MIS Quarterly

RESEARCH ARTICLE

A TIME-BASED DYNAMIC SYNCHRONIZATION POLICY FOR CONSOLIDATED DATABASE SYSTEMS[1]

Xinxue (Shawn) Qu
Mendoza College of Business, University of Notre Dame,
Notre Dame, IN 46556 50011 U.S.A. {xqu2@nd.edu}

Zhengrui Jiang
School of Business, Nanjing University,
Nanjing 210093 CHINA {zjiang@nju.edu.cn}

Data is becoming an increasingly important asset in today's organizations. Various challenges in the age of big data, such as high volume and high velocity, call for efficient database maintenance policies. This study focuses on deriving an optimal synchronization policy for consolidated database systems. By trading off synchronization cost with data staleness cost, we propose a time-based dynamic synchronization (TDS) policy, which evaluates the system state at predetermined checkpoints and synchronizes a consolidated database system only if given staleness thresholds are crossed. Although several database synchronization and knowledge refreshment policies have been proposed in the prior literature, the TDS policy retains their advantages and overcomes their inflexibility in that, under the TDS policy, system check and synchronization are easy to schedule, disruptions to business operations can be avoided, and synchronization is run only if necessary. Experimental results show that the TDS policy consistently outperforms benchmark policies, leading to substantial cost savings. In particular, the performance gap between the TDS policy and a static periodic policy is greater when data changes arrive less frequently but carry higher unit staleness costs, or when queries arrive more frequently and suffer higher unit staleness costs.

Keywords: Consolidated database (CDB), database maintenance, database synchronization, Markov decision process, dynamic programming

Introduction

It is well documented that data and knowledge can bring substantial value to organizations. For a typical Fortune 1000 company, a 10% increase in data accessibility could result in more than $65 million additional net income (McCafferty 2014). By effectively integrating patient data, the healthcare industry in the United States could save as much as $300 billion a year—equal to reducing costs by $1000 a year for every patient (Bernard 2015). Realizing the considerable value of data, many organizations have invested heavily in data collection, maintenance, and analytics. In a recent survey, 64% of CIO respondents reported that they had expended considerable resources on acquiring and maintaining their data assets (CSC 2015).

However, efficient data maintenance is becoming an increasingly challenging task in the age of big data. There are four well known characteristics of big data: *volume*, *velocity*, *variety*, and *veracity*. High volume and high velocity, in particular, are well-documented in recent reports and projections. For instance, it is predicted that from 2013 to 2020, the digital universe will grow by a factor of 10—from 4.4 trillion gigabytes to 44 trillion (Turner et al. 2014).

[1]Jeffrey Parsons was the accepting senior editor for this paper. Debabrata Dey served as the associate editor.

The appendices for this paper are located in the "Online Supplements" section of *MIS Quarterly*'s website (https://misq.org).

Such voluminous amount of raw data often cannot be directly used to support organization's operational and decision-making needs. Instead, raw data typically needs to go through a series of preprocessing (e.g., cleansing, extraction, transformation, and integration) before it can be incorporated into a clean and usable data repository. Such a data repository can take different forms (e.g., operational databases, data warehouses) and support different applications (e.g., OLTP, OLAP). To achieve a higher query processing speed, separate indexes (Chaudhuri and Narasayya 1997) and/or materialized views are typically created from the completed data tables (Chan et al. 1999). To differentiate them from the unprocessed raw data, we refer to the processed data repository, including indexes and the materialized views, if any, as a *consolidated database* (CDB). The processing of raw data and its integration into the CDB jointly constitute a *database synchronization*. The process of scheduling system checks, synchronization decision-making, and synchronization operations is *database maintenance*.

In reality, raw data may arrive either continuously or in batches. For technical and economic reasons, it is typically inefficient or unfeasible to process raw data immediately upon its arrival. Instead, a synchronization could be run after a certain amount of raw data has been accumulated or a given amount of time has elapsed since the last synchronization (Dey et al. 2006). The optimal frequency of such database synchronization depends on the tradeoff between synchronization cost and data staleness cost.

CDB synchronization is not cost-free because resources, such as IT staff's work hours, computing resources, and electricity, are needed to complete a synchronization process. In addition, if a synchronization is run during business hours, it will compete for computing resources with business processes, thus downgrading system performance and causing disruptions to regular business operations. For instance, it is estimated that a page load slowdown of just one second could cost Amazon.com \$1.6 billion in sales each year (Eaton 2012). As a result, the *business disruption cost* has to be considered when synchronization is run during business hours. Besides technical constraints, such synchronization cost and business disruption cost are the primary reasons why real-time synchronization is infeasible under most settings.

If synchronization and disruption costs were the only consideration, a CDB should be synchronized as infrequently as possible. Unfortunately, data quality would degrade quickly without a new synchronization. Stale data can affect the quality of decisions and lead to financial losses. A global data protection study found that data staleness and system downtime cost enterprises \$1.7 trillion in 2014 (Hopkinton 2014). By surveying 3,300 IT decision makers, this study found that data loss has increased by 400% since 2012. Without efficient maintenance policies, such loss may continue to rise in the future.

The primary goal of the present research is to take into consideration the costs and benefits of maintaining a high level of data recency and derive the optimal synchronization policy for CDBs. We propose a *time-based dynamic synchronization* (or TDS for short) policy under a finite time horizon. The policy assesses the state of the CDB at a predetermined frequency and synchronizes the CDB when its state exceeds precomputed optimal thresholds. The time-based system check schedule avoids unnecessary disruptions to regular business operations, and the dynamic nature of the proposed policy leads to optimal synchronization decisions at each check epoch. Our experimental results show that the TDS policy consistently outperforms benchmark policies proposed in the prior literature. In particular, the performance gap between the TDS policy and a static periodic policy is greater when data changes arrive less frequently but carry higher unit staleness costs, or when queries arrive more frequently and suffer higher unit staleness costs.

The remainder of the paper is organized as follows. We discuss related studies and describe our research problem in the next two sections. We then derive the optimal TDS policy under a finite time horizon. Given the synchronization policy, we also discuss how to select the optimal CDB system check interval. Subsequently, we compare the TDS policy against benchmark policies both theoretically and numerically. We present our concluding remarks in the last section.

Literature Review

The present study is related to several streams of research on database maintenance and knowledge management policies. One research stream focuses on incremental database maintenance and proposes efficient integration algorithms for database refreshing. As an example, Liu et al. (2008) develop a two-phase greedy algorithm to determine the optimal periodic update policy for selected view maintenance.

Other researchers have shifted away from the technical details regarding what and how to update, and instead focus on deciding "when and how frequently to update" (Chaudhuri and Dayal 1997). Segev and Fang (1991) develop a stochastic model to compare the time-based and the query-based refresh policies. Dey et al. (2006) analytically compare three different periodic policies, and show that *query-based* policy is inferior to *time-based* policy in terms of total costs. Although a time-based policy is shown to be less cost-efficient than an *update-based* policy, the difference is insignificant and the latter is difficult to operationalize. Zong et al. (2017) propose a dynamic update policy for constantly changing databases. Their policy requires that the database be checked whenever a query arrives and an update is performed only if necessary. Based on a pure time-based policy and a pure update-based policy, Dey et al. (2015) construct a hybrid patching management policy composed of a time-based threshold and a "severity"-based threshold. However, the proposed hybrid policy cannot be directly applied to the general CDB maintenance problem we consider because it considers only one type of data error and one type of query, while we consider multiple types of data errors and multiple types of queries simultaneously. In addition, the hybrid policy assumes that there is no disruption cost as long as synchronization is planned ahead of time, while we assume that disruption cost can be avoided only if synchronization is performed during off-business hours.

In addition to the maintenance of databases, there exists a stream of literature that focuses on the maintenance of knowledge (e.g., decision rules) learned from databases. Fang and Rachamadugu (2009) propose a dynamic policy for knowledge refreshing, and show that their policy is superior to static periodic refreshing policies. More recently, Fang et al. (2013) apply the Markov decision process to derive an optimal knowledge refreshment policy. They propose a query-based system check policy, which suggests that the system state be examined upon the arrival of an information query, and knowledge refreshment be performed if the state of the database reaches a threshold level.

The present research departs from the prior literature in several important ways. First, while much of the prior research considers the technical aspects of database maintenance, we focus on the economic side of the problem and develop a policy that minimizes the total cost of synchronizations and data staleness. Second, although some prior research (e.g., Dey et al. 2006) has developed static policies to minimize the total cost, the TDS policy we propose is more dynamic in nature. Specifically, unlike static policies that *always* synchronize the database or knowledge at predetermined time points or when the amount of unprocessed data or the number of queries reaches a predetermined level, the TDS policy will decide *whether or not* to synchronize based on the system state at each decision epoch. Third, the TDS policy also differs from the dynamic policies proposed by Fang et al. (2013) and Zong et al. (2017) The most important difference is that their policies are *query-based* dynamic policies, whereas ours is *time-based*. This difference leads to significant changes in the cost structure, model formulation, and subsequently the optimal solutions. For instance, we have to consider the distribution of arriving queries between two decision epochs, and data staleness cost can result even if the CDB is synchronized at every decision epoch, while such sophistication does not exist under the query-based dynamic policies. Moreover, our time-based dynamic policy is easier to schedule in real-world settings than query-based dynamic policies and can avoid disruptions to regular business operations. For instance, instead of requiring that an assessment be made whenever a query arrives, our policy allows assessments and synchronizations to be performed during off-business hours or low-traffic hours to minimize the competition for resources with other business processes. This flexibility can lead to better utilization of resources and significant financial savings. Finally, to better serve real-world applications, we consider multiple types of errors and multiple types of queries simultaneously, while prior studies consider either one type of error with multiple types of queries (Fang et al. 2013) or one type of query with multiple types of errors (Dey et al. 2006). More detailed comparisons of these policies are provided in the policy comparisons section.

Problem Description

The problem we consider is depicted in Figure 1. An organization constantly receives raw data from various sources such as transactional databases operated by different business units, internal updates and reports, third-party databases, and data gleaned from social media. Such raw data typically needs to go through a series of cleansing, extraction, transformation, and loading (ETL) before being merged into a consolidated database (CDB) system. If indexes and materialized views are maintained, they need to be subsequently refreshed. The CDB, instead of the raw data, is then used to respond to various information requests from users.

More often than not, new data or changes to existing data sources arrive continuously throughout the day. If an organization can capture all ongoing changes and synchronize the CDB in real time, the CDB will be able to provide mostly up-to-date information to all of its users. Unfortunately, CDB synchronization is not cost-free. It consumes computing resources and staff hours to run ETL and update all indexes and materialized views. Furthermore, sometimes the CDB needs to be synchronized offline (i.e., for banking systems which require high accuracy and privacy), which reduces the system availability or causes delays to other business transactions. With such *synchronization cost* considered, it is typically not cost-efficient to synchronize a CDB in real time.

Meanwhile, a CDB system is set up for the purpose of enabling users to make better business decision. For example, a high-quality inventory replenish decision depends on precise prediction of market demand and up-to-date inventory levels. An efficient direct marketing operation depends on

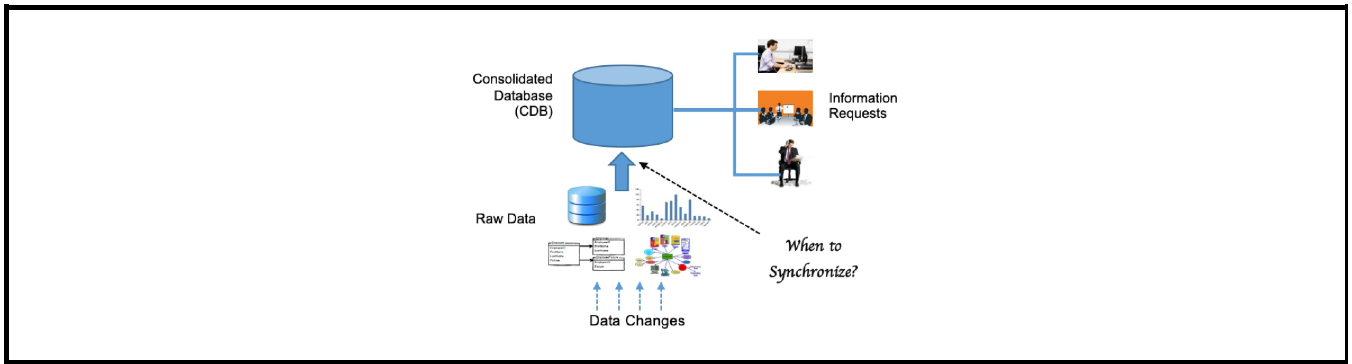

Figure 1. The Big Picture Problem Description

accurate information regarding target customers. The quality of decisions will degrade if the data is not up to date, leading to *data staleness cost* (or simply *staleness cost*). To avoid a high staleness cost, the frequency of CDB synchronizations should not be too low.

In sum, if a CDB is synchronized more frequently, the stored records will be more up-to-date, but the total synchronization costs will be higher. Conversely, a less frequent synchronization schedule reduces the total synchronization cost, but the quality of the data will be lower, leading to less effective decision-marking and hence higher staleness cost. The primary goal of this study is to balance synchronization cost and staleness cost, and derive an optimal CDB maintenance policy that minimizes the total cost.

Data Changes, Queries, and Time-Based Dynamic Synchronization Policy

Since a CDB stores data collected from various sources, when there are *data changes* in the data sources, a portion of the records in the CDB need to be synchronized as well. Data changes include arrivals of new records, modifications to existing records, and deletions of existing records, etc. In a retail chain setting, for example, there are large numbers of new sales transactions, new customers, product returns, canceled orders, customer service requests, and inventory arrivals every day at every store. If not incorporated into the CDB, such data changes can result in staleness cost. Hence we also refer to unprocessed data changes as *data errors* in the rest of the discussion.

During its operation, a CDB receives information requests, submitted in the form of *queries*, from various users in an organization. Examples include product inventory inquiries, sales transaction inquiries, and customer service inquiries. The arrival of data changes and queries as well as the TDS policy we consider are depicted in Figure 2.

As shown in the figure, during a finite time horizon $(0, T)$, various types of data changes and queries continue to arrive. Under the time-based dynamic synchronization (TDS) policy we propose, accumulated data errors are assessed/checked[2] at *decision epochs* of equal distances (e.g., midnight of each day, certain day and time of each week), and the CDB is synchronized if accumulated data errors meet predetermined conditions. If it is synchronized at a particular decision epoch, the CDB becomes error-free. In the absence of a synchronization, data errors will continue to accumulate until the next synchronization operation. If a query arrives right after a synchronization, the returned query results will be error-free, hence there is no staleness cost. Otherwise, the results may contain errors, resulting in data staleness cost. Intuitively, the staleness cost depends on the number of accumulated data errors upon the arrival of the query. The essence of the TDS policy is to decide the condition under which the CDB should be synchronized at each decision epoch.

Assumptions and Definitions

Because different data errors and queries usually lead to different data staleness costs, we consider a general formulation with multiple types of queries and multiple types of data changes/errors. Specifically, we assume that there are G types of data changes/errors and H types of queries. Following prior research (Dey et al. 2006; Fang et al. 2013), we adopt the most common assumption that the arrival of data changes and queries all follow homogeneous Poisson processes, albeit with different rates. This is a reasonable assumption because an established organization usually has

[2]Most data sources for today's CDBs are likely to be internal and structured, hence the numbers and types of changes can be automatically counted by examining just unprocessed records or with minimal interference (e.g., read operations) to existing records of a CDB. For less structured data or data from third-party sources, preprocessing may be needed before new and existing records can be compared.

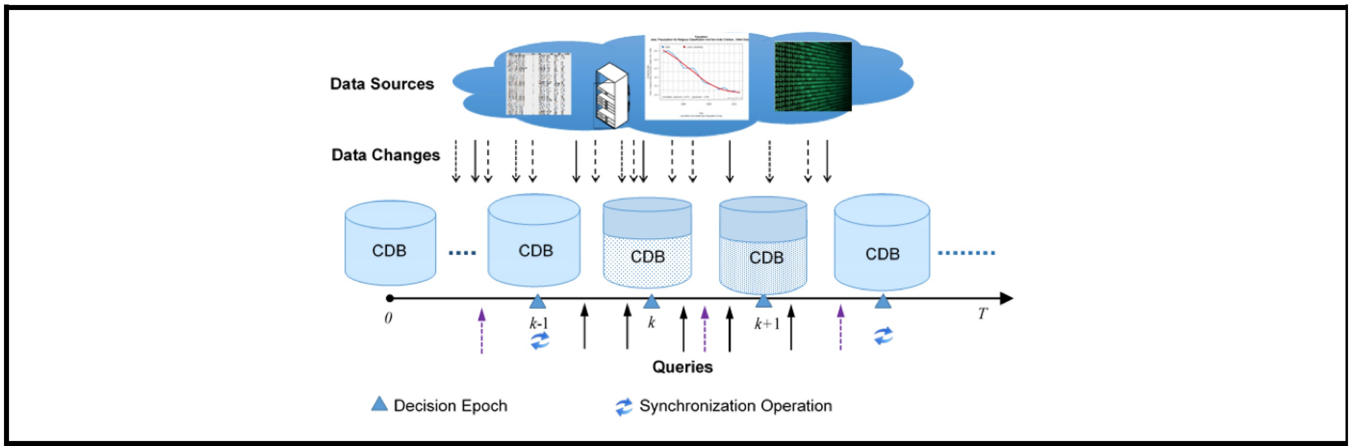

Figure 2. Data Errors, Queries, and TDS Policy

a relatively steady user base and business operations. Formally, we present our first assumption below:

Assumption 1. *The arrival of data changes and queries follows independent homogeneous Poisson processes.*

Based on Assumption 1, we assume that the number of type g data changes arriving *per unit time* follows a Poisson distribution with rate $\lambda_{\Gamma,g}$:

$$\Gamma^{g}_{(t,t+1)} \sim Poisson\left(\lambda_{\Gamma,g}\right), g = 1,2,3,\ldots \ldots, G$$

Similarly, the number of type h query arriving *per unit time* follows a Poisson distribution with rate $\lambda_{Q,h}$:

$$Q^{h}_{(t,t+1)} \sim Poisson\left(\lambda_{Q,h}\right), h = 1,2,3,\ldots \ldots, H$$

In case the traffic patterns of data changes and queries do not precisely follow a Poisson distribution, according to the prior literature (Dey et al. 2006; Fang and Rachamadugu 2009; Fang et al. 2013), the Poisson distribution is still a proper approximation of the data change/query traffic pattern, and possible variations do not affect the performance of the policies.

The Poisson arrival assumption enables us to formulate the CDB synchronization problem as a *Markov decision process* (Bellman 1957). We next define the *system state*.

Definition 1. *The system state of a CDB is a vector consisting of the number of accumulated data errors of all types, that is,* $S = \left(\Gamma^1, \Gamma^2, \ldots, \Gamma^g, \ldots \Gamma^G\right)$, $\Gamma^g \in N$, *and the system state space is* $\mathbf{S} = \mathbf{N}^G$.

When needed, we denote the system state at time t by S_t, which consists of the accumulated data errors of all types at time t: $S = \left(\Gamma^1_t, \Gamma^2_t, \ldots, \Gamma^g_t, \ldots, \Gamma^G_t\right)$. The incremental change of the system state from time t_1 to t_2 is denoted by $S_{(t_1,t_2)} = \left(\Gamma^1_{(t_1,t_2)}, \Gamma^2_{(t_1,t_2)}, \ldots, \Gamma^g_{(t_1,t_2)}, \ldots, \Gamma^G_{(t_1,t_2)}\right)$.

Assumption 2. *Different types of data errors cause staleness costs to different types of queries independently, and the unit staleness cost caused by one occurrence of a specific type of error to a specific type of query is fixed throughout the planning horizon.*

The first part of the assumption implies that, for instance, an error in a sales transaction table is independent of another error in an inventory table in causing losses to relevant queries, and before an error is fixed, the loss it causes to one query is independent of the loss it causes to another query. Analogous assumptions have been adopted in the prior literature (e.g., Dey et al. 2006). Based on this assumption, the staleness costs resulting from different types of data errors are additive. Thus, the staleness cost caused by all data errors in S to a type h query is

$$f_h(S) = f_h\left(\Gamma^1, \Gamma^2, \ldots, \Gamma^G\right) = \Sigma^G_{g=1} f_{g \to h}\left(\Gamma^g\right) \quad (1)$$

The second part of Assumption 2 implies that the staleness cost resulting from a specific type of error is a linear function of the number of accumulated data errors of that type. This is also consistent with the typical treatment in the prior literature (e.g., Dey et al. 2006; Fang et al. 2013). Given this assumption, we denote the unit staleness cost caused by a type g data error to a type h query by $\beta_{g \to h}$. Then we have

$$f_h(S) = \Sigma_{g=1}^{G} \beta_{g \to h} \Gamma^g \tag{2}$$

Let I denote the length of the time interval between two consecutive decision epochs, during which multiple queries could arrive. Suppose S represents the system state at the current decision epoch (time t), then the total staleness cost caused by S to all queries arriving between the current decision epoch and the immediately next one, that is, during time interval (t, t+I), is

$$\begin{aligned} C(S) \equiv C_{(t,t+I)}(S) &= \Sigma_{h=1}^{H} n_h \\ &* f_h\left(\Gamma^1, \Gamma^2, \ldots, \Gamma^g, \ldots, \Gamma^G\right) \\ &= \Sigma_{h=1}^{H}\left[n_h * \Sigma_{g=1}^{G} f_{g \to h}\left(\Gamma^g\right)\right] \end{aligned} \tag{3}$$

where n_h is the number of type h queries arriving during the considered time interval. Since it is a function of the current system state S, we name $C(S)$ the *current state staleness cost.*

From Assumption 1, we know that n_h follows a Poisson distribution with rate $\lambda_{Q,h} * I$. Therefore, the *expected current state staleness cost* is

$$\begin{aligned} E[C(S)] &= \Sigma_{h=1}^{H}\left[\lambda_{Q,h} * I * \Sigma_{g=1}^{G} f_{g \to h}\left(\Gamma^g\right)\right] \\ &= \Sigma_{g=1}^{G} \Gamma^g \left(\Sigma_{h=1}^{H} \lambda_{Q,h} I \beta_{g \to h}\right) \end{aligned} \tag{4}$$

Based on the expected current state staleness cost, we define the order in system state space.

Definition 2. *We define* $S^1 < S^2$ *if and only if* $E[C(S^1)] < E[C(S^2)]$, and $S^1 = S^2$ *if and only if* $E[C(S^1)] = E[C(S^2)]$, *where* $S^1, S^2 \in \mathbf{S}$ *and* $\mathbf{S} = N^G$.

In other words, a lower system state always leads to a lower expected current state staleness cost than a higher system state does, and vice versa.

As previously explained, one of the major costs of CDB maintenance is the synchronization cost. CDB synchronization consumes resources, leading to fixed costs and possible variable costs. The fixed synchronization costs include the setup cost (e.g., labor, cost of preparation, and database restoration), as well as the cost to reconstruct index files and materialized views. The variable cost is assumed to be a linear function of the number of data errors. Since the total variable synchronization cost for the entire planning horizon remains the same across policies, the variable synchronization cost does not affect the relative performance of different maintenance policies, nor does it affect the selection of the optimal system check interval. Therefore, analogous to prior research (e.g., Dey et al. 2006; Fang et al. 2013), we only keep the fixed synchronization cost in our model setup, and denote it by a constant C_U.

As previously explained, system synchronizations during business hours may lead to a slowly responding system or even system downtime for regular business transactions, resulting in business disruption cost. The TDS policy can schedule system checks during off-business or low-traffic hours, thus avoiding or minimizing such disruption. By defining the business disruption cost in relative terms (difference in costs during high-traffic and low-traffic hours), we do not need to consider the business disruption cost when developing the TDS policy.

At each decision epoch in the time horizon, the decision is to synchronize the CDB or not. Hence, the action space contains two possible actions.

Definition 3. *The action space at each decision epoch is* $A = \{0, 1\}$, *where* $a_t = 1$ *means to run synchronization and* $a_t = 0$ *means not to run synchronization. Further,* $a_t < a_t^{'}$ *if and only if* $a_t = 0$ *and* $a_t^{'} = 1$.

Based on the above assumptions and definitions, we next derive the optimal TDS policy for a finite time horizon. (A summary of notations is provided in Appendix A.)

Optimal TDS Policy

In this section, we derive the optimal TDS policy under a finite time horizon.

A Markov Decision Process Model

The CDB maintenance process we consider in this study satisfies the *Markov* property: given the current system state, the optimal action and future system states do not depend on the history (i.e., previous system states, data changes, queries, and synchronization operations). We thus model the CDB maintenance as a *Markov decision process* (MDP).

At each decision epoch, decision-makers should be *forward-looking*: they should discard the history and consider only how the action taken at the current epoch affects the current and future costs and system states. As illustrated in Figure 3, there are two possibilities at decision epoch k:

- If the decision is to synchronize the CDB system, a *synchronization cost* C_U is incurred and the CDB becomes

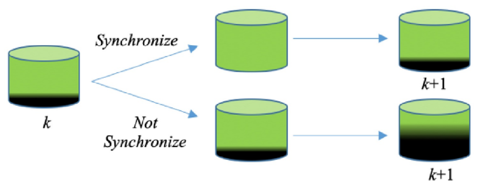

Figure 3. Markov Decision Making Process

error-free, hence data errors accumulated up to decision epoch k (i.e., S_k) will not cause staleness cost to queries arriving in the subsequent time interval (kI, $kI+I$), that is, between two decision epochs k and $k+1$. However, data errors arriving during (kI, $kI+I$) can still cause staleness cost to queries arriving during the same interval if an error arrives before a query. We name this cost *interval staleness cost* and denote it by $C_{(k,\,k+1)}$.

- If the CDB is not synchronized, the data errors accumulated up to epoch k will cause staleness costs to all queries arriving in (kI, $kI+I$). This is the *current state staleness cost* $C(S_k)$ as defined in Eq. (3). In addition, as described in the synchronization scenario, new data errors arriving in (kI, $kI+I$) can cause staleness cost to new queries arriving during the same time interval, hence the same interval staleness cost $C_{(k,\,k+1)}$ is still incurred.

With the synchronization cost, current state staleness cost, and interval staleness cost considered, the *optimal system cost* at decision epoch k takes the following form:

$$V_k(S_k) = \min_{a_k}\left\{a_k C_U + (1-a_k)E\left[C(S_k)\right] + E\left[V_{k+1}(S_{k+1})\right]\right\} + E\left[C_{(k,k+1)}\right] \quad (5)$$

Note that the interval staleness cost is not affected by the action a_k, hence $E[C_{(k,\,k+1)}]$ is a stand-alone term in Eq. (5).

In this optimal system cost function, $E[C(S_k)]$, the expected current state staleness cost captures the expected staleness cost caused by data errors accumulated up to decision epoch k to queries arriving between decision epochs k and $k+1$. At decision epoch k, the system state S_k is known. In the absence of synchronization, based on Eq. (4), the current state staleness cost takes the form of

$$E\left[C(S_k)\right] = \Sigma_{g=1}^{G}\Gamma_k^g\left(\Sigma_{h=1}^{H}\lambda_{Q,h} I\beta_{g\to h}\right)$$

The term $E[V_{k+1}(S_{k+1})]$ represents the expected optimal system cost starting from decision epoch $k+1$ until the end of the planning horizon. Here it is worth noting that the system state at the next decision epoch S_{k+1} depends on the current system state, the action taken at the current decision epoch, and the incremental data changes between the current and the next decision epochs: $S_{k+1} = S_{(k,\,k+1)} + (1-a_k)S_k$. With all possible incremental data changes considered, $E[V_{k+1}(S_{k+1})]$ takes the following form:

$$E\left[V_{k+1}(S_{k+1})\right] = \Sigma_{S_{(k,k+1)}} p\left(S_{(k,k+1)}\right) V_{k+1}\left[S_{(k,k+1)} + (1-a_k)S_k\right] \quad (6)$$

The expected interval staleness cost $E\left[C_{(k,k+1)}\right]$ can be simplified to $\frac{I^2}{2}\Sigma_{h=1}^{H}\lambda_{Q,h}\left(\Sigma_{g=1}^{G}\beta_{g\to h}\lambda_{\Gamma,g}\right)$ (derivation provided in Appendix B).

Based on the optimal system cost function (5), we obtain the following analytical findings:

Lemma 1. *In a finite time horizon, $V_k(S_k)$ is a non-decreasing function of S_k.*

(Proofs of Lemmas and Propositions are provided in Appendix B.)

It is obvious that without any synchronization operation, the system state of a CDB will deteriorate over time. Lemma 1 indicates that a system with a more severe data staleness problem cannot have a lower expected system cost.

Lemma 2. *At any decision epoch k, when $E[C(S_k)] > C_U$, it is always optimal to synchronize the CDB.*

Lemma 2 could help generate a heuristic threshold policy to guide synchronization decisions. However, by considering the current state staleness cost only, such a policy is myopic in nature and not optimal. Nevertheless, Lemma 2 provides an upper-bound for the optimal system state threshold we plan to derive. Further, it helps reduce the otherwise infinite system state space to a finite one.

Truncated System State Space

Although the system state space **S** is infinite and countable, according to Lemma 2, whenever $E[C(S)] > C_U$, $S \in \mathbf{S}$, the optimal action should always be to synchronize the CDB and reset the system to an error-free state. In other words, for any two states S^1 and S^2 satisfying $E[C(S^1)] > C_U$ and $E[C(S^2)] > C_U$, the optimal action is always $a_k(S^1) = a_k(S^2) = 1$, and the corresponding system cost is always $V_k(S^1) = V_k(S^2) = C_U + E[C_{(k,k+1)}] + \Sigma_{S_{(k,k+1)}} p\left(S_{(k,k+1)}\right) V_{k+1}\left(S_{(k,k+1)}\right)$. Therefore, we can define an *upper-bound absorbing state* S_{ub}, which includes all system states with the excepted current staleness cost larger than the synchronization cost (i.e., $E[C(S)] > C_U$). Subsequently, the infinite system state space can be reduced into a finite space. We still use **S** to denote this truncated system state space.

With the system state space simplified, we next examine the transitions between system states. Using a simple example with two types of data errors, Figure 4 illustrates the system state transitions if there is no synchronization. The system state cannot turn better or move sideways; it is impossible for the system to transit from state (1, 1) to state (1, 0), or from state (1, 0) to state (0, 1). It can only continue to "deteriorate" until it reaches the absorbing state S_{ub}.

System state transitions are different with a synchronization. Figure 5(a) shows that all previous states turn to the error-free state right after a synchronization. However, after just one time interval, the system state could transit from the error-free state to all possible states, as shown in Figure 5(b).

The system state transition probability is the joint probability of realizing the different types of data changes in vector $S_{(k,k+1)} = \left(\Gamma^1_{(k,k+1)}, \Gamma^2_{(k,k+1)}, \ldots, \Gamma^G_{(k,k+1)}\right)$. Based on Assumption 1, $\Gamma^g_{(k,k+1)}$ follows an independent Poisson distribution, hence,

$$p\left(S_{(k,k+1)}\right) = \frac{\left(\lambda_{\Gamma,1} I\right)^{\Gamma^1_{(k,k+1)}} e^{-\lambda_{\Gamma,1} I}}{\Gamma^1_{(k,k+1)}!} * \frac{\left(\lambda_{\Gamma,2} I\right)^{\Gamma^2_{(k,k+1)}} e^{-\lambda_{\Gamma,2} I}}{\Gamma^2_{(k,k+1)}!} * \cdots * \frac{\left(\lambda_{\Gamma,G} I\right)^{\Gamma^G_{(k,k+1)}} e^{-\lambda_{\Gamma,G} I}}{\Gamma^G_{(k,k+1)}!} \tag{7}$$

Then, the transition probability from state $S_k = \left(\Gamma^1_k, \Gamma^2_k, \ldots, \Gamma^g_k, \ldots, \Gamma^G_k\right)$ to state $S_{k+1} = \left(\Gamma^1_{k+1}, \Gamma^2_{k+1}, \ldots, \Gamma^g_{k+1}, \ldots, \Gamma^G_{k+1}\right)$ is

$$p\left(S_{k+1} \mid S_k, a_k\right) = \begin{cases} \dfrac{\left(\lambda_{\Gamma,1} I\right)^{\Gamma^1_{k+1}} * \ldots * \left(\lambda_{\Gamma,G} I\right)^{\Gamma^G_{k+1}} * e^{-\Sigma^G_{g=1} \lambda_{\Gamma,g} I}}{\Gamma^1_{k+1}! * \Gamma^2_{k+1}! * \ldots * \Gamma^G_{k+1}!}, & \textit{if } a_k = 1 \\ \dfrac{\left(\lambda_{\Gamma,1} I\right)^{\left(\Gamma^1_{k+1} - \Gamma^1_k\right)} * \ldots * \left(\lambda_{\Gamma,G} I\right)^{\left(\Gamma^G_{k+1} - \Gamma^G_k\right)} * e^{-\Sigma^G_{g=1} \lambda_{\Gamma,g} I}}{\left(\Gamma^1_{k+1} - \Gamma^1_k\right)! * \ldots * \left(\Gamma^G_{k+1} - \Gamma^G_k\right)!}, & \textit{if } a_k = 0 \textit{ and } \forall g\, \Gamma^g_{k+1} \geq \Gamma^g_k, g \in \{1, 2, \ldots, G\} \\ 0 & \textit{if } a_k = 0 \textit{ and } \exists g\, \Gamma^g_{k+1} < \Gamma^g_k, g \in \{1, 2, \ldots, G\} \end{cases} \tag{8}$$

Given the state transition probability, $E[V_{k+1}(S_{k+1})]$ as defined in Eq. (6) could also be expressed as

$$E\left[V_{k+1}\left(S_{k+1}\right)\right] = \Sigma_{S_{k+1} \in \mathbf{S}}\, p\left(S_{k+1} \mid S_k, a_k\right) V_{k+1}\left(S_{k+1}\right) \tag{9}$$

The expressions provided in (4) and (9) allow us to search for the optimal policy, with details provided next.

Optimal Control Limit under a Finite Horizon

In the CDB maintenance context, a policy is a sequence of rules for all decision epochs, $\pi = (d_1, d_2, d_3 \ldots \ldots)$, that determines whether to synchronize the CDB or not based on the system state. A rule for decision epoch k can be expressed as $a_k = d_k(S_k)$.

As the first step, we need to establish the existence of an optimal policy. According to Puterman (2005, p. 90), when the system state space **S** is finite or countable, and the action space A_S is finite for each $S \in \mathbf{S}$, there exists a *deterministic optimal Markovian policy*. In the CDB maintenance problem, these conditions are satisfied, hence a deterministic optimal policy exists.

Because the interval staleness cost is independent of the action taken, we can treat it as a constant. Then, the optimal action at each decision epoch can be obtained by minimizing the remaining cost components in the system cost function:

$$a_k = \arg\min\left\{a_k C_U + \left(1 - a_k\right) E\left[C\left(S_k\right)\right] + E\left[V_{k+1}\left(S_{k+1}\right)\right]\right\} \tag{10}$$

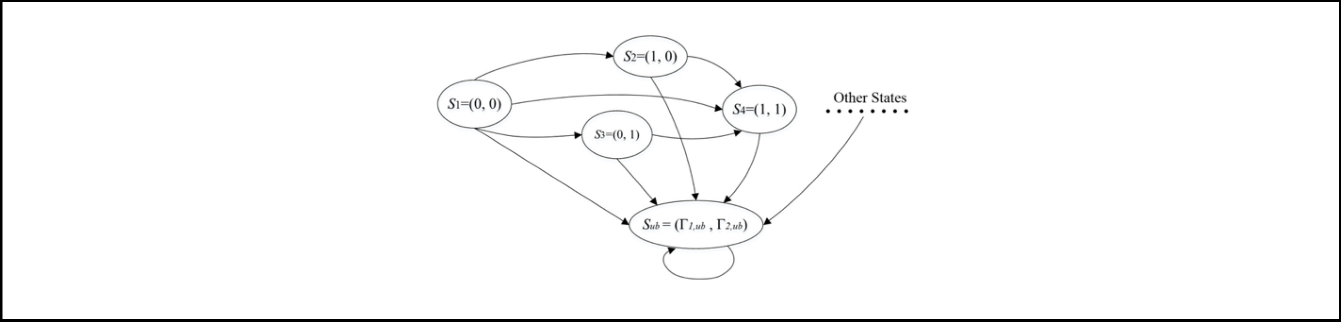

Figure 4. System State Transition Without Synchronization

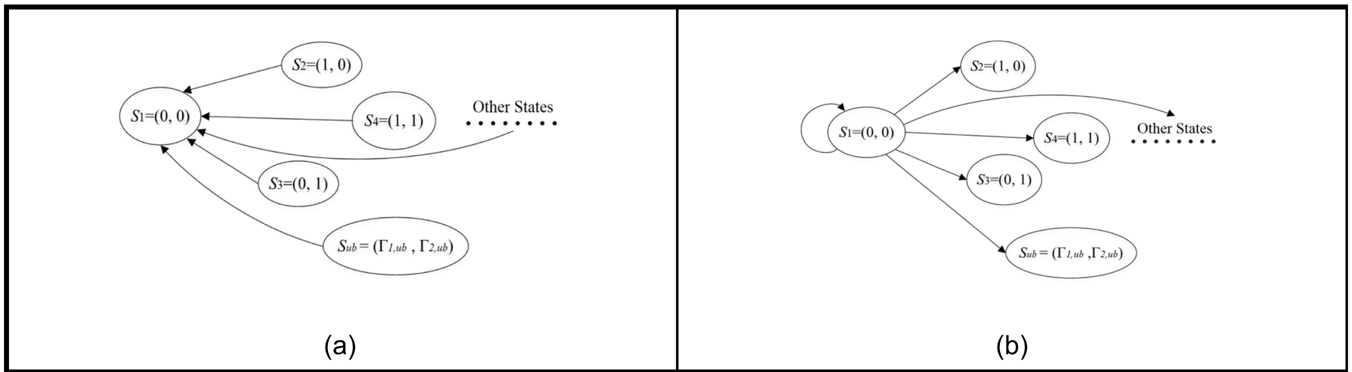

Figure 5. System State Transitions after a Synchronization

Eq. (10) is a Bellman equation, which can be solved via *dynamic programming* and backward induction. The solution is an optimal policy consisting of decision rules for all decision epochs in the form of $a_k = d_k(S_k)$, $k = 1, 2, \ldots, K$.

Since S_k is a vector, it is computationally costly and analytically cumbersome to search for decision rules in terms of the individual elements of the vector. Therefore, we define a linear normed scalar value function to simplify the assessment of the system state of a CDB:

$$W(S_k) = E\left[C(S_k)\right] + \Sigma_{S_{(k,k+1)}} p(S_{k,k+1}) \left[V_{k+1}\left(S_k + S_{(k,k+1)}\right) - V_{k+1}\left(S_{(k,k+1)}\right)\right] \quad (11)$$

Using $W(S_k)$, we can define the decision rule at each decision epoch:

Proposition 1. *The optimal action at decision epoch k is not to synchronize the CDB if* $W(S_k) < C_U$, *and to synchronize the CDB otherwise.*

Based on Proposition 1, the optimal CDB maintenance policy takes the form of a control limit policy:

$$a_k = d_k(S_k) = \begin{cases} 1, & W(S_k) \geq C_U \\ 0, & W(S_k) < C_U \end{cases} \quad (12)$$

To calculate $W(S_k)$ as defined in (11), we first use backward induction to obtain $V_k(S_k)$ for $k = 0, 1, 2 \ldots, K$ and $\forall S_k \in \mathbf{S}$, and record an optimal system cost value matrix $V(k, S_k)$. In this matrix, an element $V(k, S_k)$ represents the optimal system cost at decision epoch k when the system state is S_k. The transition probability matrix needed to calculate $W(S_k)$ can be obtained based on Eq. (8). Then $W(S_k)$ can be computed with vector-wise dot production.

The maintenance policy specified in Proposition 1 essentially uses both the current state staleness cost and expected future system cost to make the synchronization decision. This policy, while valid, is not easy to implement. A simpler and more straightforward policy is thus desired.

From Proposition 1, we conclude that it is always more necessary to synchronize the CDB under a higher system state (more severe data staleness), which leads to the following result:

Lemma 3. *In a finite time horizon, $a_k(S_k)$ is a non-decreasing function of S_k.*

Lemma 3, coupled with the fact that the ordering of system states is based on their expected current state staleness costs, helps us define a threshold-based decision rule:

Proposition 2. *At each decision epoch k, there is a threshold (ζ_k) such that it is optimal not to synchronize the CDB if $E[C(S_k)] < \zeta_k$, and to synchronize the CDB otherwise.*

Based on Proposition 2, the optimal TDS policy consists of the following decision rule:

$$a_k = d_k(S_k) = \begin{cases} 1, & E\left[C(S_k) \geq \zeta_k\right] \\ 0, & E\left[C(S_k) < \zeta_k\right] \end{cases} \tag{13}$$

Unlike decision rule (12), this new decision rule (13) determines whether to synchronize the CDB based on the *expected current state staleness cost* $E[C(S_k)]$, which is much easier to compute (see Eq. (4)) than $W(S_k)$. Although it may appear to be myopic, decision rule (13) is exactly equivalent to rule (12) if the threshold ζ_k is appropriately determined. We next propose an algorithm to search for the threshold ζ_k at each decision epoch k.

Search Algorithm for CDB State Threshold

```
Input:   $T, I, C_U, \lambda_{Q,h}, \lambda_{\Gamma,g}, \beta_{g,h}$, ordered state space S
Output:  $\zeta_k$ and $V_k(S_k)$, $k = 0, 1, 2, \ldots, K$, $K = \lfloor T/I \rfloor$, $\forall S_k \in$ S
Step 1:  when $k = K$
           for $S_k$ in S
             $V_k(S_k) = \min\{C_U + E[C_{(kI,T)}], E[C(S_k)] + E[C_{(kI,T)}]\}$
           if $C_U + E[C_{(kI,T)}] < E[C(S_k) + E[C_{(kI,T)}]$ for the
             first time: $\zeta_k = E[C(S_k)]$
           end if
         end for
Step 2:  for $k = K - 1$ to 0
         for $S_k$ in S
           $V_k(S_k) = \min\{C_U + V_{k+1}(S_{(k,k+1)}), E[C(S_k)] +$
           $V_{k+1}(S_{(k,k+1)}+S_k)\} + E[C_{(k,k+1)}]$
           if $C_U + V_{k+1}(S_{(k,k+1)}) < E[C(S_k)] + V_{k+1}(S_{(k,k+1)} +$
             $S_k)$ for the first time: $\zeta_k = E[C(S_k)]$
           end if
         end for
       end for
```

Based on this algorithm, we can obtain the optimal current state staleness cost threshold ζ_k for all decision epochs, and the optimal system cost $V_k(S_k)$ for all decision epochs with all possible system states. The threshold ζ_k can then be use to decide whether to synchronize the CDB system or not.

At the last decision epoch, we find the following result:

Lemma 4. *The threshold reaches its maximum at the last decision epoch K (where $K1 < T \leq (K + 1I)$ and the maximum threshold satisfies $\zeta_k = C_U$.*

Example: Maintenance of an ERP Database

For illustration, consider an enterprise resource planning (ERP) system used primarily for inventory management and product promotion. Assume there are two types of information requests: (1) queries for inventory levels for better inventory management and (2) queries for previous shopping transactions to identify potential customers. The arrivals of the two types of queries follow Poisson processes. Their average arrival frequencies are once per week and three times per day, respectively. Treating a day as the unit time, their arrival rates are $\lambda_{Q,1} = \frac{1}{7}$ and $\lambda_{Q,2} = 3$.

Meanwhile, there are two types of data changes: (1) consumer profiles updates and (2) new transaction records to be added. Every day, a large number of new shopping transactions are generated, but a smaller number of customers may change their profiles (e.g., addresses, phone numbers). Rather than one change at a time, we assume that data changes arrive at the CDB in batches (1,000 per batch). On average, we assume 1 (thousand) customers add/change their profiles and 10 (thousand) new/modified transactions are recorded every day (i.e., $\lambda_{\Gamma,1} = 1$, $\lambda_{\Gamma,2} = 10$).

If the data changes are not incorporated into the ERP, data errors may lead to different costs for different queries. Profile changes do not impact inventory management (i.e., $\beta_{1\text{-}1} = 0$) but may result in promotion brochures not reaching targeted customers and an average loss of \$2 (i.e., $\beta_{1\text{-}2} = 2$). We assume that missing each thousand sales transactions has a negative impact on the quality of inventory management equivalent to a cost of $\beta_{1\text{-}2} = 8$. Missing sales transactions also leads to overlooked customers' demand for certain products and hence missed opportunity for target promotion, which results in a cost of \$1 (i.e., $\beta_{2\text{-}2} = 1$).

The fixed cost of every synchronization operation, including computing cost, personnel cost, and opportunity cost, is \$1,000 (i.e., $C_U = 1000$). Finally, we consider a three-day interval between every two consecutive decision epochs (system checks), and a one-year time horizon, (i.e., $I = 3$ and $T = 365$. There are 122 decision epochs at day 0, 3, …, 363, respectively.

Given these parameter values, the system state is composed of two types of data changes: $s_t = \left(\Gamma_t^1, \Gamma_t^2\right)$. For instance, (0, 0) indicates that the system is error-free, and (2, 8) means there are 2 (thousand) customer profile changes and 10 (thousand) sales transactions.

We implement the TDS policy consisting of decision rules in the form of Eq. (13). Following the algorithm summarized above, we obtain the expected system cost $V_k(S_k)$ and the optimal threshold ζ_k for current state staleness cost at every decision epoch. In this example, the thresholds are $\zeta = \{717, 717, 717, \ldots, 741.86, 534.43, 1000\}$, a result consistent with Lemma 4. The expected total system cost at the very beginning $V_0(S_1 = (0,0))$, which is the total expected system cost in the one-year time period, equals \$111,591.6.

Optimal Check Interval

The proposed TDS policy requires that the state of a CDB system be checked at predetermined and equal-distance checkpoints (e.g., midnight of each day, certain time of a week). Because the *check interval* (i.e., the time duration between two consecutive checkpoints) is used in the derivation of several cost factors (e.g., the current state cost $E[C(S_k)]$ and the interval staleness cost $E[C_{(k,k+1)}]$), we expect it to have an impact on the optimal policy. In this section, we discuss how the optimal check interval can be determined.

Determining the Range of Feasible Check Intervals

In theory, the status of a CDB can be checked very frequently (e.g., every hour, or even a few times per hour). In practice, however, unless a synchronization operation can be run at each scheduled checkpoint and completed before the next checkpoint, the assumed system check frequency would be infeasible. For instance, the ETL processes are very complex (El-Sappagh et al. 2011), and can take more than 12 hours to complete one ETL cycle for large organizations (Woodall et al. 2016). Then the check interval should not be shorter than 12 hours. Further, if a CDB synchronization can only be run at midnight of a day, then having a check interval less than one day would make little practical sense. Moreover, even if it is technically possible to check/synchronize the system at every midnight, other business/technological considerations may stipulate that a system should not be checked more frequently than once every two days. The feasible check interval should also avoid any possible disruptions to regular business operations. Therefore, the *minimal check interval* should be determined based on operational and technological constraints faced by organizations.

Unlike the minimal check interval, the *maximal check interval* can be theoretically decided:

Lemma 5. *Under the TDS policy, there exists an upper-bound* $I^{ub} = \sqrt{C_U \left[\Sigma_{h=1}^{H} \lambda_{Q,h} \left(\Sigma_{g=1}^{G} \beta_{g\to h} \lambda_{\Gamma,g}\right)\right]^{-1}}$ *for the optimal length of check interval.*

Essentially, we should restrict the length of the check interval so that in expectation, the staleness cost caused by the data errors accumulated in a given check interval to queries arriving in the next interval should not be larger than C_U. Otherwise, in expectation it would be optimal to synchronize the CDB at every decision epoch, then there is no need to develop a dynamic synchronization policy.

In case the minimal check interval is greater than or equal to I^{ub}, based on Lemma 5, there is no need to implement TDS; therefore, we only consider the scenario where the minimal check interval is strictly less than I^{ub}. Then, the optimal check interval for the TDS policy should lie between the minimal and maximal check intervals.

Selecting the Optimal Check Interval

The shorter the check interval, the more checkpoints (decision epochs) will result. Intuitively, more system checks can improve CDB maintenance and reduce the sum of data staleness and synchronization costs.

Lemma 6. *When the check interval shrinks from I to I/m, where m is a positive integer, the optimal total system cost either remains the same or decreases.*

Lemma 6 can be explained as follows: When the check interval is shortened from *I* to *I*/*m*, the number of decision epochs will increase from *K* to *mK*+ in the given time horizon. Further, the *K* decision epochs are a subset of the *mK*+ epochs. As a result, the optimal policy with a shorter check interval *I*/*m* cannot perform worse than the optimal policy with a longer check interval *I*.

In the feasible range [*minimal check interval*, *maximal check interval* (I^{ub})], it is generally not necessary to select from a large number of feasible system check intervals. To avoid business disruptions and constrained by the duration of synchronization operations, for most practical applications, we believe that feasible check intervals should consist of multiples of the *minimal check interval*. Therefore, based on Lemma 6, the optimal check interval for the TDS policy is typically the minimal check interval.

Policy Comparisons

The prior literature has proposed a number of database synchronization and knowledge refreshment policies. Among them, four employ a cost benefit analysis similar to that of the present study. Three polices (*periodic (time-based) policy, query-based policy*, and *update-based policy*) have been analyzed by Dey et al. (2006). The periodic policy schedules synchronizations according to a predetermined frequency, at predetermined points in time. The query-based policy requires a synchronization operation whenever the number of queries reaches a predetermined threshold. The update-based policy recommends that synchronization be performed when the number of accumulated data errors reaches a predetermined level. The fourth policy, a *query-based dynamic policy* (e.g., Fang et al. 2013), requires that the system state be assessed upon the arrival of each new query, and knowledge should be refreshed only if the accumulated data changes crossed a specific threshold. In this section, we theoretically and numerically compare the TDS policy against the four existing policies, and discuss the performance of our policy under different conditions.

Theoretical Comparison with Existing Policies

The four existing policies each have their own advantages and disadvantages. The periodic policy is the easiest to operationalize because it is very easy to schedule. Its downside is that synchronization will always be performed according to schedule regardless of the number of accumulated data changes or queries, which can lead to waste of computing and labor resources if the number of data errors is low at a scheduled time. Conversely, when there is a sudden surge in changes of data sources right after a synchronization operation, the periodic policy cannot recommend a timely synchronization operation, resulting in expensive staleness cost until the next synchronization. The query-based policy and update-based policy overcome this limitation to different extents, but they require that the state of the system or the arriving queries be continuously monitored, and the actual time of synchronizations cannot be predicted, possibly resulting in disruptions to regular business operations if synchronization occurs during business hours. The query-based dynamic policy proposed by Fang et al. (2013) has the advantage of higher flexibility, and can return error-free knowledge to users if knowledge refreshment is performed, but the policy is also difficult to schedule because the timing of synchronization is uncertain due to stochastic arrivals of queries, and runs the risk of causing disruptions to business operations.

The TDS policy proposed in this study overcomes the inflexibility and retains the advantages of the existing polices. Specifically, our policy checks the system state according to a predetermined check interval, making the checks and synchronizations easy to schedule and avoiding disruptions to business operations. In the meantime, it offers sufficient flexibility so that the database is synchronized only when the expected system cost of not synchronizing is higher than the expected cost in case synchronization is performed.

Although the prior studies (Dey et al. 2006; Fang et al. 2013) and the present study all have derived the optimal policies by trading off staleness cost against synchronization cost, the business disruption cost has not been widely considered in the prior literature. By scheduling system checks and synchronizations during off-business hours, the TDS policy and periodic policy can avoid expensive disruption to business operations (Dey et al. 2015). For instance, if a synchronization operation occurs during business hours for a business, it can reduce the number of customers that can be served, leading to longer system response time or, in certain cases, render the service unavailable for an extended period of time. It is worth noting that, for some businesses, even a small percentage drop in service capacity could lead to significant loss. For instance, Google has calculated that, by slowing its search results by just four tenths of a second, 8 million searches, and a significant amount of ad revenue, could be lost per day (Eaton 2012). Therefore, disruption cost could be far more expensive than synchronization cost. By scheduling synchronizations during off-business hours, TDS and periodic policies can avoid such disruption costs.

Primarily because of disruption to business operations, the average system cost under non-time-based policies (i.e., the update-based and query-based policies) is expected to be significantly higher than that under time-based policies (i.e., the TDS and periodic policies). To better understand the impact of business disruption cost, we run a numerical experiment to examine how the TDS policy, the periodic policy, and the update-based policy perform as the business disruption cost (C_d) increases.[3] From Figure 6, we can see that the update-based policy results in a slightly lower system cost than the TDS policy only when there is no disruption cost. When the business disruption cost increases from 10% of the synchronization cost, the proposed TDS policy begins to dominate. Once the disruption cost exceeds 50% of the synchronization cost, the update-based policy is even inferior to the periodic policy. In most practical settings, disruption cost is likely more expensive than synchronization cost, hence time-based policies (i.e., TDS and periodic) are superior to non-time-based policies.

[3]The experiment is run and the default parameter values are set based on the same illustrating example that will be reported later in this section. Query-based policy is not included because, according to Dey et al. (2006), it is strictly inferior to the update-based policy.

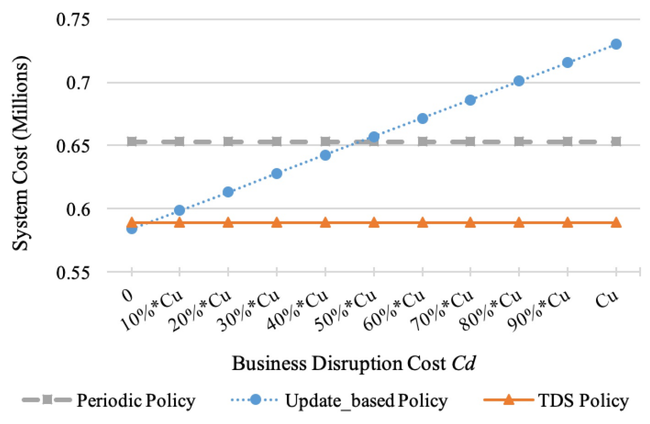

Figure 6. Performance Comparison with Varying Business Disruption Cost

In sum, taking into consideration the ease of scheduling and the avoidance of business disruptions, we believe that the periodic policy and the TDS policy are significantly more feasible and more cost-efficient than the other three policies. These could also be the underlying reasons why database system users prefer periodic synchronization schedules (Kimball and Caserta 2011). For these reasons, among the four existing policies, the periodic policy is the only one that is truly comparable to the TDS policy. For this reason, in the performance evaluation that follows, we only compare these two policies.

Performance Comparison with the Periodic Policy

The periodic policy developed by Dey et al. (2006) recommends an optimal fixed synchronization interval in the form of $I^*_{period} = \sqrt{\frac{2D}{\lambda_u \lambda_q}}$, where D equals the ratio of synchronization cost to unit staleness cost (i.e., $\frac{C_u}{\beta}$). Recall that the prior study considers only one type of data error or one type of query. For comparison with the dynamic policy proposed in this study, we need to extend Dey et al.'s policy to include multiple types of queries and errors. The optimal fixed synchronization period for multiple types of queries and errors is

$$I^*_{period} = \sqrt{\frac{2C_U}{\sum_{h=1}^{H} \lambda_{Q,h} \left(\sum_{g=1}^{G} \beta_{g \to h} \lambda_{\Gamma,g} \right)}} \qquad (14)$$

which can be computed once the model parameter values are set. Compared with the upper-bound in Lemma 5, that is,

$$I^{ub} = \sqrt{C_U \left[\sum_{h=1}^{H} \lambda_{Q,h} \left(\sum_{g=1}^{G} \beta_{g \to h} \lambda_{\Gamma,g} \right) \right]^{-1}}$$

, the optimal synchronization interval under the periodic policy is always larger than the optimal system check interval under the TDS policy. Although Eq. (14) represents an extension of Dey et al.'s periodic policy, for convenience we still call it the periodic policy.

We consider two types of errors and two types of queries in our simulated experiment. The parameter values are set at $\lambda_{Q,1} = 1$, $\lambda_{Q,2} = 3$, $\lambda_{\Gamma,1} = 0.5$, $\lambda_{\Gamma,2} = 1.5$, $\beta_{1\to 1} = 120$, $\beta_{1\to 2} = 90$, $\beta_{2\to 1} = 60$, $\beta_{2\to 2} = 30$, and $C_U = 1000$. Given these values, we have $I^*_{period} = 2$. Based on the previous discussed principles regarding the selection of optimal check interval, we select one day as the optimal check interval (i.e., $I^* = 1$). The time horizon is varied from one year to five years. In our experiment, we first compute the optimal thresholds under the TDS policy based on the algorithm shown earlier. We then simulate the stochastic arrivals of errors and queries, apply the TDS policy and periodic policy, and record the system costs for both policies. The comparisons of total system costs under the two polices are summarized in Table 1 and Figure 7.

From Table 1 and Figure 7, we can see that the TDS policy consistently outperforms the periodic policy. The performance gap widens as the time horizon increases (Figure 7a); however, the percentage of improvement decreases slightly, but still stays above 10% (Figure 7b). We thus conclude that the TDS policy can lead to significant reductions in total system cost.

In addition to the static periodic policy, we have also compared the TDS policy against the hybrid policy developed by Dey et al. (2015) under the latter's model assumptions. The

Table 1. Comparison between Periodic Policy and TDS Policy

Time Horizon	1 Year	2 Years	3 Years	4 Years	5 Years
Periodic Policy ($)	335,800	671,600	1,007,400	1,343,200	1,679,000
TDS Policy ($)	301,407	603,640	905,873	1,208,106	1,510,339
Improvement (%)	10.24%	10.12%	10.08%	10.06%	10.05%

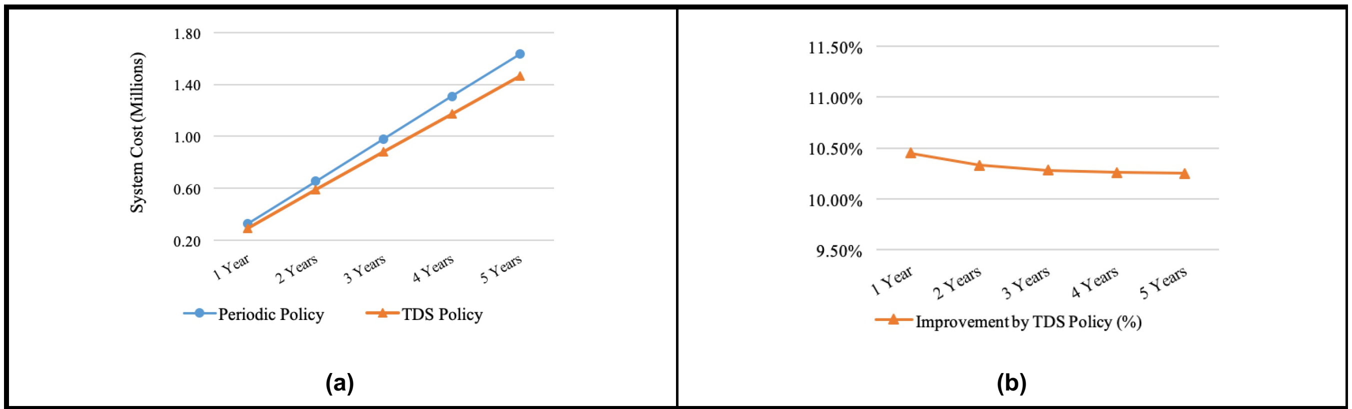

Figure 7. Periodic Policy Versus TDS Policy

results show that the TDS policy consistently outperforms the hybrid policy. Details about this comparison are provided in Appendix D.

Sensitivity Analysis

In practice, the arrival rates of errors and queries and the unit staleness costs could vary greatly from application to application. In this subsection, we examine how the TDS policy performs under a variety of scenarios.

Based on Assumption 2, each type of data error leads to staleness costs independently. Therefore, in our experiment, we first vary one type of error's arrival rate, then change its unit staleness cost, while keeping other factors constant. The key parameter values are set as follows: $C_U = 2000$, $\lambda_{Q,1} = 1$, $\lambda_{Q,2} = 2$, $\lambda_{\Gamma,2} = 1.5$, $\beta_{2\text{-}1} = 60$, $\beta_{2\text{-}2} = 30$, and $I = I^*_{period}/4$. The numerical results are presented in Figure 8. All else being equal, the advantage of the TDS policy measured by the percentage of reduction in total system cost grows as the unit cost of error increases. However, the performance improvement becomes less significant as the data errors arrive more frequently. In sum, the figure shows that the TDS policy leads to significantly larger cost savings when data errors arrive in lower frequency but cause larger unit staleness cost. Conversely, when the data sources change frequently, but each error causes small unit staleness costs, although our policy still dominates, the relative advantage shrinks considerably.

We next analyze how the arrival rates of queries and related unit costs affect the performance of the TDS policy in relation to the periodic policy. The adopted parameter values in our experiment are $C_U = 2000$, $\lambda_{Q,2} = 3$, $\lambda_{\Gamma,1} = 0.5$, $\lambda_{\Gamma,2} = 1.5$, $\beta_{1\text{-}2} = 90$, $\beta_{2\text{-}2} = 30$, and $I = I^*_{period}/4$. The performance comparison is shown in Figure 9. From the figure, we observe that, as the arrival rate of the query process increases, the performance improvement achieved by the TDS policy becomes more significant. Similarly, when the unit staleness cost incurred to queries increases, the advantage of the TDS policy also improves. In sum, the result shows that the TDS policy leads to significantly larger cost savings when queries arrive in higher frequency and carry higher unit staleness cost.

Another interesting pattern shown in Figure 9 is that the contour figure is symmetric in the query frequency and unit cost dimensions. This shows that increasing the unit cost or the frequency of a query has very similar impact on the performance improvement of the TDS policy.

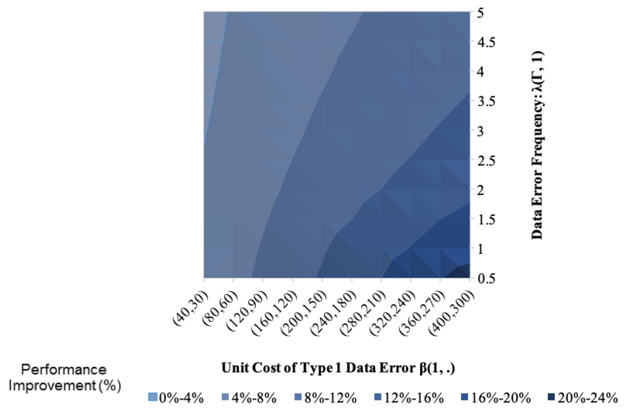

Figure 8. Impact of Errors' Arrival Rates and Unit Costs on Performance of TDS Policy

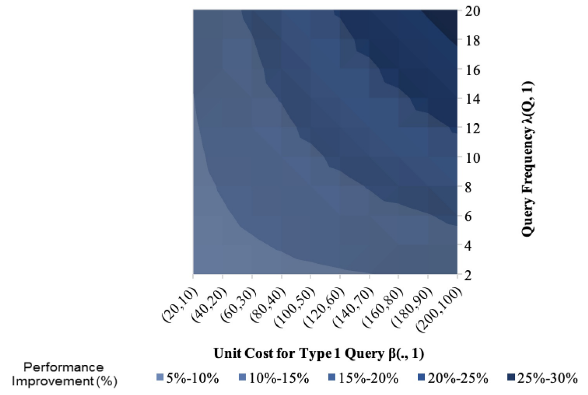

Figure 9. Impact of Queries' Arrival Rates and Unit Costs on Performance of TDS Policy

Concluding Remarks

Given the increasing reliance on big data and analytics in today's organizations, efficient maintenance of consolidated database (CDB) systems is as important an operational issue as ever. To support high-quality decision-making, database systems need to be kept up to date; otherwise, outdated data can lead to data staleness costs. On the other hand, CDB synchronizations are not cost-free because they are typically computation intensive operations, and synchronizations during business hours may disrupt regular business operations. Therefore, to derive an optimal CDB maintenance policy, decision-makers need to consider the tradeoff between data staleness cost and synchronization cost.

Deciding an optimal CDB maintenance policy, however, is not a trivial problem because of the unpredictability of arrivals of new data, changes to existing data, information requests, and varying staleness cost caused by different errors to different information requests. The prior literature has proposed a number of policies, but they are either difficult to schedule in practical settings, or lack the flexibility to skip an expensive synchronization operation if the number of data changes is low at a prescheduled synchronization time. In this study, we develop a time-based dynamic synchronization (TDS) policy, which checks the state of the CDB system according to a predetermined time interval, and synchronize the CDB only if the system state crossed an optimal threshold. As a result, the TDS policy retains the advantages and overcomes the inflexibility of the existing policies. The TDS policy outperforms the benchmark periodic policy and significantly lowers the total system cost. In particular, the TDS policy performs even better in relation to the periodic policy when data errors arrive less frequently but carry a higher unit staleness cost, or queries arrive more frequently and suffer a higher loss in the presence of data errors.

In addition to contributing to the database maintenance literature, the TDS policy we propose has significant practical implications. First, in comparison with update-based and query-based benchmark policies, TDS's ease of scheduling allows checks and synchronizations be performed during off-business or low-traffic hours, thus reducing the competition for computing resources with other system processes, and avoiding expensive disruptions to regular business operations, which could be the largest cost item to consider in formulating a database maintenance policy. Second, compared with the periodic policy, synchronization is performed only if the benefit of synchronizing is higher than the cost of synchronizing, thus avoiding unnecessary waste. Third, even with relatively conservative assumptions, our experimental results still show that the TDS policy can result in cost savings ranging from over 4% to more than 20% compared with the static periodic policy. For organizations with a large CDB system to maintain, this could translate to substantial financial savings.

We would like to point out that once the optimal synchronization thresholds are derived, which could be done prior to implementation and offline, the TDS policy is fairly easy and straightforward to implement. It is in fact no more complex than the static periodic policy. Therefore, we see little reason why the TDS policy cannot replace existing policies proposed in the extant literature.

The current research could be extended along several directions. For example, the arrivals of data changes and queries are assumed to follow homogeneous Poisson processes in this study. A future study could relax this assumption and develop an optimal maintenance policy under non-homogeneous Poisson arrivals of data changes and queries. In another possible direction, one could utilize the previous records to dynamically update the statistical distributions of the arrivals of data changes and queries at each decision epoch, and then decide the optimal threshold for synchronization. Finally, most existing database maintenance and knowledge refreshment policies are developed under a finite planning horizon. Extending our TDS policy to an infinite horizon case could be a promising direction for future research.

Acknowledgments

The authors would like to thank the senior editor, Jeffrey Parsons, the associate editor, and the two anonymous reviewers for their constructive comments and suggestions. Zhengrui Jiang served as the corresponding author of this paper.

References

Bellman, R. E. 1957. "A Markovian Decision Process," No. P-1066, Santa Monica, CA: RAND Corporation.

Bernard, M. 2015. "Big Data: 20 Mind-Boggling Facts Everyone Must Read," *Forbes* (https://www.forbes.com/sites/bernardmarr/2015/09/30/big-data-20-mind-boggling-facts-everyone-must-read/#4de45f9517b1; accessed November 22, 2016).

Chan, G. K., Li, Q., and Feng, L. 1999. "Design and Selection of Materialized Views in a Data Warehousing Environment: A Case Study," in *Proceedings of the 2nd ACM International Workshop on Data Warehousing and OLAP*, New York: ACM, pp. 42-47.

Chaudhuri, S., and Dayal, U. 1997. "An Overview of Data Warehousing and OLAP Technology," *ACM SIGMOD Record* (26:1), pp. 65-74.

Chaudhuri, S., and Narasayya, V. R. 1997. "An Efficient, Cost-Driven Index Selection Tool for Microsoft SQL Server," *VLDB* (97), pp. 146-155.

CSC. 2015. "The CSC Global CIO Survey," (http://www.csc.com/cio_survey_2014_2015; November 22, 2016).

Dey, D., Lahiri, A., and Zhang, G. 2015. "Optimal Policies for Security Patch Management," *INFORMS Journal on Computing* (27:3), pp. 462-477.

Dey, D., Zhang, Z., and De, P. 2006. "Optimal Synchronization Policies for Data Warehouses," *INFORMS Journal on Computing* (18:2), pp. 229-242.

Eaton, K. 2012. "How One Second Could Cost Amazon $1.6 Billion in Sales," *Fast Company (*https://www.fastcompany.com/1825005/how-one-second-could-cost-amazon-16-billion-sales; accessed May 17, 2018).

El-Sappagh, S. H. A., Hendawi, A. M. A., and El Bastawissy, A. H. 2011. "A Proposed Model for Data Warehouse ETL Processes," *Journal of King Saud University-Computer and Information Sciences* (23:2), pp. 91-104.

Fang, X., and Rachamadugu, R. 2009. "Policies for Knowledge Refreshing in Databases," *Omega* (37:1), pp. 16-28.

Fang, X., Sheng, O. R. L., and Goes, P. 2013. "When Is the Right Time to Refresh Knowledge Discovered from Data?," *Operations Research* (61:1), pp. 32-44.

Hopkinton, M. 2014. "Over $1.7 Trillion Lost Per Year from Data Loss and Downtime According to Global IT Study," *EMC News* (http://www.emc.com/about/news/press/2014/20141202-01.htm; accessed November 22, 2016).

Kimball, R., and Caserta, J. 2011. *The Data Warehouse? ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data*, New York: John Wiley & Sons.

Liu, Y. C., Hsu, P. Y., Sheen, G. J., Ku, S., and Chang, K. W. 2008. "Simultaneous Determination of View Selection and Update Policy with Stochastic Query and Response Time Constraints," *Information Science* (178:18), pp. 3491-3509.

McCafferty, D. 2014. "Surprising Statistics About Big Data," *Baseline* (http://www.baselinemag.com/analytics-big-data/slideshows/surprising-statistics-about-big-data.html; accessed November 21, 2016).

Puterman, M. L. 2005. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, New York: John Wiley & Sons.

Segev, A., and Fang, W. 1991. "Optimal Update Policies for Distributed Materialized Views," *Management Science* (37:7), pp. 851-870.

Turner, V., Gantz, J. F., Reinsel, D., and Minton, S. 2014. "The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things," *Analyze the Future*, International Data Corporation, Framingham, MA.

Woodall, P., Borek, A., Oberhofer, M., and Gao, J. 2016. Data Quality Problems in ETL: The State of the Practice in Large Organisations," in *Proceedings of the International Conference on Information Quality*, Ciudad Real, Spain, pp. 54-64.

Zong, W., Wu, F., and Jiang, Z. 2017. "A Markov-Based Update Policy for Constantly Changing Database Systems," *IEEE Transactions on Engineering Management* (64:3), pp. 287-300.

About the Authors

Xinxue (Shawn) Qu is an assistant professor at the Mendoza College of Business, University of Notre Dame. He received his Ph.D. in Business and Technology from Iowa State University. His primary research interests include business intelligence/analytics, database management, multi-generation technology diffusion, and reinforcement learning. He has published his work in *MIS Quarterly* and *Decision Sciences* and presented his research at multiple conferences including INFORMS Conference on Information Systems and Technology (CIST) and the Workshop on Information Technology and Systems (WITS). He won the Excellence in Doctoral Student Research award at the Ivy College of Business, Iowa State University. He is a member of the Association of Information Systems and INFORMS.

Zhengrui Jiang is a professor at the School of Business, Nanjing University. He previously was the Thome Professor in Business and professor of information systems at the Ivy College of Business, Iowa State University. He received his Ph.D. in Management Science from the University of Texas at Dallas. His primary research interests include business intelligence/analytics, data quality, decision-making under uncertainty, diffusion of technological innovations, and economics of information technology. He research appears in leading academic journals including *Information Systems Research*, *Management Science*, *MIS Quarterly*, *IEEE Transactions on Knowledge and Data Engineering*, *INFORMS Journal on Computing*, and *Journal of Management Information Systems*. He serves, or has served, as an associate editor for *Information Systems Research* and *MIS Quarterly*, as a senior editor for *Production and Operations Management*, and as a program co-chair for the 2014 Midwest Association of Information Systems Conference, the 2015 Big XII+ MIS Research Symposium, and the 2018 Workshop on Information Technologies and Systems, He received the *MIS Quarterly* Outstanding Associate Editor award in 2016.

A Time-Based Dynamic Synchronization Policy for Consolidated Database Systems

Xinxue (Shawn) Qu
Mendoza College of Business, University of Notre Dame,
Notre Dame, IN 46556 U.S.A. {xqu2@nd.edu}

Zhengrui Jiang
School of Business, Nanjing University,
Nanjing 210093 CHINA {zjiang@nju.edu.cn}

Appendix A

Notation Table

T	Finite maintenance horizon time length
$g = 1, 2, \ldots, G$	G types of data errors
$\mathrm{h} = 1, 2, \ldots, H$	H types of information queries
$\lambda_{\Gamma,\mathrm{g}}$	Poisson arrival rate for type g data error
$\lambda_{\mathrm{Q,h}}$	Poisson arrival rate for type h information query
$\Gamma^{g}_{(t,t+1)}$	Amount of type g data errors from epoch t to epoch t+1
$\mathrm{Q}^{h}_{(t,t+1)}$	Amount of type h query from epoch t to epoch t+1
Γ^{g}	Accumulated amount of type g data error
S	CDB system state
$\beta_{g\to h}$	Unit staleness cost of type g data error to type h query
a_k	Action take at the kth decision epoch
I	Check interval length
C_U	Fixed synchronization cost
C_d	Business disruption cost
$\mathbf{S}$	System state space
$S_{(t_1,t_2)}$	The change of system state from time t_1 till time t_2
$d_t(S)$	Decision rule at epoch t given system state S
π	Maintenance policy
$T[S, S']$	Transition probability from state S to state S′
d^{*}	Optimal decision rule

ζ_k	Threshold for System Staleness Cost at decision epoch k
T_p	State transition probability matrix

Appendix B

Important Derivations

Derivation of Expected Interval Staleness Cost $\mathbf{E}[\mathbf{C}_{(\mathbf{k},\mathbf{k+1})}]$

Denoting the number of type *h* queries arriving during the time interval by n_h, which follows a Poisson distribution, and their arrival times by t_{n_h}, t_{n_h-1}, …, t_1, respectively, we have

$$E[C_{(k,k+1)}] = \sum_{h=1}^{H}\sum_{n_h=1}^{\infty}\left\{\int_{kI}^{kI+I} d(t_{n_h})\int_{kI}^{t_{n_h}} d(t_{n_h-1}) \ldots \int_{kI}^{t_2} d(t_1) * (\lambda_{Q,h})^{n_h} * e^{-\lambda_{Q,h}*I} * \left[\sum_{i=1}^{n_h}\sum_{g=1}^{G} E_{\Delta\Gamma^g,(kI,t_i)}[f_h(\Delta\Gamma^g)]\right]\right\} \tag{B-0}$$

With a check interval of length *I*, the time window between the *k*th decision epoch and the *k*+1th epoch is $(kI, kI+I)$, *k*= 0, 1, 2, ….

In time interval $(kI, kI+I)$, we assume that there are n_h information queries for type *h* query (*h*=1, 2…, *H*). Because the arrivals of queries follow independent Poisson distribution, the time between two consecutive arrivals $t_i - t_{i-1}$ follow i.i.d exponential distribution. Thus, based on Assumption 1, the probability that type *h* query occurs n_h times at time $t_1, t_2, \ldots, t_{n_h} \in (kI, kI+I)$ is:

$$P(n_h, t_1, t_2, \ldots, t_{n_h}) = (\lambda_{Q,h})e^{-\lambda_{Q,h}*t_1} * (\lambda_{Q,h})e^{-\lambda_{Q,h}*(t_2-t_1)} * \ldots * (\lambda_{Q,h})e^{-\lambda_{Q,h}*(t_{n_h}-t_{n_h-1})} * e^{-\lambda_{Q,h}*(I-t_{n_h})} = (\lambda_{Q,h})^{n_h} * e^{-\lambda_{Q,h}*I} \tag{B-1}$$

During the interval, the set of new data errors is denoted by $S_{(k,k+1)}$, $S_{(k,k+1)} = (\Gamma^1_{(k,k+1)}, \Gamma^2_{(k,k+1)}, \ldots, \Gamma^g_{(k,k+1)}, \ldots, \Gamma^G_{(k,k+1)})$.

The expected interval staleness cost is

$$\begin{aligned} E[C_{(k,k+1)}] = \sum_{h=1}^{H}\sum_{n_h=1}^{\infty}\{&\int_{kI}^{kI+I} d(t_{n_h})\int_{kI}^{t_{n_h}} d(t_{n_h-1}) \ldots \int_{kI}^{t_2} d(t_1) * (\lambda_{Q,h})^{n_h} * e^{-\lambda_{Q,h}*I} \\ &* \left[\sum_{i=1}^{n_h}\sum_{\Delta\Gamma^1=0}^{\infty}\sum_{\Delta\Gamma^2=0}^{\infty} \cdots \sum_{\Delta\Gamma^G=0}^{\infty} f_h(\Delta\Gamma^1, \Delta\Gamma^2, \ldots, \Delta\Gamma^G)\frac{e^{-\lambda_{\Gamma,1}(t_i-kI)}[\lambda_{\Gamma,1}(t_i-kI)]^{\Delta\Gamma^1}}{(\Gamma^1!)} * \ldots * \frac{e^{-\lambda_{\Gamma,G}t_i}[\lambda_{\Gamma,G}(t_i-kI)]^{\Delta\Gamma^G}}{(\Delta\Gamma^G!)}\right]\} \end{aligned} \tag{B-2}$$

where t_i is the time when the *i*th type *h* query arrives.

Based on Assumption 1, the *G* types of data errors occur independently, hence the joint probability distribution of $(\Delta\Gamma^1, \Delta\Gamma^2, \ldots, \Delta\Gamma^G)$ is the product of the occurring probabilities of all *G* types of data error.

Regarding the cost function $f_h(\Delta\Gamma^1, \Delta\Gamma^2, \ldots, \Delta\Gamma^G)$, according to Assumption 2, each data error leads to business losses independently. Then,

$$f_h(\Delta\Gamma^1, \Delta\Gamma^2, \ldots, \Delta\Gamma^G) = f_h(\Delta\Gamma^1) + f_h(\Delta\Gamma^2) + \cdots + f_h(\Delta\Gamma^G) \tag{B-3}$$

Let $t_i' = t_i - kI$, then $\sum_{\Delta\Gamma^1=0}^{\infty}\sum_{\Delta\Gamma^2=0}^{\infty} \cdots \sum_{\Delta\Gamma^G=0}^{\infty} f_h(\Delta\Gamma^1, \Delta\Gamma^2, \ldots, \Delta\Gamma^G)\frac{e^{-\lambda_{\Gamma,1}t_i'}(\lambda_{\Gamma,1}t_i')^{\Delta\Gamma^1}}{(\Delta\Gamma^1!)} * \ldots * \frac{e^{-\lambda_{\Gamma,G}t_i'}(\lambda_{\Gamma,G}t_i')^{\Delta\Gamma^G}}{(\Delta\Gamma^G!)}$ can be decomposed to (under the independent assumption)

$$\sum_{\Delta\Gamma^1=0}^{\infty} f_h(\Delta\Gamma^1)\frac{e^{-\lambda_{\Gamma,1}t_i'}(\lambda_{\Gamma,1}t_i')^{\Delta\Gamma^1}}{(\Delta\Gamma^1!)} + \cdots + \sum_{\Delta\Gamma^G=0}^{\infty} f_h(\Delta\Gamma^G)\frac{e^{-\lambda_{\Gamma,G}t_i'}(\lambda_{\Gamma,G}t_i')^{\Delta\Gamma^G}}{(\Delta\Gamma^G!)} \tag{B-4}$$

Each element in Eq. (B-4) is an expected value derived based on the distribution of $\Delta\Gamma^g$.

The above result can be rewritten as

$$E_{\Delta\Gamma^1}[f_h(\Delta\Gamma^1)] + E_{\Delta\Gamma^2}[f_h(\Delta\Gamma^2)] + \cdots + E_{\Delta\Gamma^G}[f_h(\Delta\Gamma^G)] = \sum_{g=1}^{G} E_{\Delta\Gamma^g,(kI,t_i)}[f_h(\Delta\Gamma^g)] \tag{B-5}$$

This expected cost function depends on the length of the considered time interval. Specifically, during a time interval $(kI,\ t_i)$,

$$\Delta\Gamma^g_{(kI,t_i)} \sim \text{Poisson}\big(\lambda_{\Gamma,g} * (t_i - kI)\big) \tag{B-6}$$

Based on this result,

$$E\big[C_{(k,k+1)}\big] = \sum_{h=1}^{H}\sum_{n_h=1}^{\infty}\{\int_{kI}^{kI+I} d\big(t_{n_h}\big)\int_{kI}^{t_{n_h}} d\big(t_{n_h-1}\big) \dots \int_{kI}^{t_2} d(t_1) * \big(\lambda_{Q,h}\big)^{n_h} * e^{-\lambda_{Q,h}*I} * \left[\sum_{i=1}^{n_h}\sum_{g=1}^{G} E_{\Delta\Gamma^g,(t,t_i)}[f_h(\Delta\Gamma^g)]\right]\} \tag{B-7}$$

Previous studies have assumed a linear form for the cost function (Dey 2006), so a special case here is to assume a linear form for $f_h(\Delta\Gamma^g) = \beta_{h,g} * \Delta\Gamma^g$. Then

$$E[f_h(\Delta\Gamma^g)] = \beta_{h,g} * \lambda_{\Gamma,g} * (t_i - kI) \tag{B-8}$$

The linear expression for $E\big[C_{(k,k+1)}\big]$ thus becomes

$$E\big[C_{(k,k+1)}\big] = \sum_{h=1}^{H}\sum_{n_h=1}^{\infty}\{\int_{kI}^{kI+I} d\big(t_{n_h}\big)\int_{kI}^{t_{n_h}} d\big(t_{n_h-1}\big) \dots \int_{kI}^{t_2} d(t_1) * \big(\lambda_{Q,h}\big)^{n_h} * e^{-\lambda_{Q,h}*I} * [(\sum_{g=1}^{G}\beta_{h,g} * \lambda_{\Gamma,g})\sum_{i=1}^{n_h}(t_i - kI)]\} \tag{B-9}$$

Since $(\sum_{g=1}^{G}\beta_{h,g} * \lambda_{\Gamma,g})$ is not affected by the outside summation on h and n_h, we have

$$\begin{aligned} E\big[C_{(k,k+1)}\big] = &\sum_{h=1}^{H}(\sum_{g=1}^{G}\beta_{h,g} * \lambda_{\Gamma,g})\sum_{n_h=1}^{\infty}\big(\lambda_{Q,h}\big)^{n_h} * e^{-\lambda_{Q,h}*I} \\ &* \left\{\int_{kI}^{kI+I} d\big(t_{n_h}\big)\int_{kI}^{t_{n_h}} d\big(t_{n_h-1}\big) \dots \int_{kI}^{t_2} d(t_1) * \sum_{i=1}^{n_h}(t_i - kI)\right\} \end{aligned} \tag{B-10}$$

Given that *kI* is a constant denoting the starting time of the interval, we can denote $t_i' = t_i - kI$.

$$\begin{aligned} E\big[C_{(k,k+1)}\big] &= \sum_{h=1}^{H}(\sum_{g=1}^{G}\beta_{h,g} * \lambda_{\Gamma,g})\sum_{n_h=1}^{\infty}\big(\lambda_{Q,h}\big)^{n_h} * e^{-\lambda_{Q,h}*I} * \left\{\int_{0}^{I} d\big(t'_{n_h} + kI\big)\int_{0}^{t'_{n_h}} d\big(t'_{n_h-1} + kI\big) \dots \int_{0}^{t'_2} d(t'_1 + kI) * \sum_{i=1}^{n_h}(t_i')\right\} \\ &= \sum_{h=1}^{H}(\sum_{g=1}^{G}\beta_{h,g} * \lambda_{\Gamma,g})\sum_{n_h=1}^{\infty}\big(\lambda_{Q,h}\big)^{n_h} * e^{-\lambda_{Q,h}*I} * \left\{\int_{0}^{I} dt'_{n_h}\int_{0}^{t'_{n_h}} dt'_{n_h-1} \dots \int_{0}^{t'_3} dt'_2 \int_{0}^{t'_2} dt'_1 * \sum_{i=1}^{n_h}(t_i')\right\} \end{aligned} \tag{B-11}$$

According to Dey et al. (2006)

$$\int_0^I dt_n \int_0^{t_n} dt_{n-1} \dots \int_0^{t_3} dt_2 \int_0^{t_2} dt_1 * \left(\sum_{i=1}^{n} t_i^m\right) = \frac{I^{m+n}}{(m+1)(n-1)!} \tag{B-12}$$

Using the induction above,

$$\int_0^I dt'_{n_h} \int_0^{t'_{n_h}} dt'_{n_h-1} \dots \int_0^{t'_3} dt'_2 \int_0^{t'_2} dt'_1 * \sum_{i=1}^{n_h} (t'_i) = \frac{I^{n_h+1}}{2(n_h-1)!} \tag{B-13}$$

$$E\left[C_{(t,t+I)}\right] = \sum_{h=1}^{H}\left(\sum_{g=1}^{G} \beta_{h,g} * \lambda_{\Gamma,g}\right) \sum_{n_h=1}^{\infty} \left(\lambda_{\mathrm{Q},h}\right)^{n_h} * e^{-\lambda_{\mathrm{Q},h}*I} * \frac{I^{n_h+1}}{2(n_h-1)!} \tag{B-14}$$

Let $j = n_h - 1$,

$$E\left[C_{(k,k+1)}\right] = \sum_{h=1}^{H}\left(\sum_{g=1}^{G} \beta_{h,g} * \lambda_{\Gamma,g}\right) \sum_{n_h=1}^{\infty} \left(\lambda_{\mathrm{Q},h}\right)^{j+1} * e^{-\lambda_{\mathrm{Q},h}*I} * \frac{I^{j+2}}{2j!} = \frac{1}{2}\sum_{h=1}^{H}\left(\sum_{g=1}^{G} \beta_{g} * \lambda_{\Gamma,g}\right) * \lambda_{\mathrm{Q},h} I^2 \sum_{j=0}^{\infty} \frac{e^{-\lambda_{\mathrm{Q},h}*I} * \left(\lambda_{\mathrm{Q},h} * I\right)^j}{j!} \tag{B-15}$$

Here, $\sum_{j=0}^{\infty} \frac{e^{-\lambda_{\mathrm{Q},h}*I} * \left(\lambda_{\mathrm{Q},h}*I\right)^j}{j!}$ is a summation of the probability of a Poisson $(\lambda_{\mathrm{Q},h} * I)$ distribution, which equals to 1. Therefore,

$$E\left[C_{(k,k+1)}\right] = \frac{1}{2}\sum_{h=1}^{H}\left(\sum_{g=1}^{G} \beta_{h,g} * \lambda_{\Gamma,g}\right) \lambda_{\mathrm{Q},h} I^2 = \frac{I^2}{2}\sum_{h=1}^{H}\left(\sum_{g=1}^{G} \beta_{h,g} * \lambda_{\Gamma,g}\right) \lambda_{\mathrm{Q},h} \tag{B-16}$$

The above expectation of $C_{(k,k+1)}$ is a special case when the cost function is linear with the data errors.

Appendix C

Proofs of Selected Lemmas and Propositions

Proof of Lemma 1

To prove that $V_k(S_k)$ is a non-decreasing function of S_k, we only need to show that given $S_k^1 > S_k^2,\ V_k(S_k^1) \geq V_k(S_k^2)$ will hold.

According to Eq. (5): $V_k(S_k) = \min_{a_k}\{a_k C_U + (1-a_k)E[C(S_k)] + E[V_{k+1}(S_{k+1})]\} + E[C_{(k,k+1)}]$, the total expected interval staleness costs remain the same regardless of adopted maintenance policies. Hence, we consider the following two scenarios:

Scenario 1: If $a_k(S_k^1) = a_k(S_k^2) = 1$,

$$V_k(S_k^1) = V_k(S_k^2) = C_U + E[C_{(k,k+1)}] + \sum_{S_{(k,k+1)} \in \mathbb{S}} p(S_{(k,k+1)}) V_{k+1}[S_{(k,k+1)}] \tag{C-1}$$

Scenario 2: Otherwise, assume π_1 is the optimal policy when the system is in state S_k^1 and π_2 is the optimal policy for a system state S_k^2. Suppose k^U is the first decision epoch with a synchronization for the CDB in state S_k^1 under the optimal policy π_1. If the system in state S_k^2 also follows policy π_1, which also runs the first synchronization operation at k^U. Then from time $k^U + 1$, the expected system costs will be the same in the two scenarios, i.e. $E[V_{k^U+1}^{\pi_1}(S_{k^U+1})] = \sum_{S_{(k,k+1)} \in \mathbb{S}} p(S_{(k,k+1)}) V_{k^U+1}[S_{(k,k+1)}]$. Therefore,

$$V_k(S_k^1) = V_k^{\pi_1}(S_k^1) = E[C(S_k^1)] + E[C(S_{k+1}^1)] + \cdots + E[C(S_{k^U-1}^1)] + C_U + \sum_{\vec{\Delta} \in \mathbb{S}} p(S_{(k,k+1)}) V_{k^U+1}[S_{(k,k+1)}] \tag{C-2}$$

$$V_k^{\pi_1}(S_k^2) = E[C(S_k^2)] + E[C(S_{k+1}^2)] + \cdots + E[C(S_{k^U-1}^2)] + C_U + \sum_{S_{(k,k+1)} \in \mathbb{S}} p(S_{(k,k+1)}) V_{k^U+1}[S_{(k,k+1)}] \tag{C-3}$$

Since $S_k^1 > S_k^2$, we will have $E[C(S_k^1)] > E[C(S_k^2)]$. Without any synchronization from decision epoch k to epoch $k^U - 1$, the data errors and queries in the system follow the same traffic pattern. This means $S_t^1 > S_t^2$ will always hold for $t \in \{k, k+1, \ldots, k^U - 1\}$. Therefore, we have

$$E[C(S_k^1)] + E[C(S_{k+1}^1)] + \cdots + E[C(S_{k^U-1}^1)] > E[C(S_k^2)] + E[C(S_{k+1}^2)] + \cdots + E[C(S_{k^U-1}^2)] \tag{C-4}$$

Hence, $V_k(S_k^1) = V_k^{\pi_1}(S_k^1) > V_k^{\pi_1}(S_k^2)$.

Further, because π_2 is the optimal policy to apply given system in state S_k^2, the following inequality must hold:

$$V_k^{\pi_1}(S_k^2) \geq V_k^{\pi_2}(S_k^2) = V_k(S_k^2) \tag{C-5}$$

Therefore, we conclude: $V_k(S_k^1) > V_k(S_k^2)$.

From the discussions under both Scenarios 1 and 2, we conclude that given $S_k^1 > S_k^2$, $V_k(S_k^1) \geq V_k(S_k^2)$ will always hold. Therefore, $V_k(S_k)$ is a non-decreasing function with S_k.

Proof of Lemma 2

The optimal decision should always lead to a smaller expected system costs at any decision epoch. When the CDB is synchronized, based on the optimal system costs function in Eq. (5), we denote the incurred future system cost as

$$J_k(S_k, a_k = 1) = C_U + E\left[C_{(k,k+1)}\right] + \sum_{S_{(k,k+1)}} p\left(S_{(k,k+1)}\right) V_{k+1}\left(S_{(k,k+1)}\right) \tag{C-6}$$

In the absence of a synchronization, we denote the incurred future system cost as

$$J_k(S_k, a_k = 0) = E[C(S_k)] + E\left[C_{(k,k+1)}\right] + \sum_{S_{(k,k+1)}} p\left(S_{(k,k+1)}\right) V_{k+1}\left(S_{(k,k+1)} + S_k\right) \tag{C-7}$$

According to Lemma 1, $V_{k+1}(S_{(k,k+1)} + S_k) \geq V_{k+1}(S_{(k,k+1)})$. Therefore, if $E[C(S_k)] > C_U$, the incurred future system cost without an synchronization$J_k(S_k, a_k = 0)$ will be larger than the incurred future system cost after synchronization: $J_k(S_k, a_k = 1)$. Therefore, it is always optimal to synchronize the CDB when $E[C(S_k)] > C_U$.

Proof of Proposition 1

In Eq. (11), $W(S_k) = E[C(S_k)] + \sum_{S_{(k,k+1)}} p\left(S_{(k,k+1)}\right)\left[V_{k+1}\left(S_k + S_{(k,k+1)}\right) - V_{k+1}\left(S_{(k,k+1)}\right)\right]$. According to Definition 1, $E[C(S_k)]$ is increasing with S_k. Also by Lemma 1, $V_{k+1}\left(S_k + S_{(k,k+1)}\right)$ is a non-decreasing function with S_k. Hence we can conclude that $W(S_k)$ is monotonically increasing with S_k.

According to the Bellman Eq. (10): $a_k = \arg min\{a_k C_U + (1 - a_k)E[C(S_k)] + E[V_{k+1}(S_{k+1})]\}$, and from Eq. (6) $E[V_{k+1}(S_{k+1})] = \sum_{S_{(k,k+1)}} p\left(S_{(k,k+1)}\right) V_{k+1}[S_{(k,k+1)} + (1 - a_k)S_k]$, we have the optimal action a_k at epoch *k* as

$$a_k = \begin{cases} 1, & E[C(S_k)] + \sum_{S_{(k,k+1)}} p\left(S_{(k,k+1)}\right) V_{k+1}[S_{(k,k+1)} + S_k] \geq C_U + \sum_{S_{(k,k+1)}} p\left(S_{(k,k+1)}\right) V_{k+1}[S_{(k,k+1)}] \\ 0, & E[C(S_k)] + \sum_{S_{(k,k+1)}} p\left(S_{(k,k+1)}\right) V_{k+1}[S_{(k,k+1)} + S_k] < C_U + \sum_{S_{(k,k+1)}} p\left(S_{(k,k+1)}\right) V_{k+1}[S_{(k,k+1)}] \end{cases} \tag{C-8}$$

Moving $E_{S_{(k,k+1)}}\left[V_{k+1}\left(S_{(k,k+1)}\right)\right]$ to the left-hand-side of the inequality, the condition becomes the comparison between $W(S_k) = E[C(S_k)] + \sum_{S_{(k,k+1)}} p\left(S_{(k,k+1)}\right)\left[V_{k+1}\left(S_k + S_{(k,k+1)}\right) - V_{k+1}\left(S_{(k,k+1)}\right)\right]$ and C_U. The optimal action at decision epoch *k* is not to synchronize the CDB if $W(S_k) < C_U$, and to synchronize the CDB otherwise.

Proof of Lemma 3

According to the control limit policy in Eq. (12) and Proposition 1, as S_k increases, $W(S_k)$ increases from smaller than C_U to larger than C_U. When the value of $W(S_k)$ crosses C_U, the optimal action will change from 0 to 1. Therefore, $a_k(S_k)$ is a non-decreasing function of S_k.

Proof of Lemma 4

Based on the control limit policy (12), at the last decision epoch *K*, the optimal action is to synchronize whenever $W(S_K) \geq C_U$. In addition, $W(S_K) = E[C(S_K)] + \sum_{\vec{\Delta}} p\left(S_{(k,k+1)}\right)\left[V_{K+1}\left(S_K + S_{(k,k+1)}\right) - V_{K+1}\left(S_{(k,k+1)}\right)\right]$.

At the last epoch, $V_{K+1}(\cdot) = 0$, so we have $W(S_K) = E[C(S_K)]$. The optimal action to take should be determined by

$$a_k = \begin{cases} 1, & W(S_K) = E[C(S_K)] \geq C_U \\ 0, & W(S_K) = E[C(S_K)] < C_U \end{cases} \tag{C-9}$$

Following the decision rule in Eq. (13), we have the threshold $\zeta_K = C_U$. Because the threshold ζ_k is the boundary for $E[C(S_k)]$, which makes $W(S_k)$ larger than C_U when $E[C(S_k)]$ crosses ζ_k. Therefore, we have the value of threshold at *k*th epoch:

$$\zeta_k = C_U - \sum_{S_{(k,k+1)}} p\left(S_{(k,k+1)}\right)\left[V_{k+1}\left(S_K + S_{(k,k+1)}\right) - V_{k+1}\left(S_{(k,k+1)}\right)\right] \tag{C-10}$$

Based on Lemma 1, $\sum_{S_{(k,k+1)}} p\big(S_{(k,k+1)}\big)\big[V_{k+1}\big(S_K + S_{(k,k+1)}\big) - V_{k+1}\big(S_{(k,k+1)}\big)\big]$ should always be positive, therefore,

$$\zeta_k < C_U = \zeta_K, \text{ for } k=1, 2, \ldots, K\text{-}1 \tag{C-11}$$

Proof of Lemma 5

The expected accumulated data errors between two check points is: $(\lambda_{\Gamma,1}I, \lambda_{\Gamma,2}I, \ldots, \lambda_{\Gamma,G}I)$. In expectation, the expected number of new coming information queries in the next interval is $(\lambda_{\mathrm{Q},1}I, \lambda_{\mathrm{Q},2}I, \ldots, \lambda_{\mathrm{Q},H}I)$.

The expected loss to all those expected arriving queries by the accumulated data errors in one check interval equals

$$E[C(S_I)] = I^2\left[\sum_{h=1}^{H} \lambda_{\mathrm{Q},h}\left(\sum_{g=1}^{G} \beta_{g\to h}\lambda_{\Gamma,g}\right)\right] \tag{C-12}$$

If the data errors accumulated in one check interval will lead to an expected data staleness cost in the next coming interval that is larger than the synchronization cost C_U, in expectation it would be optimal to synchronize the CDB system at every single decision epoch, then there will be no need to discuss the policy in detail.

Therefore, for the upper bound, we have the constraint $E[C(S_I)] \le C_U$, i.e.,

$$I^2\left[\sum_{h=1}^{H} \lambda_{\mathrm{Q},h}\left(\sum_{g=1}^{G} \beta_{g\to h}\lambda_{\Gamma,g}\right)\right] \le C_U \tag{C-13}$$

which is equivalent to

$$I \le \sqrt{C_U\left[\sum_{h=1}^{H} \lambda_{\mathrm{Q},h}\left(\sum_{g=1}^{G} \beta_{g\to h}\lambda_{\Gamma,g}\right)\right]^{-1}} \tag{C-14}$$

Hence, we have the interval upper bound $I^{ub} = \sqrt{C_U\left[\sum_{h=1}^{H} \lambda_{\mathrm{Q},h}\left(\sum_{g=1}^{G} \beta_{g\to h}\lambda_{\Gamma,g}\right)\right]^{-1}}$.

Appendix D

Comparison between TDS Policy and Hybrid Policy

To demonstrate the superiority of the time-based dynamic synchronization (TDS) policy, we also compare the TDS policy against a hybrid policy proposed by Dey et al. (2015) with the following characteristics: It involves two thresholds: a fixed time threshold, I^*, and a staleness cost threshold, S^*, both optimized to be used in combination. Under this hybrid policy, a system refresh is initiated if either the time threshold or the staleness cost threshold is triggered, whichever happens earlier.

Since the TDS policy and the hybrid policy (Dey et al. 2015) use different model parameters and adopt different assumptions, to make the comparison possible, we have to (1) obtain the key model parameters for Dey et al.'s hybrid policy by mapping its model parameters with ours, and (2) downgrade our policy to fit Dey et al.'s model assumptions. More specifically, the key adaptions made to facilitate a meaningful comparison are summarized in Table D1 below.

Table D1. Key Adaptations Made to the TDS Policy

Original Assumptions	Adapted Settings
Multiple types of data errors	Single type of data error
Multiple types of queries	Single type of query
Unit staleness cost for each error-query pair	One generalized random unit staleness cost
Different costs for synchronizations running in business hours and off-business hours	No disruption cost when planned ahead of time or scheduled during off-business hours

Experimental Results

In our simulated experiments, we implemented the hybrid policy developed by Dey et al. (2015) and the downgraded TDS policy based on the revised assumptions described above. The key parameter values are set as followings: patching (data error) arriving rate $\lambda = 3$, mean severity level $\bar{C}_e = 2$, simulated time window $T = 5{,}000$ units of time, and system check interval for TDS policy $I = 1$ (daily). Regarding the normalized patching setup cost (c_s) and normalized business disruption cost (c_d) as defined in Dey et al. (2015), we tried multiple values: $c_s, c_d \in \{5, 10, 15, 20, 25, 30, 35, 40\}$.

In each experiment run, we first select a combination of c_s and c_d values, then compute the optimal TDS policy and the optimal hybrid policy, and subsequently simulate the synchronization operations using the two policies and record their costs.

Table D2 summarizes the percentage of cost saving achieved by the TDS policy compared to the hybrid policy. The results show that the cost saving is consistently positive under all scenarios, ranging from 3.84% to 10.22%. This clearly shows that the TDS policy is a superior method when compared with the hybrid method.

Table D2. Performance Comparison: (Cost of Hybrid – cost of TDS)/Cost of Hybrid

	c_d=5	c_d=10	c_d=15	c_d=20	c_d=25	c_d=30	c_d=35	c_d=40
c_s=5	6.84%	6.72%	6.84%	6.87%	6.90%	6.69%	6.84%	6.59%
c_s=10	7.88%	9.24%	10.20%	10.15%	10.15%	10.18%	10.22%	9.89%
c_s=15	6.95%	9.89%	9.24%	9.58%	9.56%	9.59%	9.55%	9.55%
c_s=20	5.91%	9.19%	9.45%	9.21%	9.22%	9.23%	9.33%	9.24%
c_s=25	4.86%	8.81%	9.33%	9.09%	9.03%	9.36%	9.09%	9.09%
c_s=30	5.03%	8.42%	8.81%	9.25%	8.95%	8.76%	8.93%	8.93%
c_s=35	4.57%	7.97%	8.76%	9.12%	9.10%	9.09%	9.10%	9.04%
c_s=40	3.84%	7.52%	8.35%	8.96%	8.96%	8.82%	8.64%	8.55%

To understand how the hybrid policy works, we also record the numbers of times that synchronizations are trigged by the time-based threshold and total control threshold, as summarized in Table D3. The results are as expected —when the normalized business disruption cost (c_d) is low and dominated by the normalized patching setup cost (c_s), the synchronizations are mostly triggered by the total control threshold; when the normalized business disruption cost (c_d) dominates the normalized patching setup cost (c_s), synchronizations are mostly triggered by the time-based threshold. When $c_d \geq 20$, among all the scenarios, only one synchronization is triggered by the total control-based threshold and all other synchronizations are initiated by the time-based threshold.

Table D3. Numbers of Synchronizations Triggered by Time-based and Total Control Thresholds

	c_d=5	c_d=10	c_d=15	c_d=20	c_d=25	c_d=30	c_d=35	c_d=40
c_s=5	(4336, 282)	(4742, 1)	(4743, 0)	(4742, 0)	(4743, 0)	(4743, 0)	(4743, 0)	(4743, 0)
c_s=10	(2184, 949)	(3340, 11)	(3353, 0)	(3353, 0)	(3353, 0)	(3353, 0)	(3353, 0)	(3353, 0)
c_s=15	(1233, 1293)	(2662, 59)	(2737, 1)	(2738, 0)	(2738, 0)	(2738, 0)	(2738, 0)	(2738, 0)
c_s=20	(709, 1483)	(2174, 167)	(2369, 2)	(2371, 0)	(2371, 0)	(2371, 0)	(2371, 0)	(2371, 0)
c_s=25	(454, 1531)	(1794, 285)	(2110, 9)	(2121, 0)	(2121, 0)	(2121, 0)	(2121, 0)	(2121, 0)
c_s=30	(319, 1502)	(1488, 398)	(1905, 28)	(1936, 0)	(1936, 0)	(1936, 0)	(1936, 0)	(1936, 0)
c_s=35	(232, 1459)	(1270, 472)	(1748, 37)	(1792, 0)	(1792, 0)	(1792, 0)	(1792, 0)	(1792, 0)
c_s=40	(166, 1429)	(1065, 553)	(1591, 72)	(1675, 1)	(1676, 0)	(1676, 0)	(1676, 0)	(1676, 0)

Why the TDS Policy Outperforms the Hybrid Policy?

To understand why the TDS policy consistently outperforms the hybrid policy, we first examine how the hybrid policy works. The hybrid policy tries to take advantage of the strengths of both the time-based policy and the total control policy. Which policy plays a more dominant role largely depends on the relative size of the business disruption cost:

(1) When the disruption cost is low in relation to the setup cost, synchronization operations under the hybrid policy will be primarily triggered by its total control policy component. This result comes with an extra cost — most synchronization operations under the hybrid policy will incur business disruption cost, while the disruption cost is never incurred under the TDS policy. As validated by our experimental results, this difference itself can erode any small theoretical advantage that the total control policy may enjoy over the TDS policy.

(2) When the disruption cost is high compared to the setup cost, synchronizations under the hybrid policy will be primarily triggered by the time-based policy. As we have shown in the paper, when the time-based policy (with an optimized synchronization interval) is a standalone policy, it is dominated by the TDS policy. This result is theoretically intuitive because synchronization at every decision epoch (by setting the synchronization threshold extremely low) is a possible scenario under the TDS policy. When the time-based policy is a component of the hybrid policy, as shown in Theorem 3 of Dey et al. (2015), its optimal time interval threshold (x_H^*) increases from that of the standalone policy (x^*). When comparing the time-based policy against the TDS policy, increasing the time interval of the former generally makes it worse when compared with the later. Although the lower synchronization frequency issue can be compensated by the inclusion of the total control policy, any synchronization triggered under the total control policy comes at an extra cost — the business disruption cost. With all factors considered, the TDS policy should outperform the hybrid policy under this scenario.

In sum, the dynamic nature of the TDS policy (running synchronization only when the benefit is greater than the cost) and the fact that it can avoid business disruption cost bring too much of an advantage for the hybrid policy to overcome, hence the TDS policy can outperform the hybrid policy.

Reference

Dey, D., Lahiri, A., and Zhang, G. 2015. "Optimal Policies for Security Patch Management," *INFORMS Journal on Computing* (27:3), pp. 462-477.

Silvers, F. 2011. *Data Warehouse Designs: Achieving ROI with Market Basket Analysis and Time Variance,* Boca Raton, FL: CRC Press.