebraker, M.: Migrating Legacy Systems. Morgan Kaufmann, San Francisco (1995) ISBN 9781558603301
4. Wu, B., Lawless, D., et al.: The Butterfly Methodology: A Gateway-Free Approach for Migrating Legacy Information Systems. In: IEEE Conference on Engineering of Complex Computer Systems, Villa Olmo, Como, Italy (September 1997)
5. Pyla, P.S., Tungare, M., Homan, J., Pérez-Quiñones, M.A.: Continuous User Interfaces for Seamless Task Migration. In: Proceedings of HCII: IASTED Conference on Human-Computer Interaction, San Diego, CA, USA (July 2009)
6. Ebbers, M., O'Brien, W., Ogden, B.: Introduction to the New Mainframe: z/OS Basics. Vervanté books, Huntington Beach (2005) ISBN 9780738496740
7. DeWitt, D., Gray, J.: Parallel Database Systems: The Future of High Performance Database Systems. Comm. ACM 35(6), 85–98 (1992)
8. Orlando, S., Orsini, R., Raffaetà, A., Roncato, A.: Trajectory Data Warehouses: Design and Implementation Issues. Journal of Computing Science and Engineering 1(2), 211–232 (2007)
9. Jäntti, J., Kerry, D., Kompalka, A., Long, R., Mitchell, G.: DataPropagator Implementation Guide. IBM Redbooks, Armonk (2002) ISBN 0738426342
10. Liang, S.: Java^{TM} Native Interface: Programmer's Guide and Specification. Prentice Hall, Harlow (1999) ISBN 9780201325775
11. O'Neil, P.: Database: Principles Programming and Performance. Morgan Kaufmann Publishers (June 1994) ISBN 9781558602199
12. Dünnweber, J., Gorlatch, S.: Higher-Order Components for Grid Programming: Making Grids More Usable. Springer, Germany (2009) ISBN 9783642008405