# Support Multi-version Applications in SaaS via Progressive Schema Evolution

Jianfeng Yan [#], Bo Zhang [*] [1]

[#] *SAP Research Center, China*
*1001 Chenghui Road, Shanghai, China 201203*
jianfeng.yan@sap.com

[*] *University of Shanghai for Science and Technology*
*519 Jungong Road, Shanghai, China 200093*
zhangbo@fudan.edu.cn

***Abstract*— Update of applications in SaaS is expected to be a continuous efforts and cannot be done overnight or over the weekend. In such migration efforts, users are trained and shifted from a existing version to a new version successively. There is a long period of time when both versions of applications co-exist. Supporting two systems at the same time is not a cost efficient option and two systems may suffer from slow response time due to continuous synchronization between two systems. In this paper, we focus on how to enable progressive migration of multi-version applications in SaaS via evolving schema. Instead of maintain two systems, our solution is to maintain an intermediate schema that is optimized for mixed workloads for new and old applications. With a application migration schedule, an genetic algorithm is used to find out the more effective intermediated schema as well as migration paths and schedule. A key advantage of our approach is optimum performance during the long migration period while maintaining the same level of data movement required by the migration. We evaluated the proposed progressive migration approach on a TPCW workload and results validated its effectiveness of across a variety of scenarios; Experimental results demonstrate that our incremental migration proposed in this paper could bring about 200% performance gain as compared to the existing system.**

## I. Introduction

Update of applications in SaaS is expected to be up continuously: workloads for new version of application are incrementally applied along with the decreasing of those for old version. With such property, even during update process, systems can provide online services to users for both old version and new version of application with stable and acceptable response time. One challenge for this scenario is data migration. Along with the accumulation of new users, it is generally expected that the data can be migrated into an object schema smoothly. In order to do so, instead of *shutdown-migration-restart*, the database server should support several intermediate schemas, which are adaptive to the temporal workload distribution and data statistic.

Traditional approach for system update needs the planned downtime to accomplish the reconfiguration, data migration and application transformation, which is unacceptable for multi-version applications software systems in SaaS. Especially for the systems with large volume of data, the cost of data migration is too high to accept. It will take the system several days to complete the system update, meanwhile, the customer business will be blocked during such process.

Even for quick data migration, the incremental data migration is much friendly for users. The adaptive intermediate schemas designed by the business priority and schedule provide users the choice to update their business step by step. The risks for system update, even including the training cost, can be decreased dramatically.

In this paper, we propose the problem of progressive migration for system updating, which is illustrated by the scenario in Fig.1. The system update starts from the source schema, which supports high performance and quick response for old application users. The workload changes following the rule of decreasing users of old application and increasing users for new application. Accordingly, several intermediate schemas are created and data are migrated partially to fit for the better database performance and system response. At the end of system update, the schema of database achieves a stable object schema, and the old application terminates completely.

Previous industry and academic research efforts in this area concentrated on constructing the schema mapping between source schema and object schema automatically, and, some of them pay attention to eliminate the cost of data migration. Our approach for data migration is based on the definition of basic schema evolution operators and the cost estimation for them under different workload distribution and data statistic. In order to minimize the total migration cost, a heuristic rule and a genetic algorithm are proposed to construct the effective intermediated schemas.

To the best of our knowledge, our work is the first effort to provide an incremental data migration solution for *multi-version* software systems. The key contributions including:

- Identify the challenge of incremental data migration in the update process for large scale online software systems.
- Propose the basic operators to decompose the above process into several atomic steps.
- Provide a cost models for progressive migration which

[1] Work done while with SAP.

1084-4627/09 $25.00 © 2009 IEEE
DOI 10.1109/ICDE.2009.167

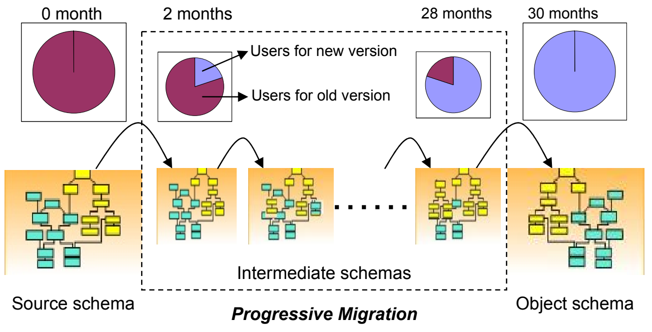
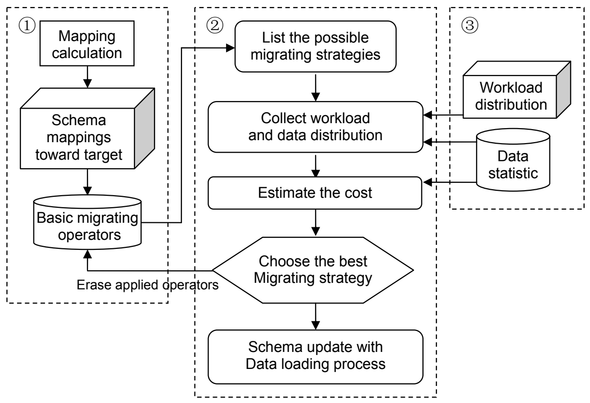


Fig. 1. A scenario of schema changing in the process of progressive migration. At the beginning of system update, all the workloads running on the source schema belong to old version. And then, several workloads for new version are added, which causes the database schema unsuitable to provide the best performance. Then, the schema is updated to an intermediate schema according to current system status (mainly including glossary workload distribution and data statistic). This action is repeated several times until all the users for old version withdraw from the system.



①
Mapping calculation
Schema mappings toward target
Basic migrating operators
Erase applied operators
②
List the possible migrating strategies
Collect workload and data distribution
Estimate the cost
Choose the best Migrating strategy
Schema update with Data loading process
③
Workload distribution
Data statistic

Fig. 2. A framework for progressive schema migration. (1) A schema mapping calculation tool is used to find out all the mappings from source schema to object schema. (2) At one *migration point* which is the predefined time point to begin one step of progressive migration, the possible schema reorganizations are virtually listed by compositing the mappings which are not applied. After the one which has the best performance to fit for system status is chosen, the new intermediate schema is physically created in database instance and then following by data loading process. After that, the mappings what have been used are erased. (3) Maintain the data to describe the system status. Workload distribution is counted by a collector or predefined by customers, and data statistic is refreshed by DBMS in real time.

chooses the best intermediate schema according to the status of current snapshot

The rest of the paper is organized as following: Section II presents the problem addressed in this paper. Section III describes our new automated, dynamic data migration approach from source schema to object schema. Section IV gives our experimental results. Section V summarizes the related work and compares them with our approach. And finally, Section VI concludes this paper.

## II. Scope of Problem

Figure 2 shows the framework of our approach. The key components of the system consist of searching the basic operator set and finding the best intermediate schema.

Previous researches [1] [2] provide us many approaches to construct the schema mapping between two schemas. Based on the generated schema mapping set, it is easy to construct the corresponding set of migration operators. The source schema could be migrated to object schema step by step by progressively applying these operators once and only once.

In order to choose the proper intermediate schema for current workload distribution and data statistic, the workload cost on this schema in current snapshot should be estimated. Such functionality is provided by SAP MaxDB, which provides the cost estimation of workload based on I/O throughput. We use these data in our experiment.

If the system is complicated, there are always numerous mappings between the source schema and the object schema. Furthermore, a big amount of possible operators will be generated for creating the intermediate schemas. Corresponding, the huge number of intermediate schemas should be estimated. The system have to make an adaptive tradeoff between choosing the best intermediate schema and saving the running time for cost estimation. Moreover, to get the higher performance of system in global performance analysis, a more complex process for cost estimation with predictive steps should also be executed. A Genetic Algorithm (GA) based approach is proposed in this paper to cope with this problem.

In this paper, we propose two models for selecting the proper intermediate schema from all possible candidates.

- **Local adaptive model** The proper intermediate schema is selected according to current status, that is, to make the system performance better than other candidate schemas.
- **Global adaptive model** The proper intermediate schema is selected according to current status and all possible future intermediate schema. This is to ensure that the strategy selected archives best performance in global way.

After selecting one proper intermediate schema in a migration point, the previous schema should be update to it and data should also be loading into it. Usually, only a part of choosing schema is different from previous one, it is reasonable to block only the workloads which need access the partial schema.

Another necessary component is **query rewriter**, which rewrites the mixed queries both of old version and new version to adapt with the intermediate schema. Many previous works were related to this problem[3] [4]. Here, we just figure out a straightforward method which is to find out the identifications from original query data access units, such as tables and attributes. Then, rewrite them for intermediate schema. During this, the schema mapping from previous schema to current one is necessary to be maintained. This process may spend extra system resource. But with the predictive of intermediate schemas, the queries can be rewritten in advance of runtime environment, and then the system performance is not affected anymore.

## III. Design Approach

Here we describe the basic operators, adaptive models and cost estimation algorithms for our progressive schema migration approach in framework above.

### A. Basic Migration Operators

*1)* ***Creating Table****:* This operator is used to add new columns into one existing schema. Functional dependency which describe the relationship between new columns with existing data should be predefined before applying this operator. For example, if source schema contains a table to describe the information of books, and the abstract of each book should be added as a new column in object schema, then a new column with type of string value will be added by creating a new table. For example, if it is reasonable to apply the functional dependency as the key *bookID* of *book* table to *abstract* column, a new table should be created with two column *bookID* and *abstract*. Figure 3 shows the corresponding schema mapping of creating table.

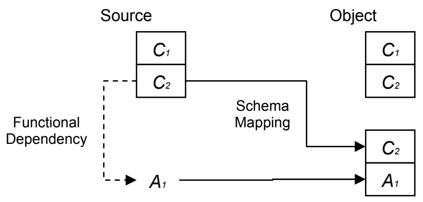


Fig. 3. **Creating Table Operator**. According to the functional dependency from $C_2$ to new information $A_1$, a new table with columns $C_2$ and $A_1$ is created.

No operator is defined for deleting table, because it is generally hard to predicate whether some information is not necessary before the migration process is finished. So, after progressive migration is finished, an additional work is to delete all the unnecessary tables.

*2)* ***Combining Table****:* This operator is used to combine two tables in source schema into one table. The reference to determine the relationship between these tables should be predefined before applying this operator. For example, if source schema contains table *book* for the information of books, and table *author* for the information of book authors, then these two tables could be combined into table *glossary* for presenting all the information related to each book. And, it is reasonable to employ the column *author name* in both tables as the reference. Figure 4 shows the corresponding schema mapping of combining table.

This operator is defined for combining just two tables into one table. When combining $n$ $(n \geq 2)$ tables, this operator should be called for $n - 1$ times.

*3)* ***Spliting Table****:* This operator is used to split one table in source schema into two tables. A reference to determine the relationship between the result tables should be created before applying this operator. For example, source schema contains *user* table for the information about users, and if we want to split the general information such as birthday and name

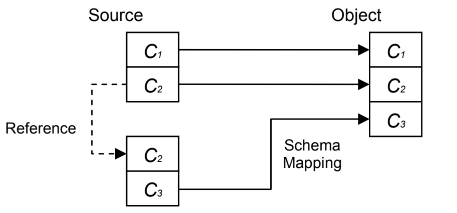


Fig. 4. **Combining Table Operator**. According to the reference between $C_2$, the two tables are combined.

from it, we can just split it into two tables, one for general information, the other for the rest information. It is reasonable to create the foreign key by user ID between two new tables as the new reference. Figure 5 shows the corresponding schema mapping of splitting table.

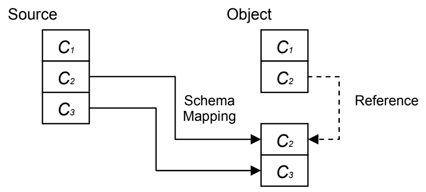


Fig. 5. **Splitting Table operator**. One table is split into two tables, and the reference of $C_2$ is created to maintain the dependency.

This operator is defined to split one table into just two tables. When splitting one table into $n$ $(n \geq 2)$ tables, this operator should be called for $n - 1$ times.

### B. Local Adaptive Model

In this section, we present the model of progressive migration to adapt local system status. The main idea of this approach is to select the proper intermediate schema just according to the cost estimation of current system status. It can be regarded as the method how to find the best intermediate schema for one snapshot among several migration points.

*1) Operator Set Calculation:* From the schema mapping relationship, the minimal basic operator set can be peeled off by applying a straightforward method as below.

- Each new column, which should be added into object schema, corresponds to one create table operator.
- Each table in object schema, which has two mappings from two tables in source schema, corresponds to one combining table operator.
- Each table in source schema, which has mappings to tables in object schema, corresponds to one splitting table operator.

*2) Schema Evaluation Based on Operator Set:* According to the schema mapping relationship, it is obviously that, by applying each operator in operator set once, the source schema could evolve into object schema. Here, the problem is what operators and how many operators should be applied at one migration point to evolve to a proper new intermediate schema. From a certain snapshot, all the different selections deduce

**Algorithm 1** Best operator set selecting

```
1: FUNCTION OpSet Select
2: IN: $W$, workload distribution
3: IN: $D$, data statistic
4: IN: $S$, current schema
5: IN: $OpSet$, current operator set
6: OUT: $Applied$, the operators should be applied
7: BEGIN
8: Cost $Min$ := $\infty$
9: Set $Applied$ := $\emptyset$
10: Set $TempOps$
11: SuperSet $SuperOpSet$
12: $SuperOpSet$ := PowerSet($OpSet$)
13: while ($SuperOpSet$ is not $\emptyset$) do
14:    $TempOps$ := PickOneSet($SuperOpSet$)
15:    $TempSchema$ := ApplyOperators($S$, $TempOps$)
16:    $TempCost$ := CostEstimate($TempSchema$, $W$, $D$)
17:    if ($Min \geq TempCost$) then
18:       $Min$ := $TempCost$
19:       $Applied$ := $TempOps$
20:    end if
21:    Erase $TempOps$ from $SuperOpSet$
22: end while
23: return $Applied$
24: END
```

several different potential intermediate schemas for this snapshot. According to the cost estimation of them, one effective intermediate schema could be selected. Correspondingly, the applied operators are erased from operator set. And then, the system finishes the data loading and waits for next migration point until object schema achieves.

In each migration point, **LAA** *(Local Adaptive Algorithm)*, shown in Algorithm 1 is executed to construct a subset of operator set. Line 14 lists all possible sub operator subset for current snapshot. From line 17 to line 19, one of them is picked up and the cost of coresponding schema is estimated. A method to estimate the cost of schema is given in next subsection. Line 21 to Line 24 is used to record the operator subset to create the best intermediate schema. Line 26 erases the checked operator subset from superset of all operators. After checking all the operator subset, this funtion returns one for constructing the minimum cost intermediate schema.

By utilizing all the operators in the output set, the intermediate schema with the best performance for current query distribution and data statistic will be migrated. After schema migration and data loading, all the operators in this subset should be erased from original operator set, because each operator can be used only once.

Suppose that at one migration point, there is $m$ operators left in operator set. Since every subset of the operator set should be tested, we have $2^m$ possible intermediate schemas to estimate for one migrating snapshot. So, if we have $c$ migration points predefined and $n$ operators found from schema mappings, in the worst scenario, there are $c \times 2^n$ intermediate schemas to be estimated among the whole migration process. It is the scenario which the source schema is always the best intermediate one, no operators is applied in each migration point. But every time, the cost estimation should be applied for all possible intermediate schemas.

Even in the best scenario, there are still $2^n$ intermediate schemas to be estimated. It is cause by applied all operators in the migration point, that is, from the cost estimation for all possible intermediate schemas, the object schema which applies all the operators is the best one. Then, after that, the operator set is empty and schema evolution is also finished.

Obviously, the resulting combinatorial search against this solution space is in exponential size. As **LAA** is based on the exhausted search strategy, it is not proper to be used in complex schema mapping environment. In the next subsection, we describe our heuristic genetic search algorithm for finding the optimum intermediate schema.

*3) Cost Estimation for One Snapshot:* From the implementation point of view, cost estimation is much critical to determine the selection of schema. We provide a concept formula to express the cost estimation for one snapshot. The cost value for one snapshot S is shown in the following:

$$CostValue = C(ObjectSchema) - C(S)$$

The $CostValue$ is the performance benefit by changing the schema to current snapshot S. The $C(ObjectSchema)$ is the evaluation cost with workload on object schema, and the $C(S)$ is the evaluation cost on current snapshot S. With increase of $CostValue$, more benefit we can get from migration.

Both $C(ObjectSchema)$ and $C(S)$ can be with following formula. The $C_i$ is the cost of the $i_{th}$ query estimated by the query engine. The $F_i$ is the frequency of the $i_{th}$ query.

$$C(Schema) = \sum C_i F_i$$

## C. Global Adaptive Model

Last section gives the strategy to find out the best intermediate schema which is matching the system status in each migration point. But it is not the best solution for global migration process. In this section, a global optimization model with the forward scan is described.

*1) Estimation with forward scan:* Under the precondition that the trend of workload distribution and data statistic is predicted, a global optimization method can be applied. That is, in each migration point, calculating the performance information on all the possible future snapshots instead of only current on. (usually, it is reasonable, since customers always have the plan to define the migration process and proper migration points. For example, if the update of system is begun from financial department to sales one and than product one, a simple tool can be applied to count the trend of workloads by the business frequency of each department. And, if every weekend evening, the system can be blocked for update, it can be regarded as the migration points).

Actually, in the first migration point, the best strategy is selected by the idea above. The reason we still prefer to check the forward scan in several migration point is, that the predictive workload trend may not very precise. If we trust it in complex system update, such imprecision may be accumulated and affect the whole performance migration plan.

Suppose we have $c$ migration points and $n$ operators. In the worst scenario, $2^{c \times n}$ intermediate schemas should be estimated in the first migration point. And, the whole process has to check the number of $\sum_{i=0}^{c} 2^{i \times n}$ schemas. It is also the scenario which the source schema is always the best intermediate one, no operators is applied in each migration point.

To search this solution space, we use **GAA** *(Global Adaptive Algorithm)* which is based on general genetic algorithm (GA) search heuristic [5], [6] that finds a tradeoff between finding the high performance intermediate schema and minimizing the estimation of intermediate schemas. The **GAA**'s output is the subset of operators that should be used to create new intermediate schema.

*2) Implementation of the global adaptation with a genetic algorithm:* We note that we use a GA as a tool to find a solution in the combinatorial problem. Since we are trying to find an subset of basic operators, potential solutions can be naturally formed as a string of integers, a well-studied representation that allows us to leverage prior GA research in effective chromosome recombination (e.g. [7]). Furthermore, it is known that a GA provides a very good tradeoff between exploration of the solution space and exploitation of discovered maxima [6].

The method we present the chosen operators as string of integers in each migration point is described as following. The length of string is the number of all operators which are not applied yet. Each position in the string responds one operator in the operator set. Suppose we have $c$ migration points from current stage to the object schema, the integer $i$ in each position of string is limited in the range of $(0, c)$, which means, the corresponding operator should be applied in migration point $i$.

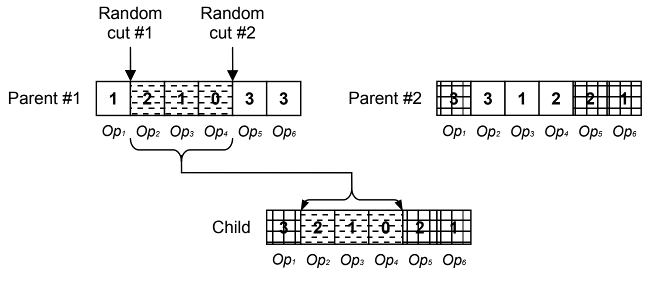

Random cut #1
Random cut #2
Parent #1
1 2 1 0 3 3
$Op_1$ $Op_2$ $Op_3$ $Op_4$ $Op_5$ $Op_6$
Parent #2
3 3 1 2 2 1
$Op_1$ $Op_2$ $Op_3$ $Op_4$ $Op_5$ $Op_6$
Child
3 2 1 0 2 1
$Op_1$ $Op_2$ $Op_3$ $Op_4$ $Op_5$ $Op_6$

Fig. 6. An example showing how the GA produces a child chromosome from parent chromosomes using a two-point crossover recombination. Each chromosome represents a permutation string. To maintain the requirement that all the integers must be unique in the string, the recombination first takes a contiguous subsection of the first parent (chosen to be the piece between two randomly chosen slices), places this subsection at the start of the child, and then picks up all remaining values in the second parent not included from the first parent..

As mentioned, the chromosomes are permutations of unique integers. Figure 6 shows a recombination of chromosomes

**Algorithm 2** GA evaluation function

```
1:  FUNCTION Select
2:  IN: $checkpoints$, the number of remained check points
3:  IN: $W(c)$, array of workload distribution
4:  IN: $D(c)$, array of data statistic
5:  IN: $S$, current schema
6:  IN: $OpArrange$, chromosome
7:  OUT: $Estimation$, cost of the intermediate schema
8:  BEGIN
9:  Schema $TempSchema$
10: Set $TempOps$
11: Cost $Estimation$
12: for $checkpoints \geq 0$ do
13:     $TempOps$ := Choose the operators from $OpArrange$
14:     $TempSchema$ := ApplyOperators($S$, $TempOps$)
15:     $Cost$ := CostEstimate($TempSchema$, $W(i)$, $D(i)$)
16:     $checkpoints$ := $checkpoints$ - 1
17:     $Estimation$ := $Estimation$ + $Cost$
18: end for
19: return $Estimation$
20: END
```

applied to two parents will produce a new child using a two-point crossover scheme [7]. Using this approach, a randomly chosen contiguous subsection of the first parent is copied to the child, and then all remaining items in the second parent (that have not already been taken from the first parent's subsection) are then copied to the child in order of appearance. The uni-chromosome mutation scheme chooses two random items from the chromosome and reverses the elements between them, inclusive. We will look at other recombination and mutation schemes from the GA community in the future.

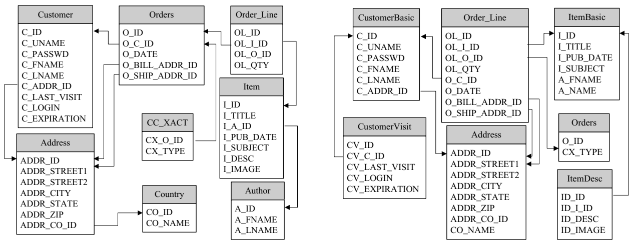
An important GA component is the evaluation function. Given a particular chromosome representing one selection of operators, the function deterministically calculates the cost estimation of intermediate schema by applying these operators with the current query distribution and data statistic. The evaluation function follows the pseudocode in Algorithm 2.

**For** loop in line 13 means the migration points. One number of the loop variable is mapping to one migration point. Line 14 is used to pickup the arrangement operators for tested migration point. For example, if current loop variable is 3, all operators which chromosome integer is just 3 will be picked up. Line 15 and 16 is used to create intermediate schema and estimate its cost. By line 18, the algorithm turns to check next migration point. Line 19 accumulates the glossary cost.

According to the operator arrangement in chromosome, **GAA** selects all the operators which should be applied in this stage and estimates the performance of system. After finishing the loop, the evaluation of one migration strategy is achieved.

## IV. Experimental Evaluation

In this paper, we use the schema in TPCW benchmark to test our method on MaxDB. TPCW specifies a book store for customers to browse and buy products from a website. Based on simplified schema in TPCW, we design source schema and object schema as shown in Figure 7.



(a) Source Schema (b) Object schema

Fig. 7. Schema Instance

## A. *Query Workload*

We test two sets of queries in our experiments, namely old queries and new queries. Each set includes ten different queries. The old queries corresponds to source schema with decreasing frequency, while new queries corresponds to object schema with increasing frequency. Notice the queries can be executed on corresponding schema directly, and both old and new queries are executed on intermediate schemas with a rewriting process during migration.

## B. *Situations for Comparison*

We compare three kinds of situations with same workload: **(a) Opt-Schema**: the source schema coexists with object schema at same time, and the old queries and new queries are executed on source schema and object schema respectively; **(b) Pro-Schema**: the schema is changed after a period of time according to both frequent workload and basic operators of migration, both old queries and new queries are rewritten and executed on intermediate schema during migration; **(c) Obj-Schema**: both the old and new queries are rewritten and executed on object schema.

Hence the performance of Obj-Schema gives the upper bound of three situations. Opt-Schema is the optimal situation for both old queries and new queries, it is the baseline of the three situations. Pro-Schema is the situation with incrementally schema evolution according to both workload performance and migration operators. Measuring the performance of Obj-Schema and Opt-Schema give us a better picture of how Pro-Schema performs in incremental migration.

## C. *Experimental Parameters*

In order to better test and understand the characteristics of our method, we design a set of parameters which include: data size, query frequency and migration frequency. The data size refers to the size of data base, we use the data size of 100MB and 1GB respectively.

With our method, the schema is changed in incremental manner, which begins from source schema, reaches to object schema after several migration points. We test three and five migration points respectively.

During the migration process, the frequency of old queries are decreasing and the new queries increasing. We use two kinds of query frequency in this paper. The first one is regular frequency, within which the old(new) queries are decreasing(increasing) with determinate rate. The second one is irregular frequency which means the frequency of old(new) queries decreases(increases) at random rate.

## D. *Performance Measurements*

We compare **Pro-Schema** with **Opt-Schema** and **Obj-Schema** with LAA and GAA methods in terms of the following two performance indicators: Phase-Cost and Overall-Cost.

The Phase-Cost is the query cost calculated on migration point, it collects the cost of queries between two migration points in the form of I/O number. The Overall-Cost is the sum of Phase-Cost, and it reflects the overall performance.

For three kinds of situations, we test Phase-Cost to examine the local optimization of LAA method. At each migration point, we multiply the cost of each query with its frequency, then by summing up the cost for each query. For testing the overall optimization with GAA method, we do the experiment with the migration points number from two to five.

## E. *Performance Results*

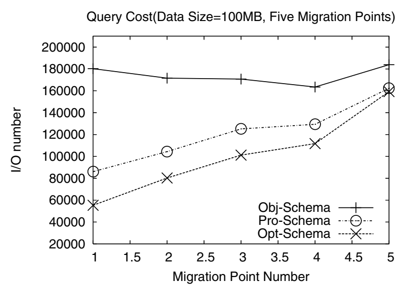
In this subsection, we study the performance of different situations by various parameters. We classify the test into two sets of irregular frequency and regular frequency. Within each set, we further varies the data size and migration frequency. This enables us to test our methods from different dimensions.

*1) Irregular Frequency:* In this subsection, we test the situation that the frequency of queries changes at random rate. In Figure 9, we list the workload frequency between different migration points. We only list the situation where there are five migration points as an example, and omit other situations for simplicity.

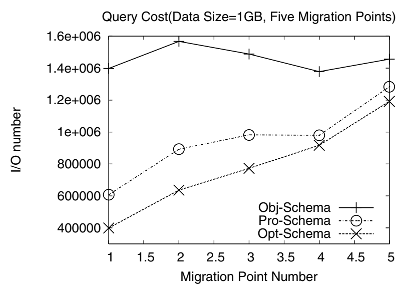
| **Workload** | **P0-P1** | **P1-P2** | **P2-P3** | **P3-P4** | **P4-P5** |
|---|---|---|---|---|---|
| O1 | 50 | 40 | 30 | 20 | 10 |
| O2 | 12 | 8 | 5 | 3 | 2 |
| O3 | 40 | 35 | 30 | 10 | 5 |
| O4 | 7 | 6 | 5 | 1 | 1 |
| O5 | 30 | 28 | 12 | 6 | 4 |
| O6 | 22 | 20 | 10 | 6 | 2 |
| O7 | 70 | 30 | 25 | 15 | 10 |
| O8 | 30 | 10 | 5 | 3 | 2 |
| O9 | 45 | 43 | 41 | 40 | 11 |
| O10 | 40 | 38 | 35 | 32 | 15 |
| | | | | | |
| **Workload** | **P0-P1** | **P1-P2** | **P2-P3** | **P3-P4** | **P4-P5** |
| N1 | 10 | 20 | 30 | 40 | 50 |
| N2 | 2 | 3 | 5 | 8 | 12 |
| N3 | 5 | 10 | 30 | 35 | 40 |
| N4 | 1 | 1 | 5 | 6 | 7 |
| N5 | 4 | 6 | 12 | 28 | 30 |
| N6 | 2 | 6 | 10 | 20 | 22 |
| N7 | 10 | 15 | 25 | 30 | 70 |
| N8 | 2 | 3 | 5 | 10 | 30 |
| N9 | 11 | 40 | 41 | 43 | 45 |
| N10 | 15 | 32 | 35 | 38 | 40 |

Fig. 9. Workload Frequency between Migration Points

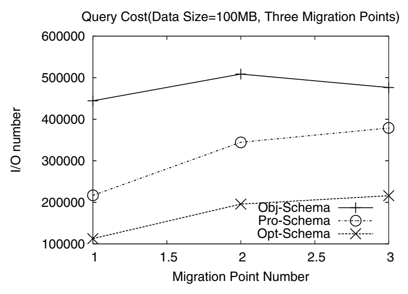
In Figure 8(a) and Figure 8(b), we test Phase-Cost of LAA method with five migration points, on the data size of 100MB and 1GB respectively. We can see that the Pro-Schema performs between the Obj-Schema and Opt-Schema,



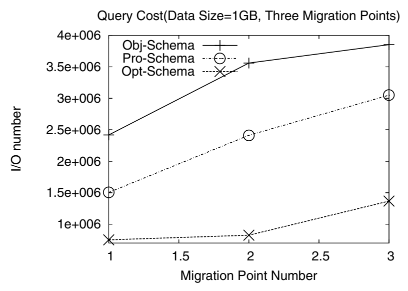
(a) Phase-Cost with LAA



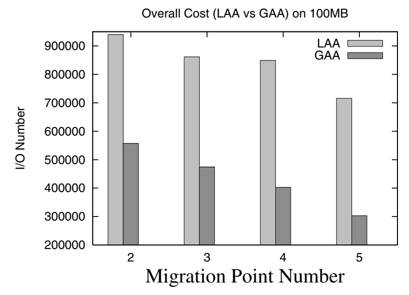
(b) Phase-Cost with LAA



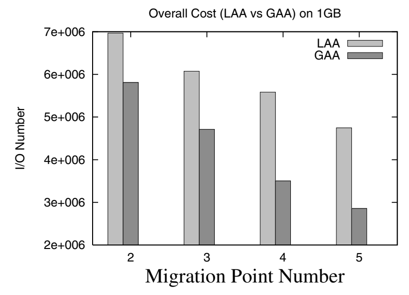
(c) Phase-Cost with LAA



(d) Phase-Cost with LAA



(e) Overall-Cost with LAA vs GAA



(f) Overall-Cost with LAA vs GAA

Fig. 8. Experiment Results

Opt-Schema provides the optimal scenario, and Pro-Schema achieves averagely 1.5 times performance gain over Obj-Schema. The advantages of Pro-Schema over Obj-Schema demonstrate that our incremental migration method can acquire an optimization at each migration point, this is because the Pro-Schema evolves according to the frequent queries at each migration point, so we get an optimized schema for executing queries after each incremental migration, and the optimized schema can improve the Phase-Cost between migration points.

In Figure 8(c) and Figure 8(d), we test Phase-Cost with LAA method with three migration points, on the data size of 100MB and 1GB respectively. The same results can be obtained since Obj-Schema and Opt-Schema define the bounds for Pro-Schema. For the above figures with five migration points, the Pro-Schema is more closer with Opt-Schema compared with the situations with three migration points. This is because with more migration points, the queries are partitioned into more groups, and each group of queries are between two migration points. Thus the changing rate of queries is more smooth, we can make an better exploration for workload distribution than the situation with less migration points, thus the Pro-Schema can be more sensitive to the workload changing, this enables the Pro-Schema progress to Opt-Schema smoothly and closely.

*2) regular Frequency:* In this subsection, we compare the Overall-Cost with LAA and GAA methods. We test the situations with migration points from two to five, and collect the Overall-Cost for each number of migration points. Also we consider the scenario that frequency of new(old) queries are increasing(decreasing) at determinate rate during the migration process, this enables the execution of GAA method.

In Figure 8(e) and Figure 8(f), we test the Overall-Cost with LAA and GAA methods on the data size of 100MB and 1GB respectively. We can see the Overall-Cost of both methods decreases with the increasing number of migration points. This benefit comes from the more refined analysis of workload. Also, GAA is more efficient than the LAA method. This is because GAA explores not only historical workload distribution, but also further workload at each migration point, aiming at changing schema according to more queries, and this enables the global optimization during migration process.

## V. Related Works

Data integration is a pervasive challenge faced in applications that need to query across multiple autonomous and heterogeneous data sources. Data integration is crucial in large enterprises that own a multitude of data sources [8]. Such projects typically involve two phases [9]. The first phase aims at establishing the schema and data mappings required for transforming schema and data. The second phase consists of developing and executing the corresponding schema and data transformations.

Several tools have been designed to assist the discovery of appropriate schema mappings [1], P. Bernstein and S. Melnik demonstrated a tool that circumvents the schema matching by doing it interactively. The tool suggests candidate matches for a selected schema element and allows convenient navigation between the candidates [10].

Y. Velegrakis, C. Yu and L. Popa provides a framework and a tool (ToMAS) for automatically adapting mappings as schemas evolve [11] [12]. The development of the corresponding schema and data transformations is usually an ad-hoc process that comprises the construction of complex programs and queries. Recently, such work has been in the road towards creating an industrial-strength tool [2]. The research of schema matching is still made by usage of new techniques [13] and domain knowledge [14].

The problem of data migration has also been extensively studied (e.g. [15] [16]) in the context of system integration, storage system, schema evolution and ETL tool chains. Various industrial tools have even been developed for the Data Migration process, such as components ProfileStage, QualityStage and DataStage in IBMs WebSphere Information Integration Suite [17], Data Fusion, which utilizes a domain specific language to conveniently model complex data transformations and provides integrated development environment [9]. Since it is always ineffective to move data from one data storage to another, and during the process, system is usually blocked for the service. Numerous algorithms have been proposed for effective or even online data migration [18] [19] [20] [21] [22]. However, none of them addresses the issue of progressive data migration.

With the help of the works above, our solution provides more advanced techniques to solve the problem in schema evolution and data migration. Schema matching related works implement the relationship between multi-version systems, effective and online data migration support system to provide service even during data loading process.

## VI. Conclusion and Further Work

Data migration is an important problem in many domains, especially in multi-version applications in SaaS. In this paper we present progressive migration, an incremental framework for migrating data for the scenario of system update. Based on the definition of basic schema mapping operators and the cost estimation for them under different workload distribution and data statistic, a framework of incremental evolution schema is presented. In order to provide optimized intermediate schemas during system updating, we employ both local and global optimized algorithms to implement the progressive migration.

To evaluate our system, we used TPCW benchmark [23] and a reasonable new version of TPCW-liked schema as the target schema in our experiments. Our results validate the utilization of incremental migration for system updating can bring a substantial performance improvement. Although we have focused on schema updating during our experiments, our approach can be applied to multi-version enterprise softwares and online data migration.

In the future, we plan to work on the optimization of schema design for more general purpose, not under the limitation on object schema driven, but the best physical design of schema for system workload distribution and data statistic.

## References

[1] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *The VLDB Journal*, vol. 10, no. 4, pp. 334–350, 2001.

[2] L. M. Haas, M. A. Hernandez, H. Ho, L. Popa, and M. Roth, "Clio grows up: From research prototype to industrial tool," in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, Baltimore, Maryland, 2005, pp. 805–810.

[3] H. Pirahesh, J. M. Hellerstein, and W. Hasan, "Extensible/rule based query rewrite optimization in starburst," in *Proceedings of the 1992 ACM SIGMOD international conference on Management of data*, San Diego, California, United States, 1992, pp. 39–48.

[4] har Krishnaprasad, Z. Liu, A. Manikutty, J. W. Warner, V. Arora, and S. Kotsovolos, "Query rewrite for xml in oracle xml db," in *Proceedings of the 30th VLDB Conference*, Toronto, Canada, 2004, pp. 1134–1145.

[5] J. Holland, Ed., *Adaptation in Natural and Artifcial Systems: An Introductory Analysis with Applications to Biology, Control, and Artifcial Intelligence*. MIT Press, 1992.

[6] D. Goldberg, Ed., *Genetic Algorithms in Search, Optimization and Machine Learning*. Kluwer Academic Publishers, 1989.

[7] L. Davis, "Job shop scheduling with genetic algorithms," in *Proceedings of 16th International Conference on Genetic Algorithms*, Mahwah, NJ, USA, 1985, pp. 136–140.

[8] A. Halevy, A. Rajaraman, and J. Ordille, "Data integration: The teenage years," in *Proceedings of the 32nd VLDB Conference*, Seoul, Korea, 2006, pp. 9–15.

[9] P. Carreira and H. Galhardas, "Efficient development of data migration transformations," in *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, Paris, France, 2004, pp. 915–916.

[10] P. A. Bernstein and S. Melnik, "Incremental schema matching," in *Proceedings of the 32nd VLDB Conference*, Seoul, Korea, 2006, pp. 1167–1170.

[11] Y. Velegrakis, R. J. Miller, and L. Popa, "Mapping adaptation under evolving schemas," in *Proceedings of the 29th VLDB Conference*, Berlin, Germany, 2003, pp. 584–595.

[12] C. Yu and L. Popa, "Semantic adaptation of schema mappings when schemas evolve," in *Proceedings of the 31th VLDB Conference*, Trondheim, Norway, 2005, pp. 1006–1016.

[13] H. Agrawal, G. Chafle, S. Goyal, S. Mittal, and S. Mukherjea, "An enhanced extractctransformcload system for migrating data in telecom billing," in *Proceedings of 24th International Conference on Data Engineering*, Cancun, Mexico, 2008, pp. 1277–1285.

[14] H. Elmeleegy, M. Ouzzani, and A. Elmagarmid, "Usage-based schema matching," in *Proceedings of 24th International Conference on Data Engineering*, Cancun, Mexico, 2008, pp. 20–29.

[15] J. Hall, J. Hartline, A. R. Karlin, J. Saia, and J. Wilkes, "On algorithms for efficient data migration," in *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, Washington, D.C., United States, 2001, pp. 620–629.

[16] K. Dasgupta, S. Ghosal, R. Jain, U. Sharma, and A. Verma, "Qosmig: Adaptive rate-controlled migration of bulk data in storage systems," in *Proceedings of 21st International Conference on Data Engineering*, Tokyo, Japan, 2005, pp. 816–827.

[17] "Information integration," IBM, http://www-306.ibm.com/software/data/integration/.

[18] S. Khuller, Y.-A. Kim, and Y.-C. Wan, "Algorithms for data migration with cloning," *SIAM Journal on Computing*, vol. 33, no. 2, pp. 448–461, 2004.

[19] C. Lu, G. A. Alvarez, and J. Wilkes, "Aqueduct: online data migration with performance guarantees," in *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, Monterey, CA, 2002, pp. 219–230.

[20] V. Sundaram, T. Wood, and P. Shenoy, "Efficient data migration in self-managing storage systems," in *Proceedings of the 2006 IEEE International Conference on Autonomic Computing*, Dublin, Ireland, 2006, pp. 297–300.

[21] G. Soundararajan and C. Amza, "Online data migration for autonomic provisioning of databases in dynamic content web," in *Proceedings of the 2005 conference of the Centre for Advanced Studies on Collaborative research*, Toranto, Ontario, Canada, 2005, pp. 268–282.

[22] J. Loland and S.-O. Hvasshovd, "Online, non-blocking relational schema changes," in *Proceedings of 10 International Conference on Extending Database Technology*, Munich, Germany, 2006, pp. 405–422.

[23] "The tpcw benchmark," http://www.tpc.org/tpcw.