# OLAP For Multicriteria Maintenance Scheduling

**Walter Cai**†, **David C. Anastasiu**‡, **Mingji Xia**§, **and Byron J. Gao**‡
†Memorial High School, Madison, WI, USA
‡Department of Computer Science, Texas State University - San Marcos, San Marcos, TX, USA
§Department of Computer Science, University of Wisconsin - Madison, Madison, WI, USA

**Abstract**— *Widely used in decision support systems, OLAP (Online Analytical Processing) technology facilitates interactive analysis of multi-dimensional data of varied granularities. In this paper, we demonstrate an interesting application of OLAP in solving multicriteria maintenance scheduling problems. Maintenance scheduling has many important applications, such as maintenance of inverted indexes for search engines and maintenance of extracted structures for unstructured data management systems. We introduce the design and implementation of* `Scube`*, an OLAP-based web service that allows users to navigate in a multi-dimensional simulation cube and comprehensively evaluate maintenance schedules. Based on the evaluation, the best trade-off schedule can be selected.*[1]

**Keywords:** OLAP, multicriteria maintenance scheduling, Pareto set, decision support

## 1. Introduction

Since its first introduction in the early 90's, OLAP (Online Analytical Processing) technology has evolved to be a must-have marketing weapon for business executives to systematically organize, understand, and use enterprise-wide data to make strategic decisions [14]. OLAP facilitates interactive analysis of multi-dimensional data of varied granularities. Most OLAP applications are business-oriented, including sales, financial and management reporting, marketing, business process management, budgeting and forecasting.

In this paper, we investigate an interesting application of OLAP in solving multicriteria maintenance scheduling problems. Maintenance scheduling is a non-typical category of scheduling problems, where jobs need to be periodically maintained over a long time span. It has many important applications, such as maintenance of inverted indexes for search engines, maintenance of extracted structures for unstructured data management systems, and maintenance of materialized views in data warehouses.

Due to the explosive growth of the web, crawling web pages has become increasingly challenging. As of ten years ago, the typical time for a major search engine to crawl one billion pages was more than one week, and it took up to six months for a new page to be indexed by popular search engines [5]. To substantially improve up-to-dateness of inverted indexes and save on network bandwidth, incremental crawling [5][4] was introduced, where crawlers estimate how often pages change, and then *schedule* pages for revisit based on their estimated change frequency and importance.

The challenge of managing unstructured data represents the largest opportunity since managing relational data [9]. An essential step in unstructured data management is to extract structures embedded in unstructured data, enabling structured queries like "how many citations did Jim Gray receive in 2009?" Web sources are highly dynamic, maintaining extracted structures is a labor intensive undertaking, and developing *scheduling* techniques to improve up-to-dateness and reduce maintenance cost is critical [10].

Such maintenance scheduling tasks are generally subject to communication and computation capacity constraints. While we want to keep the jobs maintained as in-time as possible, we also want to consume as few resources (e.g., workload) as possible. Taking account of several criteria enables us to propose more realistic solutions to the decision maker [21]. Therefore, maintenance scheduling problems are best modeled as multicriteria optimization problems, where multiple incommensurable objectives need to be optimized simultaneously.

At the abstract level, in a typical multicriteria maintenance scheduling problem, we have many jobs with different maintenance periods and maintenance costs. During each period, a job needs to be maintained at least once. We also have a daily workload capacity. The task is then to find a feasible schedule minimizing the long-term tardiness as well as workload.

Unlike single objective optimization problems, multi-objective or multicriteria problems do not have a single optimal solution. Instead, all non-dominated solutions form an optimal *Pareto set*, or *Pareto front*. There are two conceptual steps in solving multicriteria problems: search and decision making. Search refers to the optimization process in which the feasible set is sampled for Pareto solutions. Decision making refers to selecting a suitable compromise solution from the Pareto set. A human decision maker is required to make the often difficult trade-offs among conflicting objectives [13].

[1]This work originated from a student summer project Walter Cai did under the direction of Dr. Byron J. Gao at the University of Wisconsin - Madison.

Maintenance schedules are long-term schedules having a huge number of measure points, e.g., one tardiness value for each job with respect to each due day, and one workload value for each day. The performance of maintenance schedules cannot be well captured by a single measure (e.g., largest tardiness or workload) or at a single point (e.g., tardiness or workload for a particular day). For example, schedule $s_1$ with a bigger maximum tardiness may be much more favorable over schedule $s_2$ if $s_1$ performs better than $s_2$ in terms of tardiness for most due days of most jobs. Thus, maintenance schedules need to be comprehensively evaluated on their overall performance, which usually requires human intervention.

OLAP technology, facilitating interactive analysis of multi-dimensional data of varied granularities, is ideal in assisting decision makers to compare and evaluate the overall performance of alternative maintenance schedules by navigating the data cube. In this paper, we introduce the design and implementation of `Scube` (scheduling cube), an OLAP-based web service that allows users to comprehensively evaluate maintenance schedules.

**Outlines.** In Section 2, we discuss related work. In Section 3, we model multicriteria maintenance scheduling and introduce OLAP preliminaries. In Section 4, we discuss in detail the architecture and design of `Scube`. In Section 5, we discuss the implementation of the system. In Section 6, we present initial empirical evaluations of `Scube` and a case study. Section 7 concludes the paper.

## 2. Related Work

[5] studies how to refresh a local copy of an autonomous data source to maintain the copy fresh in the context of managing web data, in particular, inverted index maintenance. It defines freshness, which informally represents the fraction of up-to-date pages in the local collection.

Based on the same motivation of keeping inverted indexes up-to-date, [4][11][17] discuss designs of incremental crawlers with different topical foci, e.g., how web pages evolve over time, what distribution better models changes of web pages, and how to achieve scalability.

[6] theoretically studies a crawler scheduling problem minimizing the fraction of time that pages spend out of date, assuming Poisson page change processes and a general distribution for page access time.

Maintenance scheduling applications abound in unstructured data management systems [9][10]. In such systems, unstructured data are fetched periodically and structurized to enable SQL-like queries. It is one of the central issues to schedule the workflow wisely and keep the structured data as fresh as possible while minimizing communication and computation costs.

Scheduling theory first appeared in the mid 1950's. Scheduling concerns the allocation of limited resources to tasks over time and is normally formulated as optimization problems [18][7]. Maintenance scheduling is a non-typical scheduling problem. Instead of completion time, it concerns periodic in-time maintenance of jobs over a long time span. [1][3][19][2] theoretically study a maintenance scheduling formulation called "windows scheduling problem".

Multicriteria optimization has been studied for decades [12][8][20]. An excellent overview for multicriteria scheduling can be found in [21]. The majority of scheduling problems are single objective. Taking account of several criteria enables us to propose to the decision maker more realistic solutions [21]. Multicriteria problems do not have a single optimal solution and a decision maker is usually involved in the problem solving procedure [13].

OLAP technology has been widely used in enterprise decision support systems [14]. To our knowledge, we are the first to apply OLAP to multicriteria optimization. For a thorough coverage on OLAP, interested readers can refer to the many books dedicated to data warehousing and applications, e.g., [15][16].

## 3. Preliminaries

In this section, we briefly introduce OLAP preliminaries and a generic problem formulation for multicriteria maintenance scheduling.

**OLAP.** OLAP systems are built based on the multidimensional model, where the central concept is data cube. A *data cube* consists of a large set of numeric facts called measures that are categorized by dimensions. Hierarchical in nature, dimensions are the entities or perspectives with respect to which an organization wants to keep records. Data cubes are typically created from star schemas or snowflake schemas, with star schemas being more popular.

Typical OLAP operations include roll-up, drill-down, drill-across, drill-through, slice, dice, and pivot. There are also other statistical operations available such as ranking and computing moving averages and growth rates. These operations allow users to navigate a data cube along dimension hierarchies and view the cube from different perspectives.

In particular, roll-up summarizes data by climbing up a concept hierarchy of a dimension (or by dimension reduction) to have a "higher view", e.g., from a city view to a country view. Drill-down is the reverse of roll-up going from a higher level summary to a lower level summary. Slice performs a selection on one dimension of the cube, resulting in a subcube. Dice defines a subcube by performing a selection on two or more dimensions. Pivot changes the dimensional orientation and rotates the data axes in order to provide an alternative presentation of the data. Drill-across executes queries involving more than one fact table. Drill-through uses relational SQL facilities to drill through the bottom level of a data cube down to its back-end relational tables [14].

**Modeling of Multicriteria Maintenance Scheduling.** In a typical multicriteria maintenance scheduling problem, there are $n$ jobs to be maintained. Each job labeled $i$ needs to be maintained at least once every $w_i$ days and each maintenance costs $c_i$ workload. There is a daily workload capacity of $k$. The task is to find a feasible schedule minimizing the long-term tardiness as well as workload.

The measure *tardiness* is either 0 (maintained in time) or a positive number indicating the number of days after the due day of a maintenance. A *feasible* schedule is one satisfying all the given constraints. The notion of *day* here is symbolic, it can be time slot of any length.

We use a tuple $I = (k, (c_1, w_1), \ldots, (c_n, w_n))$ to denote an instance of a multicriteria maintenance scheduling problem. In $I$, each job $i$ has a *maintenance window* of size $w_i$ and maintenance cost of $c_i$.

Note that, real maintenance scheduling problems are almost always *dynamic*, where jobs come and go, and maintenance costs and window sizes change randomly. For example, web pages are created and deleted every day. Blogs would get updated more frequently during holiday seasons. Authors may receive higher citation rates after winning some major awards. Therefore in a dynamic maintenance scheduling instance $I$, the parameters $k$, $c_i$, and $w_i$ are not constants, but variables.

## 4. Architecture of Scube

In this section, we discuss in detail the design, architecture and workflow of `Scube`.

The architecture of `Scube` is shown in Figure 1. In the system, a dynamic multicriteria scheduling instance $I$ can either be automatically generated by the data generation module or provided by users. Some built-in schedulers in the scheduling module will then be applied to $I$ to generate a set of alternative schedules that approximate the Pareto set. Users can also apply external schedulers to generate schedules for $I$ and upload them. Then, from $I$ and the set of alternative schedules, the cube computation module computes a simulation cube. The cube navigation module allows users to navigate the cube and evaluate the schedules in three modes: SQL, CQL, and Graphic. Based on interactive and comprehensive evaluations, users can decide the best trade-off schedule for $I$.

**Data Generation Module.** `Scube` provides a synthetic data generator that generates dynamic maintenance scheduling instances. The instances can vary in size $n$, length, daily workload capacity $k$, distribution (Gaussian or Poisson) of maintenance cost $c_i$ and window size $w_i$. Since `Scube` performs a simulation calculating measures for each day, a valid instance must end at some finite length.

Dynamic factors (creation and deletion of jobs, change of parameters) are added randomly under a uniform or Poisson distribution.

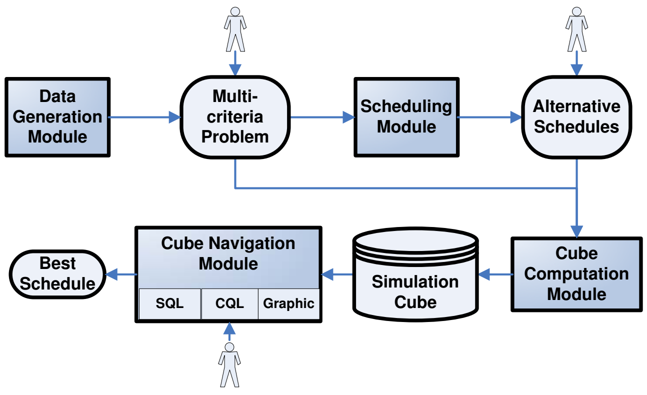

Fig. 1: Scube architecture.

`Scube` was designed to provide web services. Thus, it also allows users to upload their own scheduling instances conforming to a simple XML style format as described in the following.

**Multicriteria Problem.** In instance $I$, maintenance cost $c_i$ and window size $w_i$ can be updated at any specified day. A new job is created if it appears in $I$ with a new id. Job $i$ is deleted if $c_i$ becomes 0. The daily workload capacity $k$ can be modified as well. $I$ ends on the day when all jobs are deleted.

An example instance $I$ is given in Figure 2 (left hand side), demonstrating the required simple format. On day 1, two jobs 1 and 2 are inserted, where $c_1 = c_2 = 1$ and $w_1 = w_2 = 4$. The daily capacity $k = 2$. On day 3, $k$ is updated to 3, $c_1$ is updated to 0.5, and $w_1$ is updated to 8. On day 5, all jobs are deleted and $I$ is ended.

**Scheduling Module.** The scheduling module contains a repository of schedulers for users to choose. Currently it includes two built-in schedulers, EDD (Earliest Due Day first) and MMEDD (Multicriteria Maintenance version of EDD).

The simple greedy algorithm EDD provides optimal solutions for many non-maintenance scheduling problems minimizing the maximum tardiness [7]. In EDD, jobs are ordered by non-decreasing order of due dates (breaking ties arbitrarily) and scheduled in that order.

EDD minimizes tardiness but cares little about workload. If used in maintenance scheduling, EDD would consume too much workload unnecessarily. Non-maintenance scheduling concerns early completion of jobs, whereas in maintenance scheduling, the concern is long-term in-time maintenance. Jobs will not be completed. Each maintenance of a job simply generates a new due day for the same job. Thus EDD would end up generating too many due days unnecessarily. MMEDD minimizes tardiness while using workload wisely. We omit the algorithmic details of MMEDD as they are

```
<jobs>
        <day>1</day>
        <k>2</k>
        <job>
                <id>1</id>
                <c>1</c>
                <w>4</w>
         </job>
         <job>
                 <id>2</id>
                 <c>1</c>
                 <w>4</w>
         </job>
</jobs>
<jobs>
        <day>3</day>
        <k>3</k>
        <job>
                <id>1</id>
                <c>0.5</c>
                <w>8</w>
         </job>
</jobs>
<jobs>
        <day>5</day>
        <deleteall>yes</deleteall>
</jobs>
```

```
<schedule>
        <id>1</id>
        <day>1</day>
        <jobs>1,2</jobs>
        <day>2</day>
        <jobs>1,2</jobs>
        <day>3</day>
        <jobs>1,2</jobs>
        <day>4</day>
        <jobs>1,2</jobs>
</schedule>
<schedule>
        <id>2</id>
        <day>1</day>
        <jobs></jobs>
        <day>2</day>
        <jobs></jobs>
        <day>3</day>
        <jobs></jobs>
        <day>4</day>
        <jobs>2</jobs>
</schedule>
```

Fig. 2: Example problem and schedules.

**Algorithm 1** *Cube Computation*

```
Input: $I$: scheduling instance; $S$: set of schedules
Output: $F$: fact table of the simulation cube
1: process $I$ and store each update of $c_i$ and $w_i$ for job $i$ into $C_i$ and $W_i$ respectively, where $C_i$ and $W_i$ are arrays of (day, value) pairs;
2: for each $s \in S$
3:   for each day $d \leq$ the end day of $s$
4:     for each job $i$
5:       if (job $i$ is scheduled) then
6:         compare $d$ with $D[i]$ to obtain the tardiness for job $i$ on day $d$ w.r.t. schedule $s$ and insert into $F$;
7:         consult $C_i$ to obtain the current cost of job $i$ and insert into $F$ as the maintenance workload spent on job $i$ on day $d$ w.r.t. schedule $s$;
8:         consult $W_i$ and update $D[i]$;
9:       end if
10:     end for
11:   end for
12: end for
```

beyond the scope of this paper.

A chosen scheduler will generate a set of alternative schedules for input instance $I$. The scheduling step can be executed multiple times using different schedulers, in other words, the generated alternative schedules to be evaluated are not necessarily from the same scheduler.

Instead of using the Scube built-in schedulers, users can also apply external scheduling mechanisms on $I$ and upload the resulting schedules for evaluation, as long as those schedules conform to a simple XML style format as described in the following.

**Alternative Schedules.** An example set of generated alternative schedules is given in Figure 2 (right hand side), demonstrating the required simple format. There are two schedules in the set with id 1 and 2 for the instance $I$ given in the left hand side. Schedule 1 is an “exhausting" schedule, according to which both jobs need to be maintained every day from day 1 to day 4. Schedule 2 is an optimal schedule, according to which no maintenance is needed from day 1 to day 3, and only job 2 needs to be maintained on day 4. Note that in $I$, the due day of job 1 is updated on day 3 and it is not due on day 4 anymore, but on day 8.

**Cube Computation Module.** The cube computation module takes as input the problem instance $I$ and the set of alternative schedules to compute all the tardiness and workload measures. The computed values are then inserted into the fact table of the simulation cube. Algorithm 1 presents the pseudo code for this computation.

In line 6 of Algorithm 1, by comparing the current day $d$ with the current due day of job $i$ that is scheduled (and maintained) on day $d$, we get the tardiness for job $i$ on day $d$ w.r.t. schedule $s$. We insert this tardiness value into the fact table properly.

In line 7, by consulting $C_i$ we can get the current maintenance cost for the scheduled job $i$. This cost is inserted into the fact table as the workload spent on job $i$ on day $d$ w.r.t. schedule $s$.

Note that, the series of due days for job $i$ are not prefixed. They depend on the actual schedule as well as the dynamics of job $i$ itself in the input instance $I$. In line 8, after a maintenance, the next due day for job $i$ is updated. For this update, we need to know the current window size for job $i$ as it may change at any given day.

The algorithm is linear in the number of jobs, the number of days, and the number of schedules.

**Simulation Cube.** The data cube in Scube is called simulation cube because the facts collected and stored in the cube are based on a simulation of some maintenance scenario. In this study, we assume jobs are always maintained as scheduled.

The design of the simulation cube in Scube adopts a star schema, which involves a large central fact table containing the bulk of data with no redundancy, and a set of smaller attendant dimension tables one for each dimension [14].

In Scube, there are three dimension tables, *time*, *jobs*, and *schedules*. The “time" dimension has a concept hierarchy of “day", “week", “month", and “year". For simplicity, they are totally ordered, i.e., a year has 12 months, a month has 4 weeks, and a week has 7 days. The “day" attribute is

Table 1: Comparison of Navigation Modes

| Modes | Functionality | Usability | Implementation |
|---|---|---|---|
| SQL | Very Good | Very Bad | Very Easy |
| CQL | Very Good | Good | Easy |
| Graphic | Very Good | Very Good | Hard |

the primary key in the time dimension table with positive integers as domain. The “time" dimension allows users to evaluate schedules based on their performance on different periods of time of varied granularity.

The “jobs" dimension table has job “id" as primary key. The two other attributes are “cost" and “window", with domains of {high, normal, low} and {long, normal, short} respectively. Jobs are put into different “cost" and “window" categories based on their calculated average cost and average window size. The “jobs" dimension allows users to evaluate schedules based on their performance on different types of jobs. Note that the “jobs" dimension has a partial order concept hierarchy.

The “schedules" dimension table has schedule “id" as primary key. It also has a “type" attribute with domain of {busy, normal, lazy}, allowing comparing and evaluating schedules by groups. *Busy* schedules are those generated (e.g., by MMEDD) for $(k', (c_1, w_1), \ldots, (c_n, w_n))$ with larger $k' \leq k$ values. Such schedules tend to squander workload for the minimization of tardiness. *Lazy* schedules, on the contrary, favor economic use of workload. *Normal* schedules are in between the two.

The fact table has the primary key of every dimension table as one of its attributes. These attributes are foreign keys to the corresponding dimension tables, and they together form the primary key for the fact table. The two measure attributes are “tardiness" and “workload". There is a tardiness value, 0 or a positive number, for each scheduled maintenance and NULL otherwise. Similarly, there is a positive workload value for each maintenance, which is equal to the cost of the job being maintained. We use 0 as the default value for workload, which would not cause any confusion as in the tardiness case.

**Cube Navigation Module.** The cube navigation module allows users to interactively evaluate the alternative schedules by navigating the simulation cube. There are three navigation modes, SQL, CQL, and Graphic.

In the SQL (Structured Query Language) mode, a query window is provided taking any standard SQL query over the fact table. In the CQL (Cube Query Language) mode, only several types of predefined intuitive cube navigation queries are allowed in the query window. In the Graphic mode, the CQL operations can be performed on a visible cube without typing textual queries.

A comparison summary for the three navigation modes is shown in Table 1. In terms of functionality, the SQL mode obviously provides every possible structured way of exploring the fact table. With less querying power in general, the CQL and Graphic modes however provide sufficiently good querying capability, as they are tailored to facilitate cube navigation.

In terms of usability, the SQL mode is very bad because very few regular users are trained to write SQL queries. The CQL mode, however, is good as its syntax is simple, intuitive, and easy to learn. The Graphic mode obviously has the best usability.

In terms of implementation, the SQL mode is very easy only requiring implementation of a simple interface. The CQL mode is also fairly easy, as CQL can be considered as a selective subset of SQL queries plus some syntactic sugar to improve usability. The Graphic mode is hard to implement, but it can be essential if we want to make `Scube` truly accessible to lay users.

To enable any mode of navigation, users need to know the schema of the fact and dimension tables. `Scube` allows users to browse such information easily.

Currently, CQL defines the most basic OLAP operations, *up* (roll-up), *down* (drill-down), and *slice*. A CQL query takes the following format:

$$Operation\ Dimension\ [Value]$$

For example, “up time" means to perform a one level roll-up on the “time" dimension. “down time 2" means to perform a two level drill-down on the “time" dimension. The very top level of any dimension is the absence of the dimension. In addition, “reset" resets the cube to a default state, which corresponds to the apex cuboid in the lattice of cuboids of the simulation cube.

`Scube` allows daisy-chaining multiple commands by separating them with a semicolon, allowing users to jump from one cuboid to another directly, as demonstrated by the following example:

$$reset;\ down\ time\ 2;\ down\ jobs.cost\ 1$$

In the above example, conceptually the cube is first reset to the default state. Then it is drilled-down by 2 levels along the “time" dimension, and then it is drilled-down by 1 level along the “cost" attribute of the “jobs" dimension. Recall that the “jobs" dimension has a partial order concept hierarchy. It is inefficient if all these conceptual steps are actually computed one after the other, as the user is only interested in the final state of the cube. `Scube` analyzes the multiple commands in a daisy-chained command and translate them into a single aggregation group-by SQL query, greatly reducing processing time.

**Best Schedule.** Alternative schedules are typically non-dominated and “goodness" of schedules is generally subject to user preferences. After interactively querying the simulation cube, users will obtain comprehensive knowledge about the overall performance of those alternative schedules, from which they are responsible to decide the “best" schedule.

## 5. Implementation

Scube was implemented in PHP using an OOP architecture, utilizing standard web application development strategies such as the Front Controller Pattern, Model View Controller, and the Template Pattern. This makes the application easily maintainable and extendable. On the server side, Scube made use of several industry standard frameworks, such as the Zend PHP Framework[2] and Smarty PHP Template Engine[3].

On the client side, Scube used Ajax, implemented using the jQuery JavaScript Framework[4], to enable a rich user experience without unnecessary delays caused by repeated browser page refreshes.

In Scube, users follow three steps to complete a scheduling task: present the problem, schedule the problem, and navigate the cube. All initiated actions within the three steps are handled in the background by the server via Ajax requests, which update the browser window with appropriate responses. Scheduling a problem generally takes much longer time than submitting a problem. Thus all submitted problems are queued, and the server checks the queue each minute for new items.

Scube allows users to delete schedules for a given problem. When a schedule is deleted, the data is removed from the simulation cube.

To allow concurrent access to the Scube service, the system generates a random session id for each new application session. Users are allowed to change this id. During subsequent sessions, users can continue to work on a problem by providing the same session id.

## 6. Evaluation

Scube[5] is open for public access and evaluation. In this section, we present our initial usability and scalability studies on Scube. We also introduce a citation information monitoring system[6] as a maintenance scheduling case study.

**User Study.** Usability is a fundamental concern for decision making tools like Scube. Decision makers are business-oriented. They prefer systems that are intuitive, friendly, and easy to learn and to use.

For an initial test on the usability of Scube, we prepared 6 questions regarding learning time, response time and user-friendliness. The questions were distributed to 10 users, most are IT professionals located in Austin (TX, USA) and Wilmington (NC, USA) in a clinical research organization. The questions and the received average scores on a scale of 1 to 10 are given in the following.

[2]http://framework.zend.com/
[3]http://www.smarty.net/
[4]http://jquery.com/
[5]http://dmlab.cs.txstate.edu/scube/
[6]http://dmlab.cs.txstate.edu/citation/

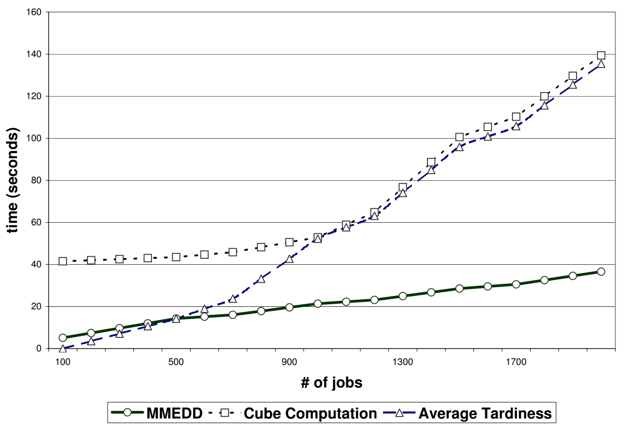

Fig. 3: Scalability.

1) Is the system easy to learn? (9.6)
2) Is the interface user-friendly? (8.8)
3) Does the system respond quickly? (8.8)
4) Are the directions clear? (9.6)
5) Would you consider using such a system if you had to evaluate alternative long schedules? (9.3)
6) How do you rate the system overall? (9.1)

The scores are higher than expected possibly because most participants are IT professionals and better than average at adapting to new technical tools and web services. The system architecture (Q1) received a high score, which is evident for the intuitive and logical design of the Scube architecture as shown in Figure 1. The simple xml style formats for the instance and schedules are also easy to digest.

The interface (Q2) received a lower score possibly because we have not fully implemented the Graphic mode for cube navigation. If users are not familiar with SQL, the SQL navigation mode can be very unfriendly. Although the CQL navigation mode is much more friendly, it may still require some basic understanding on OLAP tools.

The system response performance (Q3) received a lower score possibly because the users did not understand that long-term many-job maintenance tasks could easily take a couple of minutes to schedule. It takes even longer time to calculate the measures for numerous measure points and load them into the simulation cube. Thus, the longer response time may have appeared unexpected.

Overall, the initial deployment of Scube is satisfactory. The participants felt it could well be a convenient tool for comprehensively evaluating long schedules.

**Scalability Study.** In Scube, the most time-consuming modules are scheduling (MMEDD) and cube computation. We experimentally evaluated their scalability performance.

In this series of experiments, we used our data generator to generate instances of different sizes, where daily capacity

$k$ was set to 10 and length of schedules was set to 1000 days. The maintenance cost was set to 1 for every job and the window size values were random numbers in the range of 28 to 100 under the normal distribution.

For each instance size, 5 experiments were performed and the average time was taken. The results are presented in Figure 3, which shows the scalability of MMEDD and cube generation, and thus, Scube.

A similar series of experiments were performed for instances generated under the Poisson distribution, and the results exhibit a similar trend as in Figure 3.

Figure 3 also shows the average tardiness in these experiments, for the purpose of showing an insight that is typical in maintenance scheduling, i.e., the tardiness increases with the increase of number of jobs.

**Citation Information Monitoring.** This research was initially motivated by a citation monitoring service, where we monitor the dynamics of citations for authors (currently 8988 of them) in the database community, and provide online services taking temporal queries for citation. A pilot system, still under continuous development and evaluation, has been deployed and up running for about 10 months. This application is a typical case of dynamic maintenance scheduling problem.

Tardiness directly impacts the functionality of the system. For example, if we crawl Jim Gray once every 30 days, we will not be able to properly answer queries like “what citations did Jim Gray receive in the last 10 days?"

Workload is just as critical. Actually we have received complaints from some web sources for intensive crawling. We definitely need to reduce the total workload.

Thus, in this project we have paid great attention to scheduling. The application is also exposed to a typical dynamic environment, where new authors enter the community at random times and citation rates change dynamically.

The window size $w_i$ for each author $i$ needs to be estimated each day so as to quickly capture dynamic changes of citation rates. The basic idea for this estimation is to calculate a weighted average for the number of citations received by author $i$ in the past certain number (e.g., 300) of days. Each author also needs to be assigned a different maintenance cost, which can be estimated based on the number of publications of the author. More publications lead to bigger maintenance cost.

## 7. Conclusion

In this paper, we demonstrated an interesting application of OLAP in solving multicriteria maintenance scheduling problems. Such problems can properly model many important applications in information monitoring, such as maintenance of inverted indexes for search engines, maintenance of extracted structures for unstructured data management, and maintenance of materialized views in data warehouses, to name a few. We introduced the design and implementation of Scube, an OLAP-based web service that helps users to comprehensively evaluate alternative maintenance schedules and make decisions on the best trade-off schedule to select.

Future work includes continuous development and refinement of Scube, e.g., on the Graphic navigation mode. The built-in repository of schedulers in Scube can be expanded as well. Last but not least, although we provided initial empirical evaluation on Scube, for a more convincing justification, it is important if we can apply Scube to real applications and see the actual long-term benefits it generates.

## References

[1] A. Bar-Noy, R. Bhatia, J. Naor, and B. Schieber. Minimizing service and operation costs of periodic scheduling. *Math. Oper. Res.*, 27(3):518–544, 2002.

[2] A. Bar-Noy and R. E. Ladner. Windows scheduling problems for broadcast systems. *SIAM J. Comput.*, 32(4):1091–1113, 2003.

[3] A. Bar-Noy, R. E. Ladner, and T. Tamir. Windows scheduling as a restricted version of bin packing. *ACM Transactions on Algorithms*, 3(3), 2007.

[4] J. Cho and H. Garcia-Molina. The evolution of the web and implications for an incremental crawler. In *VLDB*, 2000.

[5] J. Cho and H. Garcia-Molina. Synchronizing a database to improve freshness. In *SIGMOD*, 2000.

[6] E. Coffman, Z. Liu, and P. Weber. Optimal robot scheduling for web search engines. *Journal of Scheduling*, (1):15 – 29, 1998.

[7] C. S. David R. Karger and J. Wein. *Scheduling Algorithms*. Algorithms and Theory of Computation Handbook, 1998.

[8] K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley & Sons, 2001.

[9] A. Doan, J. F. Naughton, A. Baid, X. Chai, F. Chen, T. Chen, E. Chu, P. DeRose, B. J. Gao, C. Gokhale, J. Huang, W. Shen, and B.-Q. Vuong. The case for a structrued approach to managing unstructured data. *CIDR*, 2008.

[10] A. Doan, R. Ramakrishnan, F. Chen, P. DeRose, Y. Lee, R. McCann, M. Sayyadian, and W. Shen. Community information management. *IEEE Data Eng. Bull.*, 29(1):64 – 72, 2006.

[11] J. Edwards, K. McCurley, and J. Tomlin. An adaptive model for optimizing performance of an incremental web crawler. In *WWW*, 2001.

[12] M. Ehrgott. *Multicriteria optimization*. Springer, 2005.

[13] G. W. Evans. An overview of techniques for solving multiobjective mathematical programs. 30(11):1268 – 1282, 1984.

[14] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2006.

[15] C. Imhoff and N. Galemmo. *Mastering Data Warehouse Design: Relational and Dimensional Techniques*. John Wiley & Sons, 2003.

[16] R. Kimball and M. Ross. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. John Wiley & Sons, 2002.

[17] L. Lim, M. Wang, and S. Padmananbhan. Dynamic maintenance of web indexes using landmarks. In *WWW*, 2003.

[18] M. Pinedo. *Scheduling - Theory, Algorithms, and Systems*. Prentice Hall, 1995.

[19] R. H. Shoshana Anily, Celia A. Glass. The scheduling of maintenance service. *Discrete Applied Mathematics*, 82(1–3):27–42, 1998.

[20] R. E. Steuer. *Multiple Criteria Optimization: Theory, Computations, and Application*. John Wiley & Sons, 1984.

[21] V. T’kindt and J.-C. Billaut. *Multicriteria Scheduling*. Springer, 2006.