Minimizing Testing Overheads in Database Migration Lifecycle

Sangameshwar Patil[*], Sasanka Roy[†], John Augustine[‡],
Amanda Redlich[§], Sachin Lodha, Harrick Vin,
Anand Deshpande[¶], Mangesh Gharote, Ankit Mehrotra[‖]

Systems Research Lab
Tata Research Development and Design Center
Tata Consultancy Services
Pune, India
Email: {firstname.lastname}@tcs.com

Abstract

As part of their information management lifecycle, organizations periodically face the important and inevitable task of migrating databases from one software and hardware platform to another. Apart from ensuring accuracy and consistency, information managers of such organizations need to minimize the costs associated with the database migration process. Some of the migration tasks such as identification of database-dependent applications and the process of data replication are primarily governed by the technology used and have relatively predictable costs. On the other hand, the cost of testing and verification in the migration life-cycle depends significantly on the planning and dominates other costs.

We refer to this problem of optimizing testing cost as Database Migration Problem (DBMP). DBMP is challenging for enterprises because large number of databases and applications are affected and a variety of constraints force the migration to spread over multiple phases (or *migration waves*, as they are typically called in the industry parlance). After each migration wave, significant overheads arise for ensuring data integrity and testing the applications for correctness. We have observed that the industry practice for migration planning is based on the experience and intuition of a few software architects and administrators. This often results in delayed migration schedules and spiraling costs. However, a careful partitioning of databases into waves can lower the testing overheads and result in significant financial savings of several hundred thousand dollars. Surprisingly we did not find any literature that addresses this problem either. Hence, in this paper, we focus on minimizing the testing overheads of the data migration life-cycle.

We begin by showing that DBMP is NP-hard. Then, based on our real-life experience, we formulate DBMP so that it is amenable to formal, rigorous analysis and provide algorithms that cater to different scenarios, likely in practice. For small problem instances, we provide an optimal solution using integer linear programming. For larger instances, we formulate DBMP as a hyper-graph partitioning problem and use the well-known hMETIS tool for solving it. hMETIS provides good solutions quickly, but violates some of the constraints that are important in industry, for instance - the total size of databases packed in a wave cannot exceed the maximum wave size limit. Finally, to tackle such scenarios, we propose a new algorithm WAVE-FIT. Using experimental evaluation, we show that WAVE-FIT provides solutions comparable to hMETIS, in reasonable time without violating any constraint.

1 Introduction

Motivation: Databases are integral and critical components of IT infrastructure operated by modern enterprises. Over the past two decades, the size, number and diversity of databases used as part of the IT infrastructure has been increasing steadily. For example, we have come across quite a few global, top-tier financial service providers whose data-centers contain

[*] Corresponding author. Email: sangameshwar.patil@tcs.com

[†] Currently at Chennai Mathematical Institute, India. Email: sasanka.ro@gmail.com

[‡] Currently at Nanyang Technological University, Singapore. Email: jea@ics.uci.edu

[§] Dept. of Mathematics, Massachusetts Inst. of Tech., USA. Email: aredlich@mit.edu

[¶] Currently unaffiliated with TCS. Email: deshpande.am@gmail.com

[‖] Currently unaffiliated with TCS. Email: ankit.mehrotra@sas.com

International Conference on Management of Data
COMAD 2010, Nagpur, India, December 8–10, 2010

hundreds of databases (many versions and instances of every major database product, such as Oracle, DB2, SQL server, Sybase, Informix, MySQL and other Unix databases, available in the market). Moreover, these databases come in various sizes that range from 100 Megabytes to Terabytes and beyond.

The need to migrate databases from one platform to another arises due to a variety of reasons such as:

i. Functional and performance enhancements
ii. Expiry of vendor support for older versions
iii. Database and server consolidation for reducing the total cost of ownership
iv. Need for reconciliation of hardware and software platform diversity due to mergers and acquisitions among enterprises
v. Balancing workload across the devices

Newer versions of databases and newer/cheaper platforms offer pragmatic solutions for these requirements and often trigger such migration projects. Given such a scenario, most organizations periodically face the daunting task of migration: *Migrate a set of production databases from one software and hardware platform to another.*

Database Migration Process: Lets first understand the major steps involved in migration of a single database.

1. Basic setup and code updates
    a. Identify applications that are dependent on the databases being migrated.
    b. Update the applications and SQL code that needs to change due to migration to target database version and configuration.
    c. Setup and configure the test environment (hardware and software) corresponding to the target configuration.
2. Extract-Transform-Load (ETL) processing
    a. If there are changes to the source schema, prepare the target schema generation scripts.
    b. There could be an optional step of data cleansing and removing redundant data.
    c. Back up the source databases.
    d. Migrate the data: It could be either a relatively straightforward export/import or, for more complex cases, an elaborate ETL sequence of tasks.
3. Database configuration
    a. Configure user roles and access rights on migrated databases
    b. Create indexes on the target database
4. Testing and Verification
    a. Unit testing, System testing, Performance/volume testing and associated bug-fixing

    The migrated database is now ready to go-live in production environment.

Note that the technology for migrating or backing up an individual database is quite mature and well understood [17, 18]. In fact, most database vendors document the process for migrating individual databases [13, 16]. However, migrating a large collection of databases is complex. Apart from ensuring correctness and consistency of migration process, the IT plant operators must minimize the costs (time and effort) while honoring a variety of constraints. The constraining factors are typically related to business processes, the people and the technology involved; such as:

a. On a business day, the production databases of a global financial institution are updated from different timezones and are accessed for most part of the 24 hours. Hence, to minimize the impact on business availability, the migration activity is constrained to happen only on specific, non-business days (e.g. weekends). Additionally, due to the stringent performance SLAs, even the ETL tasks need to be carried out only during non-business periods.
b. Migration cannot happen during particular periods (e.g. blackout or brownout[1] days)
c. Availability of competent personnel, namely, database administrators as well as application testing teams
d. Availability of necessary hardware resources for the testing process
e. Vendor specific or application specific technological issues e.g. keyword mismatch, slightly different dialects of SQL due to extensions provided by vendor, incompatible stored procedures/functions.
f. Physical limits on the data transfer process due to bandwidth and latency etc.

Considering all such constraints, it is usually not possible to migrate all databases in a single time slot. Hence, IT plant operators are often required to develop a migration plan spread across multiple time slots. In each time slot, a subset of databases, referred to as a *wave*, is migrated from one platform to another.

The cost of migrating a given set of databases includes, but is not limited to:

C1. Identifying dependencies between databases being migrated and applications that access these databases
C2. Extracting data from source systems, carrying out transformations (if required) and finally loading it into target systems.

[1] scheduled total or partial power outages, typically for maintenance

C3. Data integrity verification on target systems
C4. Testing all the dependent applications after each migration wave

For a particular instance of database migration project, the costs C1, C2, and C3 in the above list are typically well-defined and predictable. Minimizing the cost of testing the dependent applications, however, is challenging as it depends on how the databases are segregated into multiple waves. Note that, after each database is migrated, all applications using this database must be tested. If databases used by an application are migrated in two different time-slots, the application would have to be tested twice. At this point, we would like to highlight the financial implications of application testing cost in practice, especially for the enterprises with large IT infrastructure. For instance, large global financial institutions have hundreds of databases being used by an even larger number of applications. Many of these applications are complex, business critical and are used from multiple locations. Testing such applications is a cost-intensive activity. Cost of a *single test run* for *each* application includes cost of setting up and configuring the application in test environment, running the actual test-cases, interpreting the results and fixing the errors, if any. On an average, these tasks take about 4-5 person weeks of effort per test run per application. This can translate into approximately $10000 per test run per application. Further, this cost increases with complexity of applications. Any re-testing would incur repeated expenditure of this effort. Even a small saving/improvement in terms of 10s of application re-tests amounts to savings of hundreds of thousands of dollars. Therefore it makes imminent practical sense to focus on application testing cost.

Contributions: Contributions of this paper are as outlined below:

- Identification of a recurrent problem of high importance to industry
- Proof that this problem is NP-hard (even when the database sizes are distributed uniformly)
- Characterization of the input data to clearly identify realms where optimal solution is usable and when heuristic solutions are necessary
- An optimal solution to the problem and identification of its limitations
- Identification of uses and limitations of a well-known hypergraph partitioning algorithm to solve real-life problem instances
- WAVE-FIT: A new algorithm that overcomes the limitation of hypergraph partitioning algorithm, and also performs comparably on the partition cost metric

Paper organization: In Section 2, we introduce the notations, then precisely define the problem and survey the related work. In Section 3, we show that the database migration problem is NP-hard by reducing the SET-PARTITION and hypergraph partitioning problems. In Section 4, we first present an optimal solution, but observe that this method is not scalable. Then, we map our problem to hypergraph partitioning problem and use a well known hypergraph partitioning tool hMETIS [14] to solve this problem. To overcome the limitations of using hMETIS (especially for industrial scenarios), in Section 4.3, we propose WAVE-FIT, our own algorithm. We analyze characteristics of the dataset from a real-life industrial database migration project and apply the proposed solutions to it. In Section 6, we show that the solutions proposed in this paper significantly improve upon the typical industrial approach. We also analyze pros and cons of different proposed solutions, and illustrate the efficacy of WAVE-FIT algorithm using experimental evaluation. The paper concludes in Section 7 with our observations about bounds on application testing cost and future work.

2 Notations and Problem Formulation

Since our prime interest is to minimize application re-testing, we formulate the *database migration problem* (DBMP) as follows. We are given a set D of databases which need to be migrated. Each database $d_i \in D$ has a size s_i. We are also given a set A of applications which use the databases in D. Each application $a_j \in A$ uses one or more databases from D; we denote this subset as $D_j \subset D$. Each database d_i when migrated will require all the applications depending on it to be tested; we denote this subset of A as A_i. One round of testing an application a_j incurs cost t_j. For each re-testing of a_j, this cost will be incurred again. Each application uses at least one database and each database is used by at least one application. Given this information, the problem is to partition the databases into a set of migration waves W such that the total application testing effort is minimized. Let $f(W)$ be the cost function that captures this application testing effort due to W. Further, we have to ensure that no wave is overloaded, that is, sum of sizes of all databases belonging to a migration wave must be less than a user-specified upper limit. We denote this upper limit as S_{max}.

As a brief illustration of above notation, consider a wave $w_k \subset D$. The set of applications that require testing due to migration of databases in w_k, is given by $A_{w_k} = \bigcup_{d_i \in w_k} A_i$. The application testing cost due to migration of databases in wave w_k is given by $f(w_k) = \sum_{j:a_j \in A_{w_k}} t_j$. In addition, the wave w_k is required to satisfy the wave-size constraint: $\sum_{i:d_i \in w_k} s_i \leq S_{max}$. Our ultimate goal is to partition D into $|W|$ waves such that f(W), the total application testing cost over

all waves is minimized.

In short, we are looking for the solution of the following Database Migration problem (DBMP):

Input: A set A of applications which use the databases in set D and a cost function for application testing.

Output: Partitions of the databases into a set of migration waves, $W = \{w_1, w_2, \cdots, w_{|W|}\}$.

Objective: Minimize total amount of application testing cost, that is:

$$\text{Minimize: } f(W) = \sum_{k=1}^{|W|} \sum_{j:a_j \in A_{w_k}} t_j$$

Constraint: Size of each migration wave should not exceed S_{max}, that is:

$$\sum_{i:d_i \in w_k} s_i \leq S_{max}$$

Related Work: Different types of data migration problems have been analyzed in literature. Problems related to replication of files within a storage system have been studied in [8, 4]. These types of problems typically arise in case of high demand content servers such as multimedia servers or web servers. Other type of large scale data migration occurs in scientific experiments that produce voluminous data where its long term availability is important. [9] provides an overview of such a system used for migrating data from physics related experiments, but focuses on system details, rather than the algorithmic aspects. Note that such data migration problems have been studied in different domains and different contexts. The primary objectives and constraints of the data migration problems that appear in literature are completely different.

While dealing with the problem of database migration, the database community has looked at the problems of schema matching [12]. [2] describes a useful tool developed for automatic schema matching. Migration of data from relational databases as source into object-based and XML databases as targets has been addressed in [10]. However, we could not find any work that examines the need for optimizing the testing overheads of database migration projects. To the best of our knowledge, this version of DBMP per se has not been studied before.

3 Hardness of the Database Migration Problem

First we present a simple proof that DBMP is NP-hard by proving SET-PARTITION $\leq_P$ DBMP. Recall that SET-PARTITION [3] seeks to partition a set I of non-negative integers into two subsets of equal sum.

Theorem 1 DBMP *is NP-Hard.*

Proof: Note that SET-PARTITION is NP-complete [3]. We construct the DBMP instance by creating a database of size s for each $s \in I$. We set the wave size limit $S_{max} = \frac{1}{2} \cdot \sum_{s \in I} s$. Clearly, if the number of waves required to migrate the databases is 2, then the set I can be partitioned into two subsets of equal sum. ∎

However, note that, if all the numbers in set I are same, then the corresponding SET-PARTITION problem is trivial to solve. Hence we further prove that DBMP is NP-hard even when all the databases are of same size. In essence, this reduction shows that DBMP is inherently hard even when the well-known hardness of packing databases in waves is discarded.

We reduce an instance of the hypergraph partitioning problem to an instance of DBMP with uniform database size (normalized to 1). In the simplest version of the hypergraph partitioning problem, we are given a hypergraph $H(V, E)$ and asked to partition the vertices into two equal sized subsets such that the number of edges that are cut by this partition is minimized. This problem is known to be NP-hard even when H is a graph [3], but we use hypergraphs because of their relevance in our context. In fact, a recent work [1] showed that even the graph partitioning problem cannot be approximated within a constant factor unless $P = NP$.

Theorem 2 DBMP *is NP-Hard even when all the databases are of same size.*

Proof: Any instance of a hypergraph partitioning problem can be reduced to an instance of the DBMP in the following manner. Let $H = (V, E)$ be the hypergraph in our instance of the hypergraph partitioning problem. Without loss of generality, we assume that $|V|$ is even. For each vertex $v_i \in V$ construct a database d_i of unit size in DBMP. For each edge $e_j \in E$, construct an application a_j in DBMP such that a_j uses the set of databases $\{d_i : v_i \in e_j\}$. Furthermore, these databases must be migrated in two waves of capacity $|V|/2$. We say that a hyperedge e_j has c_j cuts if the elements in e_j are placed in $c_j + 1$ different parts. It is easy to observe that database migration problem has a solution that needs $(c + |E|)$ application testings if and only if the hypergraph partitioning problem instance can be partitioned with a total (over all edges) of at most c edge cuts. Therefore, even DBMP with unit sized databases is NP-hard. ∎

4 Proposed Solutions

In this section, we first propose an integer linear programming (ILP) based optimal solution to the DBMP. Expectedly, the ILP based solution does not scale for large datasets. We then turn our attention to tackling the more likely industrial scenarios of large number of databases. Taking a cue from the reduction of hypergraph partitioning to DBMP, as presented in Section 3, we use the well-known hypergraph partitioning tool, hMETIS [14]. However, we observe that hMETIS cannot impose a strict upper bound on partition sizes. In fact, experimental results (refer Figure 6) have shown that S_{max}, an important constraint in DBMP, is significantly violated by hMETIS. To overcome this problem, finally, we propose our own algorithm WAVE-FIT that compares favourably with hMETIS on the cost metric and also honors S_{max}, the wave size limit constraint.

4.1 ILP Formulation

We define $a_{jk} = 1$ if application a_j needs testing due to the kth migration wave, and 0 otherwise. Given this notation, we can define our objective function as follows:

$$\text{Minimize: } \sum_{k=1}^{|W|} \sum_{j=1}^{|A|} a_{jk} t_j. \quad (1)$$

We define $d_{ik} = 1$ if database d_i is migrated in the kth wave, and 0 otherwise. Our constraints are:

$$\forall k, \sum_{i=1}^{|D|} s_i d_{ik} \leq S_{max}, \quad (2)$$

$$\forall i, \sum_{k=1}^{|W|} d_{ik} \geq 1. \quad (3)$$

$$\forall j, k, \sum_{i:d_i \in D_j} d_{ik} \leq |D_j| \cdot a_{jk}. \quad (4)$$

Constraint 2 limits the total size of databases allocated to a wave within its capacity S_{max}. Constraint 3 ensures that all the databases are migrated. Finally, Constraint 4 builds the dependency between application testing, given by the a_{jk} values, and the wave in which each database is migrated (given by the d_{ik} values). In other words, a_{jk} must be set to 1 if even one of the databases migrated in wave k is used by a_j. If the left hand side is greater than zero (i.e., a_j uses at least one of the databases in wave k), then clearly the inequality *can* hold only if $a_{jk} \geq 1$. Since the left hand side can never exceed $|D_j|$, the inequality *will* hold if $a_{jk} = 1$.

Our experimental results (Section 6.2.1) indicate that the ILP solution works well for very small cases, but does not scale well as the size of the input instances increase. With standard LP solver *lpsolve* [15], the execution did not complete even after several (3+) days for instances with a reasonably small number (40) of databases. The scalability problem is even more pronounced in the instances with uniform distribution in database sizes. These facts underscore the need for more scalable solutions to tackle more realistic industrial scenarios.

4.2 Hypergraph Based Solution

We have observed in Section 3 that DBMP can be viewed as a hypergraph partitioning problem. In this section, given an instance of DBMP, we provide a precise construction of an instance of the hypergraph partitioning problem. We construct the hypergraph $H(V, E)$ as follows: Databases form the vertices of H weighted by the size of the databases. For each application a_j, the vertices corresponding to the databases used by a_j form a hyperedge $e_j \in E$ with weight t_j. Notice that two (or more) applications using the same subset of databases induce two (or appropriately more) parallel hyperedges. If we want to carry out the database migration activity in $|W|$ waves, we have to construct $|W|$ groups of the databases to be migrated such that the application testing cost is minimized. This is exactly same as a $|W|$-way partitioning (with vertex weight balancing) of the hypergraph we have constructed above. From the above construction, it is clear that if we have an algorithm that solves the hypergraph partitioning problem and the solution needs c hyperedge cuts then migration problem has a solution that needs $(c + |A|)$ application testings for this specific partition. As the upper bound on wave size, S_{max} is fixed for a given instance of DBMP, we need to have a balanced hypergraph partitioning such that the sum of weights of the vertices in each partition is at most S_{max}.

After mapping DBMP to hypergraph partitioning problem, we use the well known hypergraph partitioning tool hMETIS [14] to solve our problem. But to the best of our knowledge, hMETIS cannot impose a strict upper bound of S_{max}. hMETIS may produce waves which are greater than S_{max}. Hence, we may encounter wave size violation, a situation in which the sum of the database sizes in a wave exceeds S_{max}. Note that such an upper bound on the partition size has not been a hard requirement in past work [4, 5, 6, 7], but it is an important requirement in DBMP. We use the term *Total Wave Size Violations* or $TWSV$ to denote the number of waves in which the sum of database sizes exceeds S_{max}. To minimize $TWSV$, we used the method shmetis of hMETIS, which takes a parameter called *UBfactor*, an integer value between 1 and 49 (both inclusive). When we set *UBfactor* to 1, the partitioning is more balanced than when set it higher. Given the number of desired partitions $|W|$, shmetis partitions a hypergraph into $|W|$ partitions using recursive bisection.

As presented in Section 6.2, we observed that there are significant number of instances (sometimes, even

upto 45%) when the wave size constraint is violated by solutions produced by hMETIS. Figure 6 summarizes the worst cases of $TWSV$ by hMETIS observed during experimental evaluation.

4.3 WAVE-FIT: An Algorithm for DBMP That Honors Wave-size Constraint

We now present WAVE-FIT, a polynomial time greedy algorithm which guarantees to give solutions without violating the maximum wave size constraint, S_{max}.

PHASE-1:

For each application $a_j \in A$, let D_j be set of databases used by a_j. Sort the set A in descending order, based on $|D_j|$. For subsequent computation, we use applications in this sorted order.
Create an empty set of temporary groups of databases, G = {}

for *each application* $a_j \in A$ **do**
- Create g = set of databases D_j used by application a_j
- Define $dbsizeof(g) = \sum_{i:d_i \in g} s_i$
- **if** $dbsizeof(g) \leq S_{max}$ **then**
  - $A_g = \{a_j\}$
  - **repeat**
    - Let $a_p \in A \backslash A_g$ be application such that D_p has maximum overlap with g ($i.e. D_p = max_{i:a_i \in A \backslash A_g} |D_i \bigcap g|$) and $D_p \bigcap g \neq \emptyset$
    - **if** $dbsizeof(g \bigcup D_p) \leq S_{max}$ **then** $g = g \bigcup D_p$ **end**
    - $A_g = A_g \bigcup a_p$
  - **until** $(dbsizeof(g) \leq S_{max})$;
  - $G = G \bigcup \{g\}$
- **else**
  - Arbitrarily split g into multiple smaller sized minimal number of subsets such that size of databases in each subset is less than or equal to S_{max}. These newly created subsets are inserted in G as temporary groups.
- **end**

end

PHASE-2

Form final migration waves by using bin packing for temporary groups $g \in G$ generated from PHASE-1. ;

Algorithm 1: WAVE-FIT

WAVE-FIT is designed to tackle the complex, many-to-many dependencies among the applications and databases, which are the crux of DBMP. The algorithm works in two phases. In the first phase, we partition D into a set of temporary groups of databases G such that, as far as possible, all the databases used by each application go into one of the groups without violating the wave size constraint S_{max}. Intuition behind this phase is the fact that a given application needs to be tested more than once only when the databases used by the applications are migrated as part of more than one migration wave. We try to put D_j, the set of all the databases used by an application $a_j \in A$ in a single temporary group, say g, if their combined size is within the wave size limit, i.e. $\sum_{i:d_i \in D_j} s_i \leq S_{max}$. When this condition is valid, D_j can go as part of single migration wave and application a_j needs to be tested only once for each database. However, note that even after putting D_j in a single group g, there is still spare capacity ($= S_{max} - \sum_{i:d_i \in g} s_i$) in g. One needs to use this spare capacity judiciously in order to reduce the overall application testing effort. We keep expanding g by choosing set D_p corresponding to $a_p \in A \backslash a_j$ such that D_p has maximum overlap with databases in g till $\sum_{i:d_i \in (g \bigcup D_p)} s_i \leq S_{max}$.

Due to large number of databases involved in migration activity in most cases, the number of groups, $|G|$ at the end of PHASE-1 will typically be larger than the intended number of migration waves. Given that the inter-relationship among databases have been exploited in PHASE-1, we use bin-packing to combine the temporary groups $g \in G$ to form W, the set of final migration waves in PHASE-2. Pseudocode of the algorithm is presented in the box titled Algorithm 1.

5 Real-Life Industrial Database Migration

The DBMP was motivated by a real-life database migration project involving 191 applications and 116 databases at a top-tier, global financial service provider. We applied the two algorithms: WAVE-FIT and hMETIS to this real-life industrial dataset. We found that these databases can be migrated in 4 waves with an application testing cost of 191 application testings reported both by WAVE-FIT, and by hMETIS. Note that the cost is optimal as each application is tested only once.

For detailed analysis of pros and cons of the proposed algorithms and their comparison with industrial practice, we now characterize the real-life input dataset from this database migration project. These characteristics are used to generate synthetic data in Section 6, required for experimental analysis.

5.1 Input Data Characteristics

To characterize the input dataset for a database migration project, we need information about:

1. Number of databases $|D|$ to be migrated,
2. Size of these databases,
3. Number of applications $|A|$ which use these databases, and
4. Application-to-database usage dependency information.

Out of the above, $|D|$ and $|A|$ are readily available as numbers. The database sizes can be easily modeled using appropriate probability distributions. However, quantifying the application-to-database dependencies is not so straight forward. We use graph theory for this purpose. We model the dependency information as a bipartite graph between the application nodes and database nodes. There is an edge between an application node and a database node in the bipartite graph if the corresponding application uses that database. We denote the set of edges in the bipartite graph as B. The degree of each application node is the number of database nodes it is connected to and vice-versa. The number of edges $|B|$ in the bipartite graph and the degree distributions of application and database nodes help us quantify the application-to-database dependency information.

For the real-life dataset, values of these parameters are: $|D| = 116$, $|A| = 191$ and $|B| = 204$. We found that, in this dataset, the database sizes (in Megabytes) follow lognormal distribution (mean = 6.18 and std.dev. = 2.79), as shown in Figure 1. We observe from Figures 2 and 3 that application and database degree distributions show power-law like behavior.

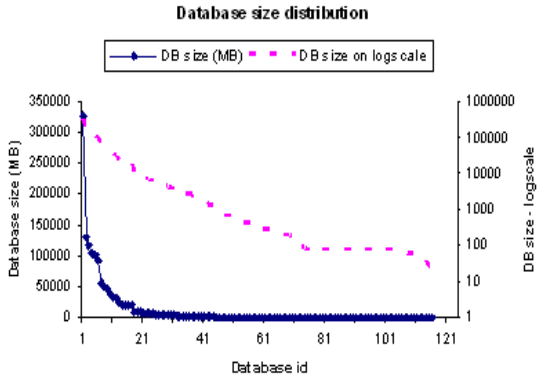

Figure 1: Database size distribution in the real-life data.

5.2 Available Options and Typical Industrial Approach to DBMP

Lets briefly discuss the options available and typical solution approaches used in industrial scenarios. The most obvious option is to migrate all the databases in a single wave. As mentioned in Section 1, this trivial solution is ruled out due to various constraints on the migration process. Another solution is to use techniques similar to standard bin-packing to group the databases into multiple migration waves. From the real-life data characteristics presented in earlier section, one can see that a large number of databases by a single application. Hence the bin-packing based

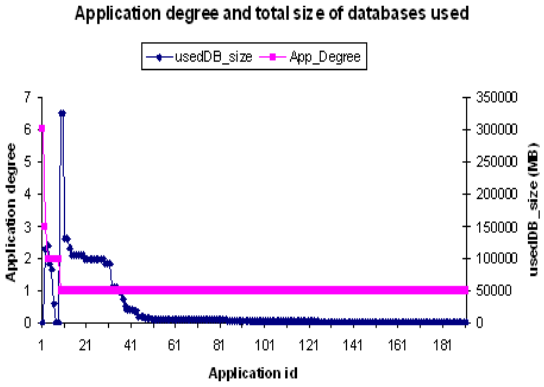

Figure 2: Application degree distribution

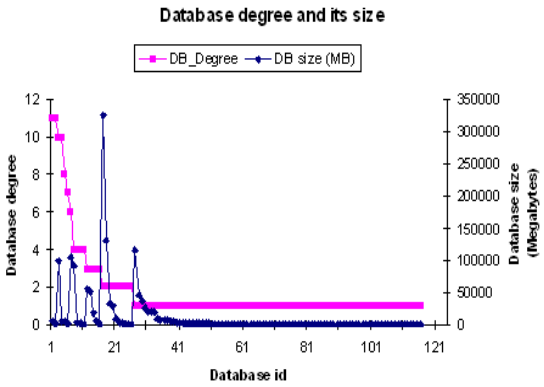

Figure 3: Database degree distribution

approach seems justified as well. Often, industry people do realise the importance of dependencies and try to put together the databases used by an application. However, the many-to-many nature of application to database dependencies, the large number of databases and applications involved, and the complexity of constraints take the problem beyond manual calculations or simplistic solutions. In the end, the pressing project deadlines force the industrial practitioner to approach the database migration planning using a close approximation of bin-packing technique or even worse, in absolutely ad-hoc manner.

6 Experimental Evaluation

In this section, we experimentally validate the need for more sophisticated solutions for DBMP. We also evaluate pros and cons of the solutions proposed in Section 4.

6.1 Synthetic Data Generation

We generate synthetic data for experimental analysis using the characteristics of real-life input dataset as discussed in Section 5.1. We use $|D|$ as the independent variable and then use $|A|$ and $|B|$ as multiples of $|D|$. Database sizes are generated by sam-

pling from the lognormal distribution (mean = 6.18 and std.dev. = 2.79), the same database size distribution as that of real-life data. However, to avoid getting biased by the dataset we have, we also need to evaluate the performance using another distribution. We take lognormal distribution to represent the case when the database size distribution is skewed. To analyze significantly different scenario, we generate another database size distribution by sampling the uniform distribution(min=10, max=500000). We evaluate the performance of the proposed solutions for both these database size distributions.

We capture the application-to-database dependencies by generating random bipartite graphs. We consider different scenarios as shown in Table 1. While deciding the $|A|/|D|$ and $|B|/|D|$ ratios, we take the following real-life observations into account: (a) in industrial organizations, the number of applications are typically larger than the number of databases, and (b) few applications use more than one database. Hence, based on these observations, we consider the scenarios when $|A|/|D|$ and $|B|/|D|$ ratios are lower and higher than the real-life dataset as well as similar to it.

Table 1: Experimentation Scenarios

	$\|A\|/\|D\|$	$\|B\|/\|D\|$
Experiment 1	1.1	1.25
Experiment 2	1.5	1.75
Experiment 3	1.75	2.0
Experiment 4	2.0	2.25
Experiment 5	2.5	2.8
Experiment 6	3.0	3.3

An important constraint for a database migration project is the maximum size of a wave, S_{max}. Note that there is an upper bound on S_{max} due to constraints such as physical hardware limits, availability of database administrators (DBAs) etc. For experimental evaluation, we take S_{max} to be 1.1 times the maximum database size for the scenario when database sizes follow lognormal distribution. This value is appropriate due to the skewed form of lognormal distribution. For the scenario when the database sizes follow uniform distribution, we let S_{max} to be twice the maximum database size.

We varied the independent variable $|D|$ as per the following list: $(5, 10, \ldots, 95, 100, 200, \ldots, 400)$. Granularity is smaller till $|D| <= 100$ to get more experimental observations for the ILP based solution (proposed in Section 4.1) as that particular solution does not scale for larger number of databases. To gain statistical confidence in the results, we have carried out 10 iterations of each scenario mentioned in Table 1 for each value of $|D|$. The results are presented in the next section.

6.2 Experimental Results

6.2.1 ILP-based Solution does not scale

To understand the practical limits of ILP, we implemented it using *lpsolve*, a standard open source LP solver. We conducted our experiment on a Linux server with a dual core dual processor Intel Xeon(TM) CPU running at 3.00GHz with 4GB RAM.

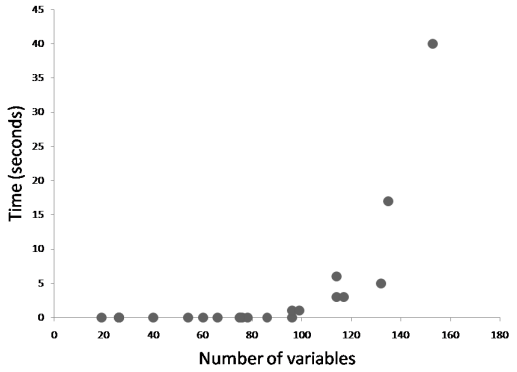

Figure 4: Running time of ILP solution w.r.t. number of 0-1 variables in the ILP formulation.

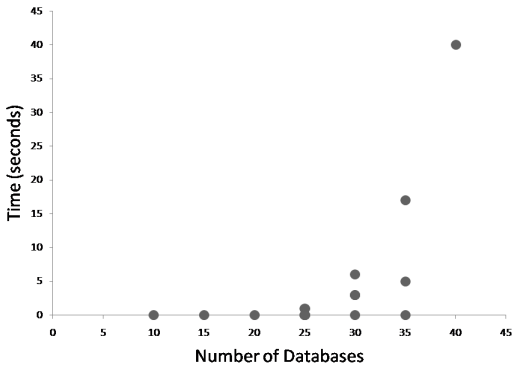

Figure 5: Running time of ILP solution w.r.t. number of databases

For our experiment, the instances were generated randomly and follow the lognormal distribution in database sizes. As described in Section 5.1, we varied the number of databases from 5 onward (in increments of five) and for each, we generated 20 instances. We see that there is a quick increase in the running time as the number of databases increases. In fact, in many instances, although the parameters were within the range shown in the graphs, the lpsolve execution did not complete even after 72 hours. Furthermore, we also noticed a sharp increase in the running time as the number of variables in the ILP formulation increased. The graph in Figures 4 and 5 show the running times

plotted against the number of databases in the input problem. We also experimented with uniform distribution in database sizes, but those instances were not viable even in very small instances ($\leq$ 15 databases). Regardless of the way we generate our instances, it is clear that beyond some small threshold, the ILP formulation is not a viable option.

6.2.2 Comparison of WAVE-FIT and hMETIS

We now compare the WAVE-FIT and hMETIS algorithms. Before proceeding to cost comparison, we first note that hMETIS runs faster than WAVE-FIT. However, WAVE-FIT, whose running time is polynomial in the size of the input, runs reasonably fast; even the larger problem instances were solved in less than two minutes. In industrial practice, this is acceptable time limit.

Observation 1: *Testing costs for solutions using hMETIS and WAVE-FIT are similar.*

We use the following notation to denote the application testing costs.

U_h = testing cost due to hMETIS (Uniform case)
U_w = testing cost due to WAVE-FIT (Uniform case)
L_h = testing cost due to hMETIS (Lognormal case)
L_w = testing cost due to WAVE-FIT (Lognormal case)

Table 2: Ratio of Testing Costs by hMETIS and WAVE-FIT

	U_h/U_w	L_h/L_w
Experiment 1	1.00	1.00
Experiment 2	1.00	0.99
Experiment 3	1.00	1.00
Experiment 4	1.00	1.00
Experiment 5	1.01	1.00
Experiment 6	1.01	1.00

We also explored whether the solutions are sensitive to the variation in application-to-database dependencies. From Table 2, we can see that for wide variety of application-database dependency scenarios and for both uniform and lognormal distributions, the solutions produced by hMETIS and WAVE-FIT are very much comparable.

Observation 2: *hMETIS violates the maximum wave size constraint, S_{max}.*

As mentioned in Section 4.2, limitation of solution produced by hMETIS is violation of wave size constraint. Figure 6 summarizes the violations observed experimentally. As discussed earlier, we use $TWSV$ to denote the total number of waves in which S_{max} - the wave size constraint is violated by hMETIS. We observe from Figure 6(b) that if the database sizes are distributed as per lognormal distribution, the average $TWSV$ is about 2.5%. However, the average $TWSV$ increased to 37% when the database sizes were sampled from uniform distribution (min=10, max=500000), as seen from Figure 6(a). Also note that, as the number of databases increase, $TWSV$ also increases significantly.

Another matter of concern was the extent to which the wave sizes generated by hMETIS exceeded the limit S_{max}. If we denote sum of databases sizes belonging to a wave as WS, then we found that $Max(WS)$ produced by hMETIS often significantly exceeded the S_{max}. This extent of violation can go as high as 30% of S_{max}. Such an high extent of violation may render the resulting solution unusable.

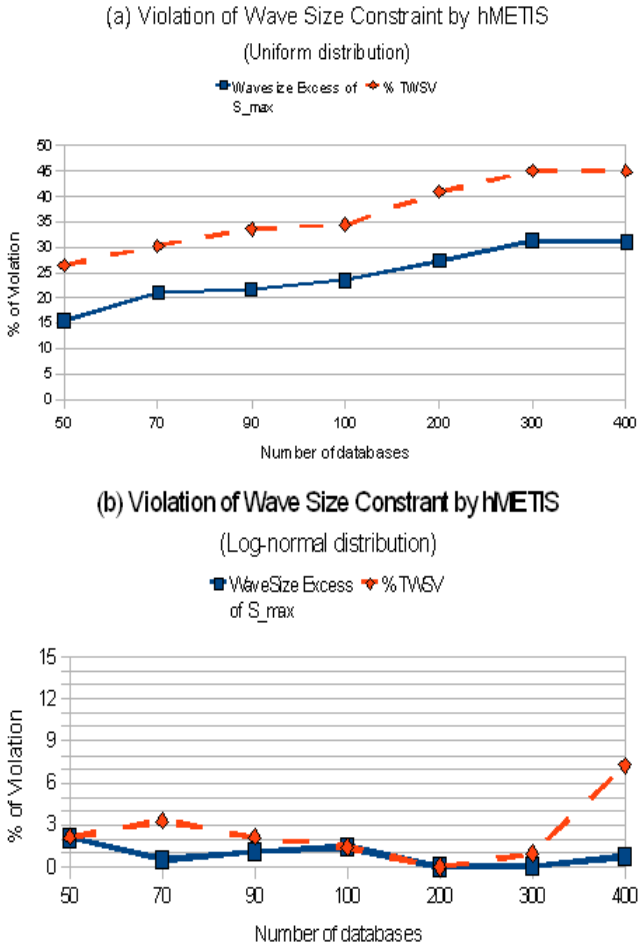

Figure 6: Analysis of wave size constraint violation by hMETIS: (a) Uniform distribution, (b) Lognormal distribution.

6.2.3 Comparison of WAVE-FIT with Typical Industrial Approach

In this section, we analyze the quality of solutions produced by WAVE-FIT with the typical industrial

approach. As discussed in Section 5.2, we use bin-packing approach as a close approximation of typical industrial approach. To evaluate the algorithms under different scenarios, we had to use the synthetically generated data as described in Section 6.1. To denote the application testing costs, we use notation similar to the notation in Section 6.2.2:

U_b = testing cost due to bin-packing (Uniform case)
L_b = testing cost due to bin-packing (Lognormal case)

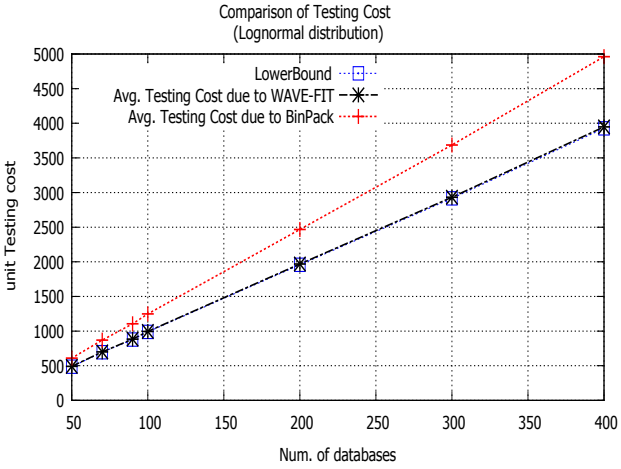

Figure 7: *Comparison of average application testing cost by WAVE-FIT and Bin-packing. The database sizes follow lognormal distribution (mean=6.18, sd=2.79).*

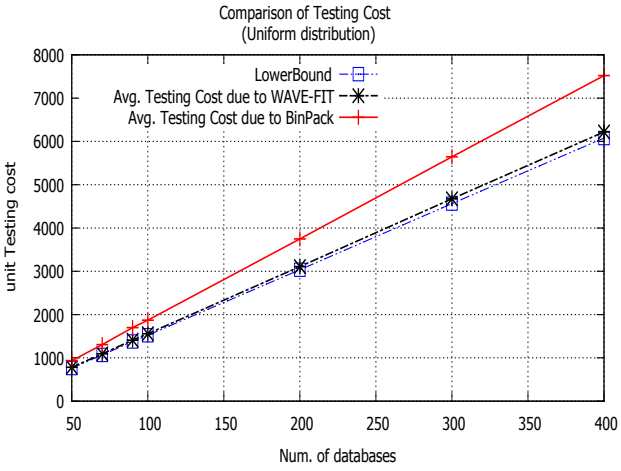

Figure 8: *Comparison of average application testing cost by WAVE-FIT and Bin-packing. The database sizes follow uniform distribution (min=10, max=500000).*

Figures 7 and 8 show the average application testing cost for solutions produced by bin-packing and WAVE-FIT. It also shows a feasible lower bound on application testing cost. To derive this lower bound, we have used the fact that each application needs to be tested at least once. So sum of individual application testing costs is a valid, non-trivial lower bound. We observe that the solution produced by WAVE-FIT (and similarly, by hMETIS) significantly improves upon the bin-packing based solution. From Table 2, we can see that for wide variety of application-database dependency scenarios, the solutions produced by WAVE-FIT outperforms the bin-packing based solution. Additionally, from Figures 7 and 8, note that the solutions produced by WAVE-FIT are very close the lower bound on testing cost. Thus we have experimental evidence that the solutions proposed in this paper not only improve the industry practice of planning database migration projects, but also provide solutions that are close to the optimal.

Table 3: Ratio of Testing Cost by Bin-packing and WAVE-FIT

	U_b/U_w	L_b/L_w
Experiment 1	1.23	1.28
Experiment 2	1.27	1.32
Experiment 3	1.22	1.27
Experiment 4	1.20	1.25
Experiment 5	1.19	1.24
Experiment 6	1.16	1.20

7 Conclusion And Future Work

In this paper, we have identified the challenges faced periodically by modern organizations in migrating large number of databases. Industry practice relies heavily on intuition and experience for minimizing the testing overheads in such projects. We have showed that database migration problem (DBMP) is NP-hard when we have to optimize the application testing cost. Based on our real-life experience, we have formulated DBMP to enable formal and rigorous analysis. We have identified three different likely scenarios and provided algorithms to tackle each one. For small problem instances, we provide an optimal solution using integer linear programming(ILP). However, ILP suffers from scalability problems. To tackle large datasets encountered in practice, we tried to build upon the hypergraph based formulation of DBMP and used the well-known tool for this purpose, hMETIS. Use of hMETIS however has a limitation as it violates the wave size constraint; the total size of the databases it packs in waves sometimes exceeds the wave capacity. Hence, we have devised a polynomial time, easy to implement algorithm - WAVE-FIT that overcomes this limitation and provides solutions whose testing cost metric is comparable to the solutions obtained using hMETIS. We have successfully applied the techniques developed in this paper in real-life database migration projects.

Future Work: It is easy to see that for a given instance of DBMP, if there is only one application that uses n databases then the optimal solution for DBMP is equivalent to finding the optimal bin packing. Since there is no polynomial time exact $(\frac{3}{2} - \epsilon)$-approximation for bin packing (for any $\epsilon > 0$) [19], DBMP also does not have a polynomial time exact solution that is a $(\frac{3}{2} - \epsilon)$-approximation.

We have an interesting observation on our experimental results. In all our experiments, the number of application testings reported by WAVE-FIT is less than twice of the lower bound on the number of application testings. As part of future work, we are exploring if there are more efficient partitioning algorithms that exploit the characteristics of real-life input data and achieve better/provable approximation factor for special cases of DBMP (such as when DB sizes follow particular distribution e.g. lognormal, uniform etc.).

Another promising area of improvement could be the running time of our WAVE-FIT algorithm. Quite recently, [11] showed that any partition problem of hypergraphs has an O(n) time approximate partitioning algorithm and an efficient property tester. For the given experimental conditions, the observed running time of WAVE-FIT is quadratic. Though this is acceptable for the large and practically relevant instances of DBMP (less than 2 minutes for 700 databases), we believe that there is scope for further improvement by leveraging recent developments in hypergraph partitioning.

References

[1] K. Andreev and H. Räcke. Balanced graph partitioning. In *Proc. of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, 2004.

[2] C.Drumm, M. Schmitt, H. Do, and E. Rahm. Quickmig - automatic schema matching for data migration projects. In *Proc. of the Sixteenth Conference on Information and Knowledge Management-CIKM*, 2007.

[3] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, CA, 1979.

[4] J. Hall, J. Hartline, A. Karlin, J. Saia, and J. Wilkes. On algorithms for efficient data migration. In *Proc. of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 620–629, 2001.

[5] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20:359–392, 1995.

[6] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48:96–129, 1998.

[7] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal*, 49:291–308, 1970.

[8] S. Khuller, Y. Kim, and Y. Wan. Algorithms for data migration with cloning. In *Proc. of the symposium on Principles of database systems*, 2003.

[9] M. Lubeck, D. Geppert, K. Nienartowicz, M. Nowak, and A. Valassi. An overview of a large-scale data migration. In *Proc. of the NASA Goddard Conference on Mass Storage Systems and Technologies*, 2003.

[10] A. Maatuk, A. Ali, and N. Rossiter. Relational database migration : a perspective. In *Proc. of the Database and expert systems application (DEXA)*, 2008.

[11] A. Matsliah, E. Fischer, and A. Shapira. Approximate hypergraph partitioning and applications. In *Proc. of 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2007.

[12] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB JOURNAL*, 10, 2001.

[13] `http://download-uk.oracle.com/docs/cd/B10501_01/server.920/a96530/toc.htm`. Oracle online documentation - oracle9i database migration.

[14] `http://glaros.dtc.umn.edu/gkhome/metis/hmetis/overview`. hmetis - hypergraph partitioning tool.

[15] `http://lpsolve.sourceforge.net/`. lpsolve - linear programming solver.

[16] `http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.doc/doc/t0024286.htm`. Ibm db2 online documentation - migrating information management systems.

[17] `http://www.ispirer.com/products`. Sqlways - a product for database migration from ispirer systems.

[18] `http://www.swissql.com/database-migration-solution.html`. Swissql - data migration tool.

[19] V. V. Vazirani. *Approximation Algorithms*. Springer, March 2004.