HiPAS: High Performance Adaptive Schema Migration

Development and Evaluation of Self-Adaptive Software for Database Migrations

Hendrik Müller, Andreas Prusch, and Steffan Agel
Pasolfora GmbH
An der Leiten 37, 91177 Thalmässing, Germany
{hendrik.mueller|andreas.prusch|steffan.agel}@pasolfora.com

Abstract – **HiPAS stands for "High Performance Adaptive Schema Migration" and is a self-adaptive software system, aimed at reducing downtime during offline database migrations by automatically adapting to available system resources. The process of a database migration can be shortened by parallelizing the data transfer up to a certain degree. In this article, we describe how HiPAS was enabled to continuously adapt the parallelization degree according to its operational environment in order to avoid both overloading and idle resources. To automate the developed method, we implemented HiPAS following decisions taken within the dimensions of design space for self-adaptive software. Based on a centralized control pattern in distributed systems, HiPAS uses a feedback loop to enable adaptions. Hence, according to monitored system information, the current utilization is adjusted whenever necessity is assumed. To enable a flexible adaption, the total amount of migration data is partitioned into equal sized transfer jobs, which are distributed across available instances and networks. HiPAS is invoked on database layer and controlled by a temporarily created autonomous database user. Therefore, migration metadata are stored inside tables and highly integrated with the actual migration data. HiPAS was designed and evaluated iteratively following the IS research framework and reveals significant downtime reduction potential compared to non-adaptive migration approaches like Oracle "Data Pump". Our results serve as a contribution for all practitioners, who seek to perform database migrations within a challenging timeframe, as well as researchers on self-adaptive software and their various fields of application.**

Keywords-Adaptability; Anticipation; Self-Adaptive Software; Database Migration; Parallelization.

I. INTRODUCTION

The rapid technical developments inside changing markets, as well as the need for efficiency enhancements, mainly driven by cost pressure, require an occasional transfer of running information systems into a new environment, which fulfills the operational requirements in a more suitable way. This process is referred to as software migration [1], [2] and meanwhile the software's availability can be limited depending on the chosen migration method. Regarding this, basically two approaches can be differentiated:

- online Migration: continuous availability
- offline Migration: interrupted availability

In some critical environments, a downtime is not acceptable, thus online migrations need to be performed. This article deals with the variety of cases, which do not require a costly and complex online migration and a planned downtime is tenable. In that case, the main concern is to keep the downtime as short as possible since the duration of unavailability may result in opportunity costs. In particular, we target migrations applying the "big-bang" strategy [3], thus data is fully migrated at once in contrast to incremental migrations. Since the legacy system (source system) is shut down during the data transfer, starting the target system, referred to as cut-over [4], cannot be performed before all required data has been transferred to the target system's database. The length of downtime depends on the migration approach taken. For database migrations, different system layers can be involved determining the performance and granularity of data selection (see Section II). We investigated the applicability of adaptive capabilities for database migration software in order to reduce the necessary downtime by parallelizing data transfer up to an optimal parallelization degree, which will be continuously adapted to the system's load capacity. Prior tests indicated that overloading the target or source systems resources leads to a temporary stagnation of the whole migration progress, whereas a low utilization wastes available resources, thus underachieving existing downtime reduction potential. In this manner, we contribute to the field of self-adaptive software and support the statement by de Lemos et al. that "self-adaption has become an important research topic in many diverse application areas" and "software systems must become more versatile, flexible, dependable […] and self-optimizing by adapting to changes that may occur in their operational contexts, environments and system requirements" [5].

Moreover, the developed approach "HiPAS" (High Performance Adaptive Schema Migration [1]) is intended to provide dependability and interruptibility, since migration software should be able to identify where to resume an interrupted migration process instead of starting from scratch avoiding the necessity of rescheduling a planned downtime.

Further technically conditioned features will be added in Section III as consequences of the preliminary considerations. Section IV summarizes HiPAS´ architecture by means of introducing adaptability challenges of the subsequent described migration process (Section V). The adaptive capabilities are outlined in Sections VI and VII.

Finally, in Section VIII, we evaluate HiPAS, which refers to both the designed migration method and the migration software, currently implemented in Oracle PL/SQL syntax comprising 8,540 lines of source code.

II. Present Migration Approaches

As introduced previously, migration approaches can be differentiated regarding the availability of the migrated systems into online and offline migrations. For stated reasons, we focus on offline migrations, which can be further classified concerning their own characteristics and their applicability for certain database characteristics:

- invocation layer
- support for change of platform
- support for change of endianness
- support for change of character set
- downtime proportionality

To obtain an overview of present migration approaches, we classified existing available methods using the example of Oracle databases and the above enumerated characteristics:

TABLE I. Classification of Present Migration Approaches.

Migration Method	Invocation Layer/ Granularity	Downtime Proportionality	Platform Change	Endianness Change	Character Set Change
Storage Replication	Storage/ Storage	negligible	no	no	no
Transportable Database	OS/ Database	Database Size	yes	no	no
Transportable Tablespaces	OS/ Tablespace	Tablespace Size	yes	no	no
Cross Platform Transportable Tablespace	OS/ Tablespace	Tablespace Size	yes	yes	no
Transportable Tablespaces using Cross Platform Incremental Backups	OS/ Tablespace	Data Alteration Rate	yes	no	no
Oracle-to-Oracle (O2O)	OS/ Schema	Amount of Migration Data	yes	yes	no
Datapump	Database/ Value	Amount of Migration Data	yes	yes	yes
Export/Import	Database/ Value	Amount of Migration Data	yes	yes	yes

As shown in Table I, the divergence of the source and target database in terms of platform, endianness and character set technically limits the available migration methods.

A critical decision criterion for the remaining contemplable methods is the demand for downtime shortness resulting in lower opportunity costs during the unavailability of the database and all relying applications. The fact that a high throughput for data transfer was achieved as yet by eliminating upper layers and protocols, leads to the conflicting goals of flexibility and performance when selecting a migration method. The lower a layer a migration is invoked on, the more flexibility is lost, since changes of database characteristics might not be supported and the possible granularity for migration data selection decreases. Finally, downtime proportionality refers to the entity, which the downtime length depends on; this can be the amount of migration data or the data alteration rate if incremental methods are used.

When designing HiPAS, we pursued the goal of achieving a short downtime and at the same time providing the flexibility of migrating between divergent databases and selecting the data as granular as possible. This was achieved by invoking the migration on database layer without ever leaving this layer during the whole migration process and by parallelizing the data transfer adaptively in respect of the system's resources. Therefore, we add "adaptability" as a further decision criterion for migration software capabilities.

III. Preliminary Considerations

The performance of migration software highly depends on how well its design fits to the operating environment and the intended range of functions. Previous system and data analyses are necessary to conclude with a migration design, which has been aligned to the findings in multiple iterations following the guidelines of design science in information system research [6]. Figure 1 shows how the designed artefact HiPAS is related to its environment and knowledgebase base inside the information systems research framework.

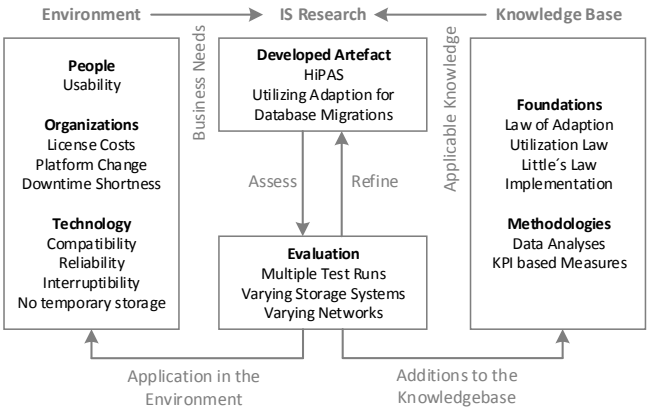

Figure 1. HiPAS as an IS Research Artefact (Adapted from Information Systems Research Framework [6]).

HiPAS was intended to be built upon findings of preliminary analyses (Knowledgebase) described in this section, as well as business requirements (Environment) and from then on has been improved continuously, based on evaluation runs performed in a variety of different environments provided generously by customers.

A. Enterprise Data Structures

When moving existing data files to the target system, as migration approaches invoked on storage and database layer do (see Section II), the valuable downtime is partly spent migrating unnecessary or useless data. The allocated size of a data file implies unused space and indexes. To gain an overview of typical storage occupancies, we analyzed 41 SAP systems productively running at a German public authority by querying the allocated disk space, the used disk space and the space used for indexes with a result shown in Figure 2.

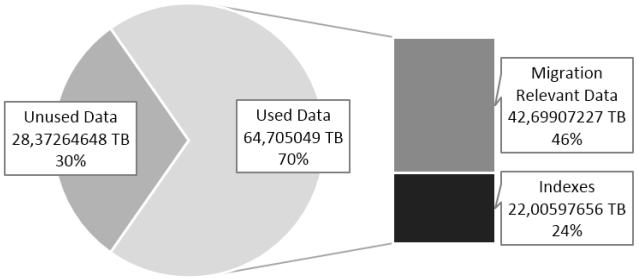

Figure 2. Average Structure of Allocated Data.

For these 41 SAP Systems, we identified an overall amount of 93.08 TB allocated data. From this amount, about 28 TB (30 %) represented allocated space, which was not yet filled with data. From the used space of 65 TB, about 22 TB (24% of the overall amount) were filled with indexes. The remaining 43 TB (46% of the overall amount) represent the actual relevant data, which necessarily needs to be transferred into the target database within a migration. Indexes can be created at the target system and do not have to be transferred, thus saving network bandwidth. Depending on the layer the migration is invoked on, unused but allocated data can be excluded as well.

In this case, if all of the analyzed SAP systems needed to be migrated, migration tools not supporting data selection would utilize all involved system resources for transferring data, of which approximately 54% is useless on the target system. Invoking a migration method on software layer, enables both excluding useless data autonomously and implementing self-adaptability.

B. Endianness

When performing a database migration, the byte order in which the source and target system store bytes into memory needs to be considered. This byte order is referred to as endianness and data is stored into data files accordingly, so the endianness can affect the amount of available migration methods and the overall needed downtime.

A major part of migration demanding customers served by the authors of this paper currently initiate migration projects due to licensing and maintenance costs, this amount is strongly influenced by an increasing number of platform migrations from Solaris to Linux, requiring subsequent migrations on upper layers such as the databases tier. The latest International Data Corporation (IDC) report on worldwide server market revenues substantiates this observation by stating that Linux server revenue raised from 17% in Q4 2010 to 23.2% in Q2 2013 compared to Unix decreasing from 25.6% down to 15.1% [7]. The Unix-based Solaris operates on processors following Oracle´s SPARC architecture, whereas Linux distributions can be used on systems based on Intel processors. When migrating from Solaris to Linux, the endianness changes accordingly from big endian to little endian, so the data files cannot simply be moved without converting them before or after the transfer as shown in Figure 3.

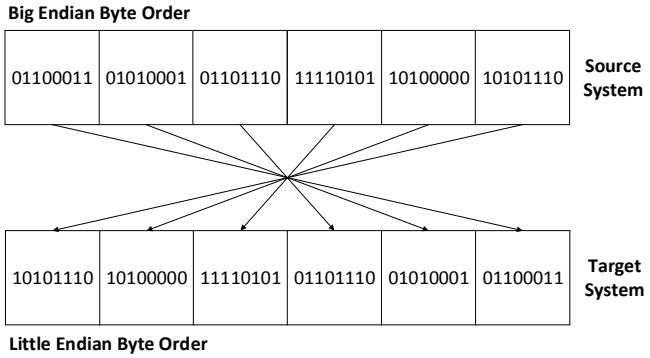

Figure 3. Change of Endianness between Source and Target System.

Alternatively to converting data files, the database migration can be invoked on a layer, which supports saving the data into new files on the target system, such as export-import-tools and HiPAS do. In this case, migration performance can be enhanced by means of adaptive capabilities.

C. Storage I/O Controller

As a consequence of the requirement for downtimes as short as possible, a utilization degree of the underlying storage systems has to be achieved, which enables short response times. The overall amount of requests inside a system (N) equals the product of arrival rate (a) and average response time (R) as expressed by Little´s Law [8]:

$$N = a \times R \tag{1}$$

In addition the Utilization Law [9] defines the utilization (U) of the I/O controller as the product of arrival rate and average processing time (R_S):

$$U = a \times R_S \tag{2}$$

By combining these relations, it becomes clear that the response time depends on the I/O controller's utilization as described within the following formula [10]:

$$R = \frac{R_S}{1 - U} \tag{3}$$

The relation shows that the response time does not change linearly to the utilization. At higher utilizations, the response time grows exponentially as clarified in Figure 4.

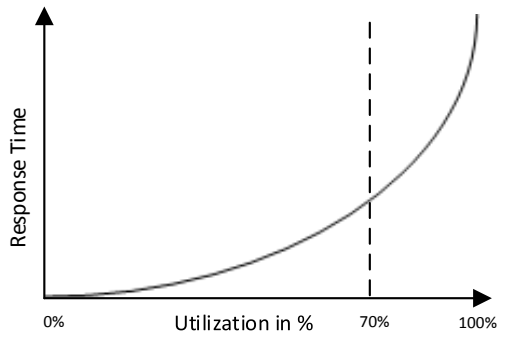

Figure 4. Relation of Utilization and Response Time.

By adapting to the source and target system resources, HiPAS continuously changes the utilization of the I/O controller in order to achieve an optimal relation of response time and utilization supporting the shortest possible overall duration. The storage manufacturer EMC generally describes an average utilization of 70% as optimal [10].

IV. HiPAS Architecture

Following the goals introduced in Section I, we designed the HiPAS migration method as describes in the following.

A. Everything is a Tuple

When performing an automated and controllable migration, a number of interim results arise, e.g., during the analysis of source data. Keeping these information, as well as logging and status information is necessary for the administrator to manage and verify the migration and for the software itself to handle parallel job executions autonomously. The necessity for saving and querying migration metadata leads to HiPAS's design paradigm of not leaving the database layer during the whole migration process. Interim results such as generated DDL and DML Statements for later execution are represented by tuples of tables inside a temporary migration schema enjoying advantages of the databases transactional control mechanisms. The paradigm of everything being a tuple is emphasized by the following list:

- objects to create are tuples (table "cr_sql")
- data to transfer are tuples (table "transfer_job_list")
- running jobs are tuples (table "mig_control")
- parameters are tuples (table "param")
- logs are tuples (table "logging")

After a migration has been performed, its success and the transferred data´s integrity have to be verified. Since logging information was stored during the whole process inside the logging table, SQL can be leveraged to query for certain transferred objects or states or both. For optimizing the migration process, sorting, calculating and analytical capabilities of SQL are utilized, thus, there is no need for any other migration application on operating system level than the database management system (DBMS) itself.

B. Adaptability and Dependability Problems

When designing the migration method and implementing the related software, several challenges had to be faced. In this section, we will briefly introduce some of the most interesting problems and their intended solutions:

- Utilization Problem
- Knapsack Problem
- Distribution Problem
- Dependency Problem
- Index Problem

Subsequently described solution approaches for the above listed problems will provide an overview of the conceived migration method. In-depth sections are referenced.

1) Utilization Problem: Utilization cannot be planned generally since systems behave differently depending on their resources and further running processes. We varified this statement during the evaluation phase by performing migration test runs that have a preliminary defined static parallelization degree. This leads to the risk of both overloading a system and on the other hand not utilizing idle resources. Derived from the relationship between utilization and response time described in Section III-C, Figure 5 shows how the overall performance, in terms of transfer time, behaves at increasing parallelization degrees:

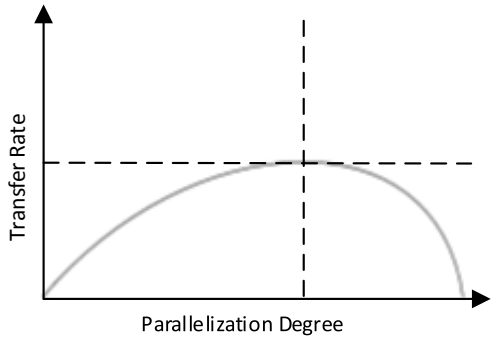

Figure 5. Expansion when Overloading the Storage System.

By choosing the currently optimal parallelization degree adaptively at any time, HiPAS targets an optimal and dynamic utilization, which leads to the shortest possible transfer phase. In this way, we reduced the risk of utilizing the systems too much or not enough. Parallelization is implemented by means of background jobs started through the database scheduler. In this way, the yet manual task of finding the optimal parallelization degree for the respective system environment is intended to be done by HiPAS automatically and adaptively, implicating the ability to change this value dynamically during the whole transfer process.

2) Knapsack Problem: From an amount of objects, defined by their weights and values, a subset with limited weight and maximum total value has to be chosen [11]. This knapsack problem reflects the challenge of choosing optimal combinations of different sized tables to transfer in parallel, since the available computing resources are limited. Large tables should be preferred in a way of starting their transfer at the beginning of the migration process, because a possible failure can require a restart of the table transfer thus delaying the whole migration when started too late. HiPAS circumvents the knapsack problem by dividing large tables into equal sized partitions, which can be transferred in parallel. This offers flexibility in scheduling the data transfer and dynamically adapting the current parallelization degree.

3) Distribution Problem: Depending on the migration environment, the accruing work load can be distributed on multiple instances of a cluster. In terms of network bandwidth, multiple database links can be created on different physical network connections between the source and target system. In this case, HiPAS will distribute data to be transferred equally on the available database links in order to utilize the total available bandwidths. In case of a real application cluster (RAC), HiPAS distributes running transfer jobs on the available instances. Then the fact of the previously mentioned partitioning of large tables needs to be considered. We optimized the data buffers of the instances by distributing transfer jobs, which continue a large table, to the instance, which already transferred previous parts of the same table to avoid reloading the table into multiple buffers of different instances. The corresponding algorithm is explained in Section VII-D.

4) Dependency Problem: When invoking the migration on database layer, dependencies among the transferred objects need to be considered for the transfer order. Surely, users need to exist before importing data into created tables and granting permissions found in the source schema. Constraints like foreign keys have to be disabled temporary, so HiPAS does not have to spend time for calculate a strict and inflexible transfer order. If reference partitioning was used inside the source schema, a parent table needs to exist before the child table can be created following the same partitions. For considering such dependencies, HiPAS calculates a transfer schedule in the first place. Since possible existing triggers will be transferred as well, they need to be disabled during the migration process in order to avoid unexpected operations on the target system, e.g., invoked by an insert trigger.

5) Index Problem: Indexes can either be created directly after table creation or after the table has been filled with data. When creating the index before data load, they will be built "on the fly" during the transfer phase, in contrast, after data load, an additional index buildup phase would need to be scheduled. The right time for indexing depends on the target storage system and network bandwidths. In case of a highly powerful storage system, it might be reasonable to build the indexes directly during data import since the network represents the bottleneck of the whole migration and the storage system would idle otherwise. On the other hand, storage systems can be overloaded when indexes have to be created at import time. Consequently, the decision about the indexing time is another use case for the adaptive capabilities of HiPAS explained in Section VII-C.

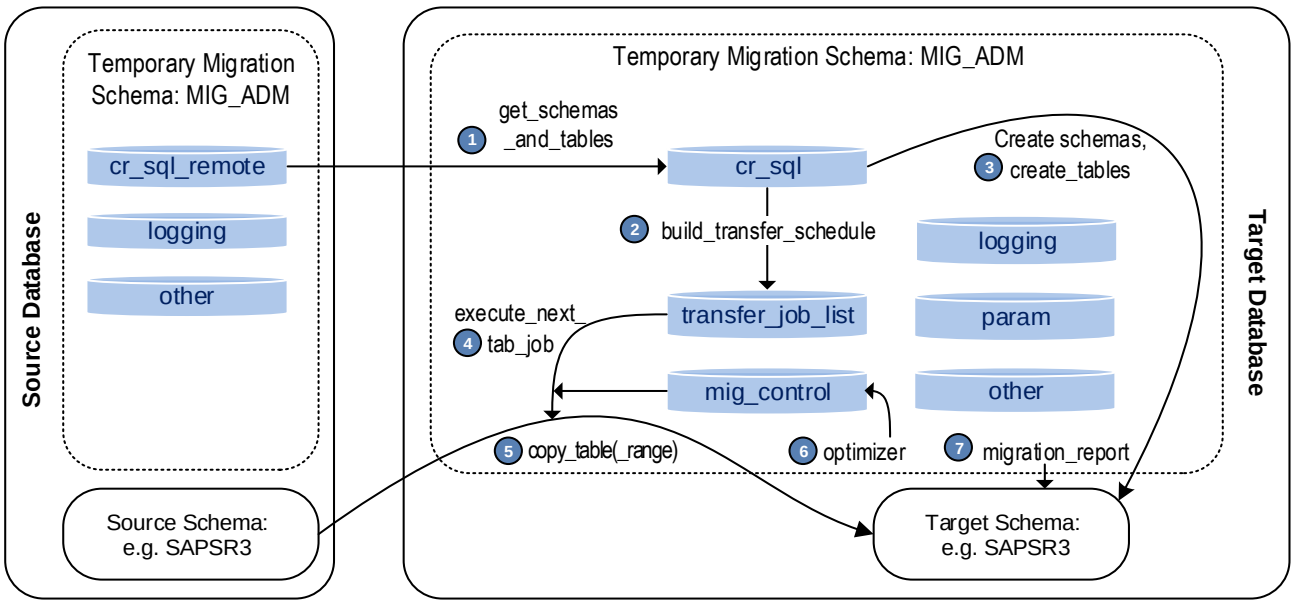

Figure 6. HiPAS Architecture.

V. COMPONENTS AND MIGRATION PROCESS

Assuming that both source and target database system have been physically connected preliminary and are configured to be accessible by each other, the migration process consists of three main phases invoked on the target system, which are briefly described subsequently:

1. Installation and Pre-Transfer (Step 1-3)
2. Adaptive Data Transfer (Step 4-6)
3. Post-Transfer and Uninstallation (Step 7)

Figure 6 shows the steps of these phases, which are invoked on the target system.

A. Installation and Pre-Transfer

Following the paradigm of not leaving the database layer, an additional and temporary schema is created inside both source and target database during an automated installation phase. All subsequent operations will be done by the owner of this schema. Creating this user, as well as creating and compiling a PL/SQL package, needed for performing the migration, is part of an automated installation process. Prior to the data transfer phase, the source schemas need to be analyzed and accordingly created inside the target database. For this purpose, SQL statements for creating the identified objects will be generated and stored inside the table "cr_sql_remote". This table will be copied to the target site and contains information regarding the objects to be created and its creation status. In addition, every operation performed causes status information to be written into the table "logging" (see Figure 7), enabling the database administrator to perform any necessary analysis, e.g., by querying for possible errors during or after the migration:

```
select logdate, loginfo from logging where info_level
= 'ERROR';
```

After the initial analyses of the source schema, all identified objects have the status "init" and, therefore, will be created by HiPAS at the target site. All objects containing "created" inside their corresponding status column will be ignored, enabling the whole migration process to be paused and continued at any time. The table "param" (see Figure 7) serves as a user interface for parameterizing HiPAS manually beforehand, in case certain adaptive capabilities shall not be utilized.

Techniques like reference partitioning inside the source schema have to be considered and will determine the order of creation, since child tables will not be created and partitioned unless the related parent table exists. The Index creation is either part of the pre-transfer or will be initiated after all tables are filled with data. HiPAS decides automatically for the most suitable approach depending on the storage system and network bandwidth as described in Section VII-C.

B. Adaptive Data Transfer

The data transfer is based on two simple SQL statements:

- Insert into a table as selecting from a source table
- Querying remote tables through a database link

The combination of these statements makes it possible to fill local tables with remotely selected data. The resulting command is generated and parameterized at runtime:

```
sql_stmt := 'insert /*+ APPEND */ into "' || schema ||
'"."' || table_name || '" select * from "' || schema
|| '"."' || table_name || '"@' || db_link;
```

This statement is generated and executed by transfer jobs. The number of transfer jobs running in background is adapted continuously and depends on the resource utilization. As a pre-transfer stage, metadata of all objects stored in the source schema has been inserted into a table named "transfer_job_list". Tables to be transferred, exceeding a defined size, will be partitioned and, thus, transferred by multiple transfer jobs. In this case, the job type changes from "table" to "table_range" and row IDs mark the range's start and end (see Figure 7).

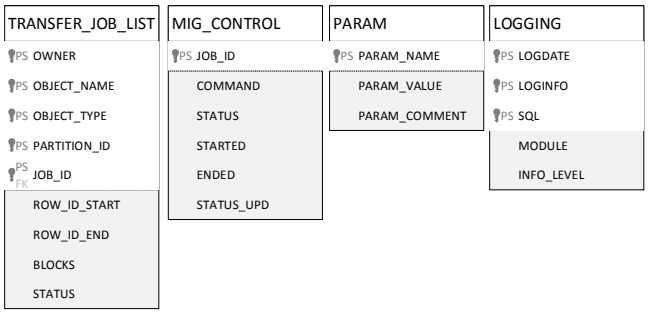

Figure 7. Metadata Entities for the Adaptive Data Transfer Phase.

Through partitioning, HiPAS can adapt more flexible to the current utilization, since the number of parallel jobs can be reduced or increased more frequently. HiPAS' table "mig_control" (see Figure 7) lists all background jobs transferring the objects stored in "transfer_job_list". In this respect, the column "command" inside "mig_control" serves as an interface for controlling the transfer process, either autonomously by HiPAS or manually by the database administrator. When overwriting its content with keywords like "stop" or "continue", individual jobs will be stopped after finishing or continued, causing timestamps to be written into the column "status_upd" and if necessary into "ended". By this means, HiPAS is able to reduce or increase the number of parallel running transfer jobs transparently in respect of the optimizer's decision, which is described in Section VII. For the migration time, all constraints will be disabled temporary by HiPAS, enabling the table "transfer_job_list" to be ordered by blocks instead of considering key dependencies. Existing database triggers will also be disabled avoiding any unintended execution during the database migration.

C. Post-Transfer

After all source data has been transferred into the target schemas, the data has to be validated. Documenting data consistency and integrity is mission critical both for target

database operation and for legal reasons. Only after verifying the equality of source and target data, the migration can be declared as successful, requiring HiPAS to not only compare source and target sizes, but also counting the rows of all tables. Finally, the disabled constraints and triggers will be enabled again.

VI. ENABLING PARALLELIZATION

In order to control the degree of HiPAS utilizing the available hardware resources the migration data transferred at the same time must be limitable. Restricting the number of parallel processed tables would be inappropriate since it required similar sized tables. Instead, a defined number of blocks form a pack of data and a certain number of packs can be processed at the same time. That is, each pack has the same size and will be transferred by a single transfer job. Thus, adding or removing a transfer job burdens respectively disburdens the source and target system. HiPAS adapts to the underlying system resources by deciding autonomously how many transfer jobs are possible at any time.

To enable the amount of data to be partitioned into equal packs, a so called block split range defines their size. Since the tables on the target system are filled by generated "insert as select"-statements, its scope can be limited to a range between two row IDs, which represent the beginning and the end of each data pack. During the source schema analysis, these row IDs are identified by an analytical function. In this manner, large tables are partitioned into groups with row ID boundaries as Figure 8 shows exemplary.

```
GRP MIN_RID            MAX_RID
--- ------------------ ------------------
  2 AAAig0AAAAAABGAAAA AAAig0AAAAAABh/CcQ
  0 AAAig0AAAAAAACAAAA AAAig0AAAAAAAl/CcQ
  1 AAAig0AAAAAAAmAAAA AAAig0AAAAAABF/CcQ
```

Figure 8. Assigning Row IDs as Group Boundaries.

The identified IDs will be used during the transfer phase to limit the data of a single transfer job to the given block split range by adding a "where rowid between"-clause when selecting from the remote database:

```
insert into schema.table_name select * from
schema.table_name@db_link where rowid
between MIN_RID and MAX_RID;
```

Having partitioned the full amount of migration data into parts of a maximum defined size (block split range), HiPAS creates equally treatable transfer entities. These entities can be parallelized up to a degree defined by an adaptive transfer optimizer.

VII. ADAPTIVE CAPABILITIES

For parallelizing the data transfer during phase 2 of the migration process (see Section V-B) with an optimal parallelization degree, we target an adaptive migration software. Adaptivity in general describes the capability of adjusting to an environment. In biology, the term is often used to describe physiological and behavioral changes of organisms in process of evolution. In informatics, the term is transferred to systems or components, which adapt to their available resources. However, here not to increase reproduction chances but often in order to achieve an optimal system performance. Adaption improves the resource efficiency and flexibility of software-intensive systems and means that a system adapts to changes of its environment, its requirements and its resources [12]. According to Martín et al. adaption can also be seen as the first of three stages of the currently conceivable system complexity extent. Anticipation and rationality follow as further stages [13] (see Figure 9).

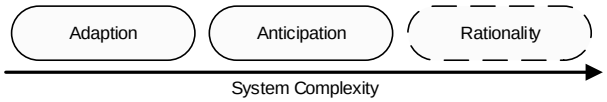

Figure 9. Levels of System Complexity (Adapted from [13]).

Thus, adaption describes the interaction of two elements: A control system and its environment. The goal is to reach a defined state of the environment by means of actions initiated by the control system [14]. The control system then reacts on the self-precipitated changes of the environment with initiating new changes. It has been defined that an adaptive system is present, if the probability of a change of a system S triggered by an event E is higher than the probability of the system to change independently from the event:

$$P(S \rightarrow S'|E) > P(S \rightarrow S') \text{ [13]} \quad (4)$$

Furthermore, the condition has to apply that the system reaches the desired state after a non-defined duration. This implies the convergence of the mentioned probabilities towards infinite:

$$\lim_{t \rightarrow \infty} P_t(S \rightarrow S'|E) = P_t(S \rightarrow S') \text{ [13]} \quad (5)$$

This law of adaption [13] requires the control system to know for each modification of its environment a sufficiently granulated attribute, which contributes to the desired state´s achievement allowing the adaption to end. For the first stage of complexity, the direction and the extent of modifications are built upon each other, thus, enabling the system to reach the desired state incrementally. If the modifications are not steps of a targeted adjustment process, but based on knowledge, predictions [15] or intuition, the process can be defined, in terms of system complexity, as anticipation. The third stage "rationality" implies intelligence; those systems are able to react to unpredictable changes of their environment and to balance contradictory objectives against each other [13]. This stage exceeds the objective of this paper and, therefore, was not scoped. Applying this differentiation on the design of an adaptive migration software, two approaches emerge for parallelizing the data transfer:

- A solely **adaptive** system, based on an incremental adjustment process, until changes do not evoke further improvements, thus, reaching the state of an optimal parallelization degree.
- An **anticipatory** system, which makes continuously new modification decisions independently of each other, based on knowledge about used and monitored resources.

These two approaches have been designed and implemented as described subsequently and evaluated as described in Section VIII. Due to HiPAS' scalable architecture, the respective procedures could be implemented as plugins and additionally started for evaluation. Both plugins control the data transfer via values inside the table "mig_control" (see Section V-B) serving as an interface.

A. Adaption

The solely adaptive approach will successively increase the parallelization degree and, therefore, the source and target systems utilization. After each enhancement its consequences on the system environment meaning the migration performance is measured in terms of inserted megabytes per time unit. The adaption can be started by running an additional procedure "calibrate", which invokes either the procedure "increase" or "decrease" for modifying the parallelization degree, starting from one transfer job per database link at the same time. The number of jobs to be added or deducted will be reduced after each time a change in direction was required, by this means the algorithm brings the number of parallel jobs closer to the optimum. After reaching a defined modification count (number of jobs to add or deduct) the algorithm assumes having approximated the optimal parallelization degree and the adaption ends, representing the finiteness requirement of adaptive systems.

The variable "diff_level" describes the current modification extent, meaning the number of jobs to start additionally or to stop after finishing. To reach a required level of flexibility for changing the number of jobs shortly, the size of a transfer job is limited to the introduced block_split_range. The following code example shows how the number of jobs is reduced by the value of the variable "diff_level":

```
update mig_control set command = 'STOP' where job =
'loop_while_jobs_todo' and command = 'continue' and
rownum <= diff_level; commit;
```

Since the tuples inside "mig_control" represent background jobs and each tuple has a row number, jobs can be stopped for each row number being smaller than "diff_level". The mentioned value "loop_while_jobs_todo" is the name of the procedure every background job runs for processing all defined transfer jobs listed inside the table "transfer_job_list". If a background job is marked with the command "STOP", it will be deleted after finishing the current transfer job and afterwards marked with the keyword "ended".

B. Anticipation

If the adaption is based on predictions, we call it anticipation as the next level of complexity [13]. In contrast to the solely adaptive approach, HiPAS now optimizes the parallelization degree during the whole data transfer phase and based on a different algorithm. For mapping the described theoretical insights to our migration use case, we implemented an optimizer package, which predicts the optimal amount of parallel running jobs for the upcoming period. In this manner, we developed self-adaptive software, for which several definitions exist.

The anticipation-based version of HiPAS complies with a widely referenced definition [16], which was provided by the Defense Advanced Research Projects Agency (DARPA) in an Agency Announcement of 1997: "Self-adaptive software evaluates its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible" [17]. In particular, we address the identification of possible performance improvements. Another definition is given by Oreizy et al.: "Self-adaptive software modifies its own behavior in response to changes in its operating environment. By operating environment, we mean anything observable by the software system, such as end-user input, external hardware devices and sensors, or program instrumentation" [18]. The operating environment in our case is formed by the relevant components of the source and target database system identified in the "Observation" Paragraph.

Properties of self-adaptive software were introduced by the IBM autonomic computing initiative in 2001 [19]–[21]; these are known as eight self-properties. According to Salehie and Tahvildari, this classification serves as the de facto standard in this domain [16]. On major level the following self-properties exist:

- Self-configuring
- Self-healing
- Self-optimizing
- Self-protecting

Salehie and Tahvildari provided a list of research projects in the domain of self-adaptive software from academic and industrial sectors, and classified these projects according to their supported self-properties. It was stated that the majority of the analyzed projects focus on one or two of the known self-properties [16]. When developing HiPAS, we strongly focused on self-optimizing capabilities, which are demanded by customers of our domain. In addition, some self-healing properties for transfer job interruptions are supported. In the following paragraphs, we explain the chosen design decisions for HiPAS according to the dimensions of design space for adaptive software [22] and the optimizer's functionality by running through the phases of one adaption loop, also referred to as feedback loop or MAPE-K loop (see Section B-2).

1) Design Space

Design decisions about how the software will observe its environment and perform adaptions were defined by Brun et al. as design space for self-adaptive systems: "A design space is a set of decisions about an artifact, together with the choices for these decisions. […] A designer seeking to solve a problem may be guided by the design space, using it to

systematically identify required decisions, their alternatives, and their interactions" [22]. The following dimensions of design space were outlined by Brun et al.:

- Observation
- Representation
- Control
- Identification
- Enabling Adaption

Following this systematic approach, we describe the main decisions within these dimensions.

a) Oberservation

Observation is concerned with information about the external environment and the system itself, which needs to be observed by the system [5]. Therefore, Hinchey and Sterrit distinguish between environment-awareness and self-awareness [23], whereby environment-awareness is also called context-awareness [16], [24]. Based on the primarily intended property of self-optimization with respect to performance, we identified the following environmental components, which need to be observed:

- Storage system of the source database system
- Storage system of the target database system
- CPU resources of the source system's instances
- CPU resources of the target system's instances
- Memory resources of the source system's instances
- Memory resources of the target system's instances

The instances of both source and target database system use shared storage, but dedicated computing resources like CPU and memory, therefore, the monitored information must be analyzed on both global and instance-level.

To support system-awareness, the software needs to be aware of the overall migration progress, the number of currently running transfer jobs on each instance, and the status of any job at any time. For implementing self-healing capabilities, information about failed jobs needs to be stored along with respective log information in order to identify root causes and enable retry decisions.

Another important decision is related to the time of observation, which highly depends on domain knowledge about the expected frequency of environmental changes. According to our experiences with job modifications that need to be triggered manually, we implemented a timer, which triggers a cyclic observation every two minutes.

b) Representation

The identified information, which needs to be observed, is represented by performance values and monitored by the database management system. The DBMS stores these values inside performance views, which can be queried by HiPAS. The following values represent the necessary information to base adaption decisions on it:

- Concurrency events on target system
- Concurrency events on source system
- Average write time on target system
- Average read time on source system
- Average read time on target system
- Average write time on source system
- Redo log buffer size
- Available memory size

In order to be self-aware, HiPAS stores all status information regarding running and pending jobs in tables as described in Section V. These tables can be queried by adaption plugins such as the HiPAS optimizer.

c) Control

Control related decisions determine the involved control loops and their interaction, as well as the computation of enacted changes for each adaption [16]. According to [22], different patterns for interacting control loops exist. HiPAS is based on a hierarchical relationship between one master that is located on the target system and multiple slaves represented by instances of the DBMS. This architecture results in a variation of the Master/Slave pattern [22], where the master is responsible for global monitoring, analyzing and planning, and the slaves for instance-specific monitoring and executing (see Figure 10).

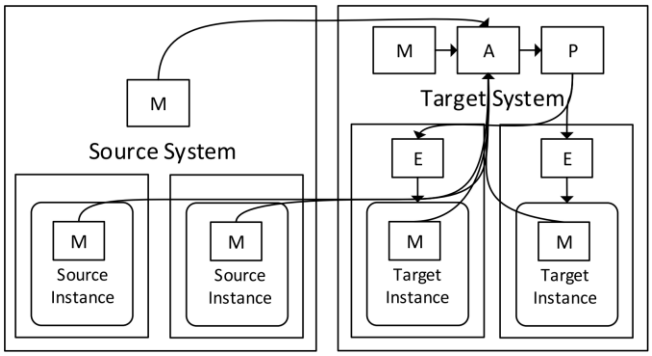

Figure 10. Variation of Master/Slave Control Pattern (adapted from [22])

The master aggregates and analyzes all collected information and then calculates a plan, which includes instance-specific commands. This plan is executed only against the target instances, since data is pulled by transfer jobs running on the target system. Reducing the amount of jobs will reduce the utilization on both target and source system. Hence, we decided for centralized control in a distributed system, which is comparably easy to realize, since all information that is required for performing adaptions is available at the target system through database links [22].

d) Identification and Enabling Adaption

The identification dimension deals with identifying instantiations of the self-adaptive system, which describes the system structure and behavior at a specific point in time [5]. HiPAS' system state is defined by the set of values for the observed performance attributes and the number of transfer jobs running on each target instance. Information about running jobs is stored inside a control table named "MIG_CONTROL" (see Figure 7). This table serves as a central interface for enabling adaptions because it allows plugins such as the HiPAS optimizer to access and change job information at runtime as described earlier in Section V. In the next paragraph, we explain how adaptions are performed by running through one adaption loop.

2) Adaption Loop

The relevant system performance attributes, which represent the information identified for observation, is continuously monitored by the DBMS across all involved database instances.

The optimizer analyzes the enumerated values and calculates a fail indicator, as well as the number of additionally possible jobs according to the measured available resources like memory size and disk utilization. In contrary, the fail indicator indicates possible bottlenecks and can prompt the optimizer to reduce the amount of currently running jobs. The introduced components form a feedback loop according to the MAPE-K (Monitor-Analyze-Plan-Execute-Knowledge) loop reference model developed by IBM [21] as shown in Figure 11.

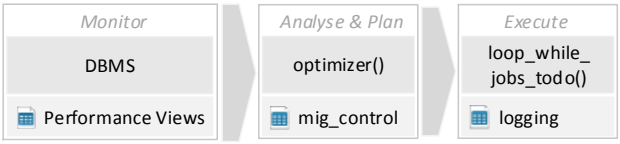

Figure 11. MAPE-K Based Adaptive Feedback Loop.

Typical indicators for possibly arising bottlenecks are increasing concurrency events while the redo log buffer size decreases. An important concurrency event, for instance, occurs when the high water mark of a segment needs to be increased, since new blocks are inserted into the same table by multiple and competing processes, this is known as high water mark enqueue contention [25].

If such a situation has been monitored, the optimizer will reduce the number of parallel jobs based on a high failure indicator. Whenever the optimizer acts, a log string is written to the logging table as in the following example:

```
"Prev Jobs: 40/ Jobs: 40 Max Jobs: 400 # Read Avg: 3.32(20-40) # Write Avg: 105.9(100-200) # R_Read Avg: .12(20-40) # R_Write Avg: .3(20-40) # R Fail Ind: 3 conc:3026(2607) redo:5720763732(5776886904) r_conc:5157(5069) # numjobs > 0 # Jobs being stopped: 0 # (Resource Overload) and numjobs > minjobs and jobs_being_stopped = 0 # Running: 20/Stopping: 5 on inst:1 # Running: 20/Stopping: 5 on inst:2"
```

In the above extracted example, 40 jobs are running in parallel. Due to increasing concurrency events, the optimizer detects a possible overload of the target system and decides to stop 5 running jobs on each instance. The jobs will terminate after they completed transferring their current objects. This is implemented by writing "stop" commands into the table "mig_control" (see Figure 7), which the procedure "loop_while_jobs_to_do()" will carry out. The next log string will start with the information "Prev Jobs: 40/ Jobs: 30" accordingly. Additionally, not only the overall amount of jobs is measured, but also the memory each server process allocates. This value highly depends on the data types of the currently transferred data. If too much memory is allocated, the number of jobs will be reduced as well. In order to avoid downward or upward spirals, e.g., due to the reducing redo log buffer size when stopping jobs, bottom lines and limits are defined. Hence, the optimizer decides on the basis of a branched search for indicating relations between the monitored information. Surely, these are only indicators not to be seen as evidence, so the algorithm follows a heuristic approach. In contrary to the solely adaptive approach and to a statically parallelized transfer, the optimizer is able to dynamically react to unexpected events and predict a possibly optimum level of system utilization during the whole migration process. In the following sections and for the evaluation, when mentioning the adaptive capabilities, we always refer to the anticipatory approach as it was performing more efficient during preliminary tests.

C. Time of Indexing

As previously termed as the "index problem", the right time for indexing the data depends on the combination of storage system performance and network bandwidth. If not manually parameterized inside the "param" table, HiPAS decides by means of test tables filled with random data and having indexes on multiple columns, if it creates the indexes before or after data loading. For the two possibilities of index creation, the time for performing the respective steps is measured and compared to each other. After comparing the two measurements, HiPAS updates the parameter "index_while_transfer" inside the "param" table autonomously by inserting "true" or "false". This test can be performed during a common migration test run on the actual system environment and excluded for the productive migration reusing the "param" table.

D. Transfer Order and Instance Affinity

The table "transfer_job_list" contains all objects, which need to be transferred to the target. When selecting the next object for transfer, this table needs to be ordered by blocks since large objects are preferred by HiPAS. Furthermore, an instance prefers table partitions of tables, which already have been started to be transferred by this instance. Accordingly, the next table or table range to be transferred is always selected as follow:

```
select * from transfer_job_list where status = 'PENDING' and object_type = 'TABLE' and (instance = 0 OR instance = sys_context('USERENV', 'INSTANCE')) order by instance desc, blocks desc, partition_name;
```

If an instance starts transferring a table range of a large table, it marks all other table ranges of the same table by inserting the instance number into all tuples related to this table. By this means, instances reserve tables in order to avoid loading the same table into data buffers of other instances. For this reason, instances prefer tuples marked by themselves and tuples not reserved by any other instance, which has been implemented by means of the above displayed "where clause". In addition, the actual block split range, defining the limit for the size of all table ranges, is identified partly adaptively. For a given maximum block split range, HiPAS calculates the optimal block split range by counting tables and their sizes resulting in an optimal ratio of a ranges size and its total count.

VIII. EVALUATION

The migration method has been tested in several customer environments with differently powerful server, storage systems and networks. Following the design science approach, HiPAS has been improved in multiple iterations based on test results.

A. Experiment Setup

For this paper, we set up a test environment consisting of a source and target system installed on physically separated virtual machines, each having 4 CPUs and 16 GB of main memory. Both the source and target database are real application cluster (RAC) environments running Oracle Database 11g Enterprise Edition Release 11.2.0.3.0. On each side two instances are available connected to the other side through a 1 Gigabit Ethernet. The source system reads from solid state drives and the target system writes on common SATA disks. For evaluation, we performed multiple test runs belonging to the following three different main tests:

(1) Function test with a 300 GB schema (Test A)
(2) Performance test with a 16 GB schema (Test B.1)
(3) Performance test with a 32 GB schema (Test B.2)

To create the different database schemata, we implemented a software package, which generates database schemata filled with random data and including all special cases we could imagine HiPAS to encounter at productive customer environments. By means of this software, we created different sized test schemata inside the source database for test migrations. For the function test (Test A), the schema included characteristics like foreign key constraints, a variety of character, numeric and binary data types, reference partitioning, indexes, table clusters, views, as well as different rights and roles. In this manner, we were able to test the compatibility of HiPAS with different data types, objects and complex data structures. The used schema has an overall size of 300 GB, which was large enough to analyze HiPAS adaptive behavior during the migration run. To compare HiPAS migration performance with the current Oracle standard migration tool for exports and imports "Data Pump" [26], [27], we reduced the size for being able to perform multiple test runs and to average out performance values across all performed runs. These performance focused migration runs are referred to as test B. After each migration, we fully deleted the migrated schema and rebooted the whole server in order to have the same initial cache situation for all runs. The results of all tests are shown subsequently.

B. Results

In the following the results of the function test (A) and the performance tests (B) are presented.

1) Function Test (Test A): As described in Section VI-A, "Test A" aims at analyzing HiPAS adaptive behavior and compatibility. We implemented a package, which compares the created target schema with the original source schema by counting rows and columns. We verified that all data objects were created inside the target schema successfully. The optimizer, providing the adaptive capabilities of HiPAS, writes log information whenever an adaption is needed. An example of such a single log string has been introduced in Section VII-B. Analyzing all tuples, written into the logging table during a migration run, leads to the migration process shown in Figure 12. The transfer started at 12:08 pm and ended at 12:47 pm. HiPAS transferred the created test schema, filled with 300 GB of random data, starting with 20 background jobs running in parallel meaning 10 jobs per instance, since HiPAS identified two available instances on the target system for job distribution. After 39 minutes, the transfer ended with a current total number of 116 parallel running jobs. The "block split range" was 25120 blocks, so, with a configured data block size of 8 KB, each job transferred a maximum amount of approximately 200 MB.

Tables, smaller than the split range, were not partitioned and transferred at the end of the migration, since large tables are preferred by the data selection algorithm. If a job transfers less data (small table), more parallel jobs are possible, so HiPAS raised the number of running jobs as the migration time goes by, which explains the slope of the graph shown in Figure 12.

In a different test using the same schema, we monitored the network interfaces in order to evaluate how the 1 Gigabit Ethernet is utilized by HiPAS. Figure 13 shows the number of kilobytes received and transmitted by one of the physical interfaces, which was monitored using the Linux command "sar".

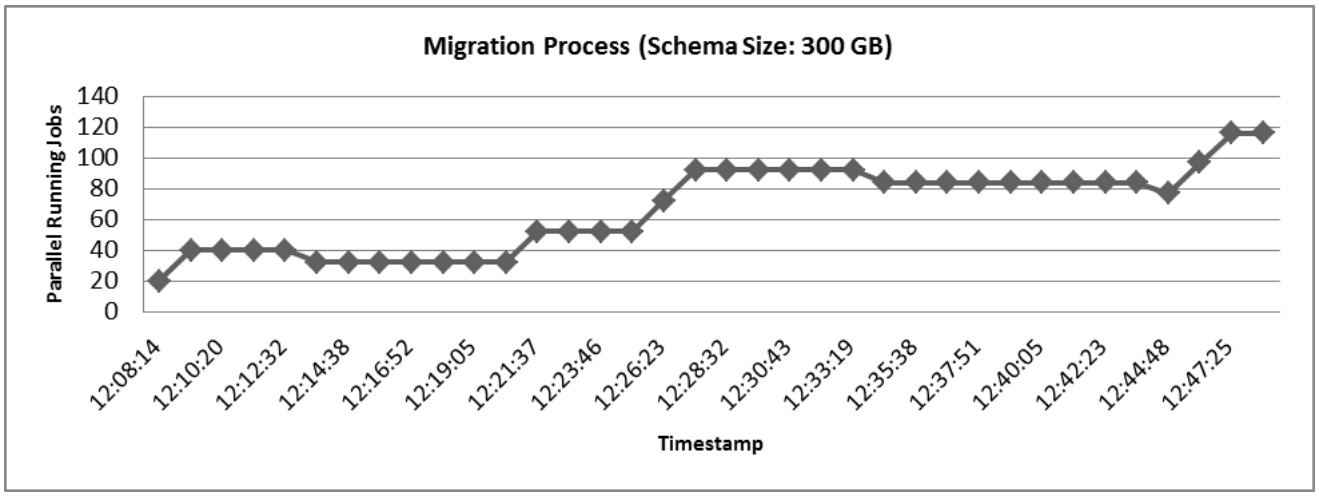

Figure 12. Adaptive Migration Process with HiPAS

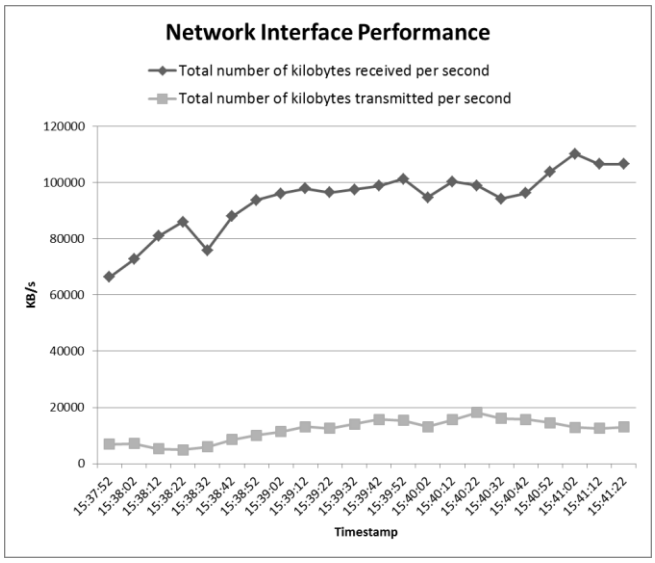

Figure 13. Network interface Performance (Excerpt)

During the monitored timeframe, the network interface, at its highest utilization, received up to 110138.34 KB per second. At that time 12849.31 KB were transmitted, resulting in a total amount of 122987.65 KB/s.

2) Performance Test (Test B.1): For the first performance test, we created a schema of 16 GB including the mentioned data types in Section VIII-A. The different test runs of test B.1 are described as follows:

(1) Migration by means of HiPAS adaptively and with enabled partitioning of large tables
(2) Migration by means of HiPAS with a static parallelization degree of 20 running jobs and enabled partitioning of large tables
(3) Migration by means of HiPAS with a static parallelization degree of 10 running jobs and enabled partitioning of large tables
(4) Migration by means of HiPAS with a static parallelization degree of 10 running jobs and disabled partitioning of large tables
(5) Migration by means of HiPAS without parallelization (sequential) and with disabled partitioning of large tables
(6) Migration by means of Oracle Data Pump

We performed the described test runs three times in order to compensate statistical outliers, possibly caused by uninfluenceable events of the database management system or the operating system. This was necessary because the test runs had to be performed successively to provide the same environment for all tested methods. Afterwards, we calculated for each method the average total duration of the three runs. The final result is shown in Figure 14. The small test schema of 16 GB has been transferred by HiPAS averagely within 11 minutes, enabling adaptive capabilities (more precisely "anticipation") and partitioning of large tables. Transferring the same schema by means of the Oracle tool Data Pump, using the number of available CPUs as the "parallel" parameter [26], took averagely 53 minutes, which means a deceleration of approximately 382% compared to HiPAS. Comparing the different HiPAS migration runs with each other, it can be stated that parallelizing in general

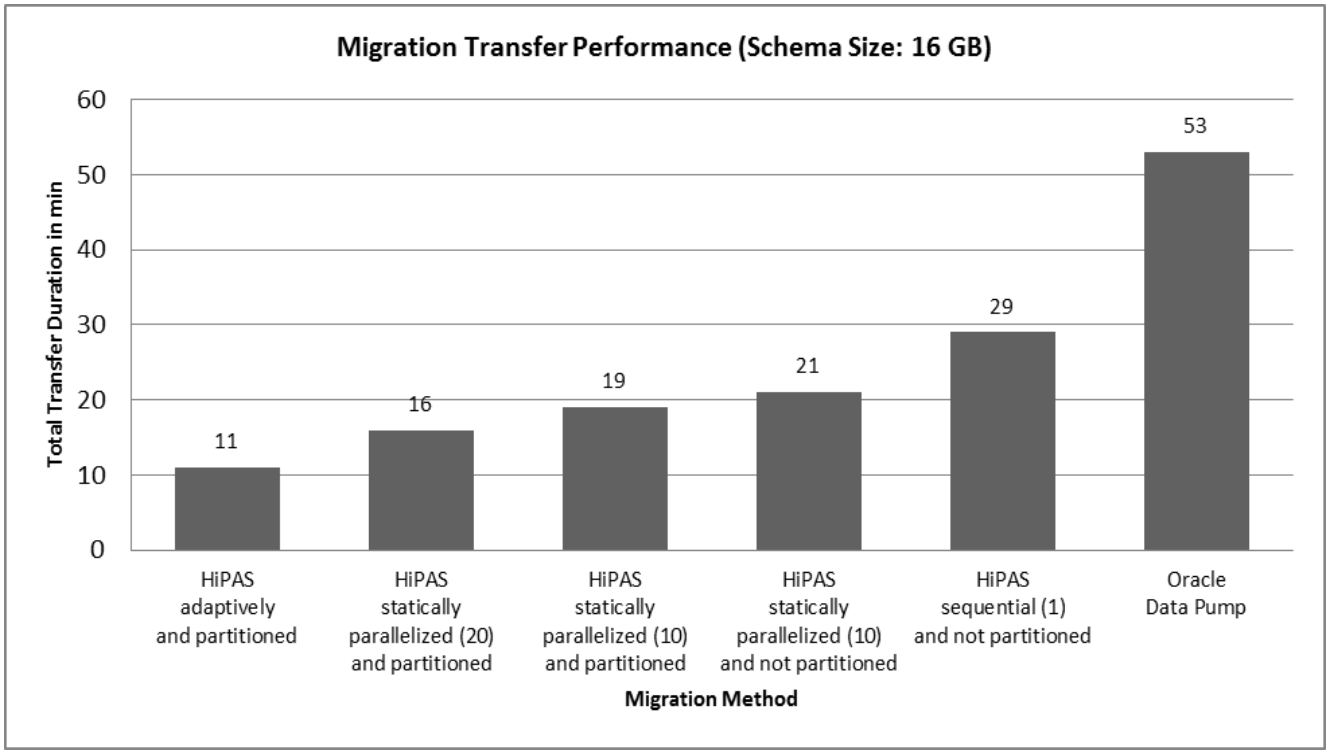

Figure 14. Transfer Performance for a 16GB Schema (B.1).

noticeably reduces the transfer duration, which is indicative for our assumption of utilizing the available resources more efficiently by parallelizing. Comparing test run 3 and 4 shows that partitioning large tables for the transfer barely improves the overall performance, since the partitioning feature was implemented to improve the flexibility of HiPAS when its optimizer needs to adapt quickly to changing resource availabilities. Thus, the adaptive migration run with enabled partitioning of large tables performed best in terms of downtime shortness.

3) Performance Test (Test B.2): In addition to the 16 GB schema, we performed the same test runs with a schema size of 32 GB to evaluate how the adaptive capabilities work for a longer period of transfer time. The static parallelized runs have been performed as well and showed results proportional to test B.1, so we excluded them from Figure 15.

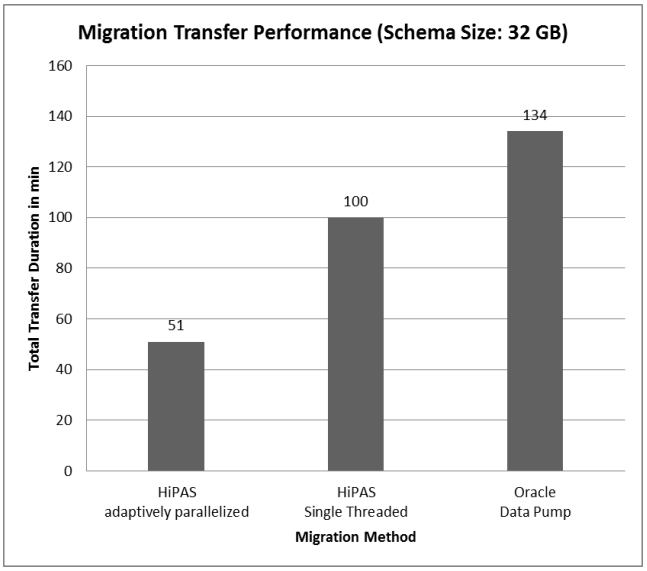

Figure 15. Transfer Performance for a 32GB Schema (B.2).

HiPAS, with enabled partitioning, adaptively transferred the schema within 51 minutes, compared to 2.23 hours needed by Data Pump, meaning this time HiPAS took 38% of Data Pump's transfer duration, whereas the single threaded configured HiPAS took about 75%. As a consequence, we assume that non-adaptive sequential and Data Pump migrations leave useful resources idle or need to be tuned manually. In addition to the introduced test runs for evaluation within the scope of this paper, we performed several further tests in customer environments achieving considerable results, especially for schemata storing large objects. In terms of network bandwidth, we reached transfer rates of 120 MB/s for each database link created on a 1 Gigabit Ethernet.

IX. Conclusion and Future Work

By designing and developing HiPAS, we applied adaptive software into the field of database migrations. We focused on self-optimization as the main adaptive property and implemented a system, which continuously optimizes the system utilization by adjusting the current amount of parallel workload. The observed environment is represented by monitoring information regarding the performance of the source and target database instances and their underlying storage systems. For optimizing the transfer phase, we state that implementing anticipatory capabilities into migration software using a MAPE-K feedback loop significantly improved the performance of migrations invoked on database layer, compared to a solely adaptive approach or non-adaptive migration software. Statically parallelized test runs did not adapt to changing utilization requirements, thus, performed less efficiently. The decisions taken in the design space for adaptive software and the paradigm of saving all migration metadata inside the database allows a highly reliable and transparent architecture, which supports an efficient interaction of all HiPAS migration components and the actual migration data. On the contrary, the implementation as a stored object leads to the disadvantage of having to develop separate implementations for different database systems. Therefore, we plan to build and evaluate further versions of HiPAS supporting different types of source and target systems. Another new version, we are working on, is intended to support online migrations, where the adaptive optimizer can be leveraged to utilize the source system up to a degree, which does not affect its availability and response time during productive use. Our results serve as a contribution for all practitioners in the field of database migrations, as well as researchers on self-adaptive software and their various fields of application.

Acknowledgment

We strongly like to thank all members of the Pasolfora Performance Research and Innovation Group (PPRG) for the support and possibility of performing the countless number of test and demo migrations during the development and evaluation of HiPAS. Furthermore, we thank Prof. Dr. Michael Höding of the Brandenburg University of Applied Sciences for giving scientific relevant input when mapping adaptive insights to the requirements of offline database migrations.

References

[1] H. Müller, A. Prusch, and S. Agel, "HiPAS: High Performance Adaptive Schema Migration – Evaluation of a Self-Optimizing Database Migration," in *DEPEND 2014, The Seventh International Conference on Dependability*, 2014, pp. 41–50.

[2] H. M. Sneed, E. Wof, and H. Heilmann, *Softwaremigration in Praxis*. dpunkt.verlag, 2010.

[3] M. L. Brodie and M. Stonebraker, *Migrating legacy systems: gateways, interfaces & the incremental approach*. Morgan Kaufmann Publishers Inc., 1995.

[4] J. Bisbal, D. Lawless, B. Wu, and J. Grimson, "Legacy information systems: Issues and directions," *IEEE software*, vol. 16, no. 5, pp. 103–111, 1999.

[5] R. De Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel, et al., "Software engineering for self-adaptive systems: A second research roadmap," in *Software*

Engineering for Self-Adaptive Systems II, Springer, 2013, pp. 1–32.

[6] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design Science in Information Systems Research," *MIS quarterly*, vol. 28, no. 1, pp. 75–105, 2004.

[7] M. Eastwood, J. Scaramella, K. Stolarski, and S. M., "Worldwide Server Market Revenues Decline -6.2% in Second Quarter as Market Demand Remains Weak, According to IDC." International Data Corporation, 2013 [Online]. Available: http://www.reuters.com/article/2013/08/28/ma-idc-idUSnBw276497a+100+BSW20130828 [Accessed: 29-May-2015]

[8] J. D. Little, "A proof for the queuing formula: L= λ W," *Operations research*, vol. 9, no. 3, pp. 383–387, 1961.

[9] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*, vol. 84. Prentice-Hall Englewood Cliffs, 1984.

[10] R. et al. Alves, *Information Storage and Management - Storing, Managing, and Protecting Digital Information*. EMC Education Services, 2009.

[11] R. M. Karp, *Reducibility Among Combinatorial Problems*. Springer, 1972.

[12] "Fraunhofer. Adaptive Systems. Fraunhofer Institute for Embedded Systems and Communication Technologies." [Online]. Available: http://www.esk.fraunhofer.de/de/kompetenzen/adaptive_systeme.html [Accessed: 29-May-2015]

[13] J. A. Martın H, J. de Lope, and D. Maravall, "Adaptation, anticipation and rationality in natural and artificial systems: computational paradigms mimicking nature," *Natural Computing*, vol. 8, no. 4, pp. 757–775, 2009.

[14] N. Wiener, E. Henze, and E. H. Serr, *Kybernetik*. Econ-Verlag Düsseldorf, 1963.

[15] R. Rosen, *Anticipatory Systems*. Springer, 2012.

[16] M. Salehie and L. Tahvildari, "Self-adaptive Software: Landscape and Research Challenges," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 4, no. 2, 2009.

[17] R. Laddaga, "Self-adaptive Software," Defense Advanced Research Projects Agency, 1997.

[18] P. Oreizy, M. M. Gorlick, R. N. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum, and A. L. Wolf, "An architecture-based approach to self-adaptive software," *IEEE Intelligent systems*, vol. 14, no. 3, pp. 54–62, 1999.

[19] P. Horn, "Autonomic Computing: IBM's Perspective on the State of Information Technology." IBM, 2001 [Online]. Available: http://people.scs.carleton.ca/ soma/biosec/readings/autonomic_computing.pdf [Accessed: 29-May-2015]

[20] "The 8 Elements." IBM [Online]. Available: http://www.personal.psu.edu/users/a/l/alw/autonomic/autonomic8.pdf [Accessed: 29-May-2015]

[21] "An Architectural Blueprint for Autonomic Computing," *IBM White Paper*. Citeseer, 2005 [Online]. Available: http://www-03.ibm.com/autonomic/pdfs/AC%20Blueprint%20White%20Paper%20V7.pdf [Accessed: 29-May-2015]

[22] Y. Brun, R. Desmarais, K. Geihs, M. Litoiu, A. Lopes, M. Shaw, and M. Smit, "A Design Space for Self-Adaptive Systems," in *Software Engineering for Self-adaptive Systems II*, Springer, 2013, pp. 33–50.

[23] M. G. Hinchey and R. Sterritt, "Self-Managing Software," *Computer*, vol. 39, no. 2, pp. 107–109, 2006.

[24] M. Parashar and S. Hariri, "Autonomic Computing: An overview," in *Unconventional Programming Paradigms*, Springer, 2005, pp. 257–269.

[25] "Enqueue: HW, Segment High Water Mark - Contention." Oracle, 2009 [Online]. Available: http://docs.oracle.com/cd/B16240_01/doc/doc.102/e16282/oracle_database_help/oracle_database_wait_bottlenecks_enqueue_hw_pct.html [Accessed: 29-May-2015]

[26] G. Claborn, W. Fisher, C. Palmer, J. Stenoish, and R. Swonger, "Data Pump in Oracle Database 11g Release 2: Foun-dation for Ultra High-Speed Data Movement Utilities." Oracle, 2010 [Online]. Available: http://download.oracle.com/otndocs/products/database/enterprise_edition/utilities/pdf/datapump11gr2_techover_1009.pdf [Accessed: 29-May-2015]

[27] K. Rich, "Oracle Database Utilities 11g Release 2." Oracle, 2014 [Online]. Available: http://docs.oracle.com/cloud/latest/db112/SUTIL.pdf [Accessed: 29-May-2015]