

Chapter 2

Literature and Research Review

Abstract Quality-aware scheduling is a novel term employed in our study. However, the related work on this topic appeared in the 1990s. In this chapter, we briefly summarize the literature and research work on quality-aware scheduling. On the one hand, we overview the typical metrics for quality-aware scheduling, including QoS metrics and QoD metrics. On the other hand, we analyze the research progress on quality-aware scheduling in data management systems such as real-time database management system, data stream management system, relational database management system, and key-value data stores. In particular, the goal of quality-aware scheduling includes optimizing QoS, optimizing QoD, as well as optimizing both QoS and QoD. In the following, Sect. 2.1 illustrates metrics for quality-aware scheduling in term of QoS and QoD; Sect. 2.2 describes the research progress on quality-aware scheduling in data management system; Sect. 2.3 summarizes this chapter.

Keywords QoS Metrics · QoD Metrics · Related Work

2.1 Metrics for Quality-Aware Scheduling

How to evaluate the QoS and QoD is a critical issue in the context of quality-aware scheduling. In this section, we briefly overview some metrics for QoS as well as QoD.

2.1.1 QoS Metrics

In the context of data management system, a service-level agreement (SLA) between clients and services engage is a formally negotiated contract where a client and a service agree on several system-related characteristics, which most prominently include the clients expected request rate distribution for a particular API and the expected service latency under those conditions [1]. SLAs are used to indicate the profits the service provider may obtain if the service is delivered at certain levels, and the penalty the service provider has to pay if the agreed-upon performance is not met.

Hence, quality of service (QoS) metrics are employed to depict deviations between system behavior and prespecified SLA which evaluates the overall performance of data management systems. There exist many forms of different metrics and we list some of them in the following:

- *Query response time*: The query response time is an intuitive way to measure QoS. In general, the shorter response time is, the more benefit is or the lower cost is. Formally, a function $cost(t)$ is adopted to illustrate the corresponding cost if the response time is t . Typically, $cost(t)$ is a piecewise linear function [2] as the examples shown in Fig. 2.1. In the figure, supposing the query arrives to the system at time 0, the x -axis and y -axis represents query response time t and the cost corresponding to different response time t , respectively. For a number of queries, the total cost is simply summarized as follows:

$$cost = \sum cost(t)$$

- *Deadline miss ratio*: To specify SLA, queries or transactions are usually assigned deadlines. Clearly, queries respond before the deadlines meet the requirements. Otherwise, the requirements are not satisfied. Hence, considering a number of queries, minimizing the number of missed deadlines would improve QoS [3]. Alternatively, the deadline miss ratio is adopted to depict the QoS and defined as

$$MissRate = 100 \times \frac{\#missed}{\#missed + \#met}(\%)$$

where $\#missed$ and $\#met$ represent the number of queries that have missed and met their deadlines, respectively.

- *System availability*: Availability is the probability that a system will work as required during the period of a mission. In the context of QoS metrics, availability is based on observations after running a system for a period. Specifically,

$$Availability = 100 \times \frac{T_m}{T_m + T_d}(\%)$$

where T_m is the mission duration and T_d is the observed down time.

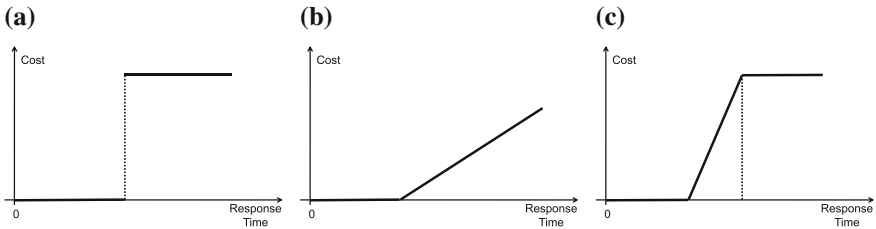


Fig. 2.1 Examples of cost function related to query response time

- *System resources utilization*: The utilization is usually employed to evaluate the utility of each individual node. It is the percentage of the actual running time versus the total running time and formally illustrated as

$$Utilization = 100 \times \frac{T_{total} - T_{idle}}{T_{total}} (\%)$$

where T_{total} is the total running time and T_{idle} is the idle time during system running.

- *System throughput*: The throughput is the number of completed transactions per time unit and the typical time unit is second. Formally, the throughput is specified as

$$Throughput = \frac{\#completedTrans}{T_{total}}$$

where $\#completedTrans$ is the number of completed transactions and T_{total} is total system running time.

The result by metrics above is usually acquired by a long-term observation. In addition to these long-term metrics, transient performance metrics such as overshoot and settling time are adopted for system's responsiveness and efficiency of QoS adaptation.¹

- *Miss ratio overshoot*: Miss ratio overshoot is the maximum amount that the system overshoots its miss ratio reference divided by its miss ratio reference as

$$M_o = \frac{M_{max} - M_s}{M_s}$$

where M_{max} is the maximum miss ratio and M_s is the desired miss ratio reference. In particular, this reference represents the desired system performance. For example, a particular system may require a deadline miss ratio $M_s = 0$.

- *Utilization overshoot*: Utilization overshoot is the maximum amount that the system overshoots its utilization reference divided by its utilization reference as

$$U_o = \frac{U_{max} - U_s}{U_s}$$

where U_{max} is the maximum utilization and U_s is the desired utilization reference. Similarly, this reference represents the desired system performance. For instance, a particular system may require a CPU utilization $U_s = 90\%$.

- *Settling time*: Settling time takes the system to enter a steady state in response to a load profile. The time represents how fast the system can settle down to steady state with desired miss ratio and/or utilization.

Typically, transient metrics are useful for the performance specification of dynamic systems where performance can be time-varying [5].

¹Reprinted from Ref. [4], with kind permission from Springer Science+Business Media.

2.1.2 QoD Metrics

Quality of data (QoD) is very similar to another terminology, i.e., data quality (DQ). In term of DQ, the data are deemed of high quality if they correctly represent the real-world construct to which they refer. Furthermore, apart from these definitions, as data volume increases, the question of internal consistency within data becomes paramount, regardless of fitness for use for any external purpose, e.g., a person's age and birth date may conflict within different parts of a database [6]. Typically, DQ has a large number of dimensions including accessibility, believability, completeness, reputation, security, etc [7]. However, in the context of quality-aware scheduling, QoD is presented as how "good" the served data are. Goodness of data can be measured in metrics such as freshness that need to be defined from the semantics of the application [8].

In this section, we present an overview of QoD metrics that can be used to measure the freshness of data objects. We present three typical different metrics, i.e., time-based metric, lag-based metric, and divergence-based metric.²

- *Time-based metric*: Time-based metric uses the time elapsed from the previous update to quantify how stale a certain data object is. In particular, a function $f(\Delta t)$ is defined to map the time elapsed Δt to another space to describe QoD. For example,

$$f : \Delta t \rightarrow [0, 1]$$

is a mapping of the time since last update to a 0–1 range, which is defined similarly to the QoS curves from Aurora [10]. There is an initial period after the last update, for which the value is considered valid and the data item is fresh (and thus the QoD has a value of 1). After this period, the freshness of the data object declines according to a function (which can be linear or any other monotonically decreasing function). The freshness of the data object keeps dropping until 0 and remains 0 after that.

- *Lag-based metric*: Lag-based metric uses the number of unapplied updates to quantify how stale a certain data object is. For instance, the freshness of an object is measured as a decreasing function of the number x of updates missed, and specifically

$$f(x) = a^x \quad x = 0, 1, \dots$$

where a is defined as the freshness decay rate and has a value between 1 and 0. In semantic, as each object has its own freshness decay rate, a is a knob that can record the varying characteristics of the data objects with regards to freshness or be used to record user preferences. Clearly, there are two special cases, i.e., $a = 1$ and $a = 0$. The value $a = 1$ indicates the user considers the object perfectly fresh no matter how many updates it is lagging; whereas, the value $a = 0$ corresponds to an extremely demanding user who just considers perfectly up-to-date object as

²Reprinted from Ref. [9], with kind permission from Springer Science+Business Media.

useful one. In particular, the latter is a special case for a binary model [8] where the freshness is 0 if something is stale and 1 if something is fresh.

- *Divergence-based*: Divergence-based metric compares the current version with the most up-to-date version and quantify the difference in values (i.e., the divergence). As the case that data source with stock prices and real-time weather information stored continuously sends updates remote repository [11], the absolute value difference is employed as the divergence metric between a data source S and a data repository P . Suppose x_i^S and x_j^P represent the value of x at S and P , respectively. Let the next update at S be x_{i+1}^S . Thus, the divergence-based value difference between S and P could be presented as

$$|x_j^P - x_{i+1}^S|$$

Instead of absolute difference, normalize the difference (by dividing with the current value) to compute a relative percentage is alternative. Obviously, the relative percentage approach cannot be used in cases where the value domain includes 0.

QoD metrics should be selected according to specific scenarios. Generally, the time-base metrics are especially useful in distributed environments where the exact time of the next update is not known [12]; whereas, lag-based metric metrics are especially useful when we have exact information on the upcoming updates. Divergence-based metrics are especially useful when data object are simple values (e.g., a stock price) but do not work as well on complex data object such as entire web page since it is difficult to accurately quantify the difference between two arbitrary HTML fragments [9].

2.2 Quality-Aware Scheduling in Data Management System

Quality-aware scheduling problem has been studied since 1990s in data management system. These work locates in the context of real-time database management system (RTDBMS), data stream management system (DSMS), and relational database management system (RDBMS), key-value data stores. In the following, we will illustrate quality-aware scheduling in these data management systems.

2.2.1 Quality-Aware Scheduling in RTDBMS

A real-time database management (RTDBMS) is a database system which uses real-time processing to handle workloads whose state is constantly changing. This differs from traditional databases containing persistent data, mostly unaffected by time. For example, a stock market changes very rapidly and is dynamic. Real-time processing means that a transaction is processed fast enough for the result to come

back and be acted on right away. RTDBMS is useful for accounting, banking, law, medical records, multimedia, process control, reservation systems, and scientific data analysis [13].

Transaction processing depends on the scheduling strategies adopted by the RTDBMS. To accelerate this processing (i.e., maximize QoS), earliest deadline first (EDF) is used for task scheduling in real-time database. However, it performs bad under heavy workload. Haritsa et al. [3] employs classical EDF to propose AED (adaptive earliest deadline) which aims to reduce the number of missed deadlines. The main idea of AED is to rectify EDF for the context of data access by adding a feedback control mechanism that detects overload conditions so that to modify transaction priority assignments accordingly. In addition, Haritsa et al. [3] also proposes HED (Hierarchical Earliest Deadline), in order to minimize the weighted sum of missed deadlines. It is noticeable that, Buttazzo et al. [14] studies the task scheduling in overloaded context, and argues that EDF is suited for light workload; whereas, SJF (Shortest Job First) performs better than EDF in heavy workload. In general, a real-time database processes various types of tasks (e.g., hash join and external sorting) and it makes sense any deadline misses across the different classes are around a prespecified ratio [15]. Motivated by that, Pang et al. [15] proposes PAQRS (priority adaptation query resource scheduling) which is specifically developed to handle multiclass query workloads in order to minimize the number of missed deadlines. Further, it also ensures that any deadline misses are scattered across the different query classes according to an administratively defined miss distribution.

To improve QoD in RTDBMS influenced by updates, Adelberg et al. [16] discusses several basic strategies on how to apply updates in real-time database systems including: applying updates first, applying updates on demand, split updates, etc. Specifically, the strategy based on applying updates first would install an update whenever it arrives at the system so that all updates have higher priority than queries; whereas, in split updates, updates to important data will be applied on arrival and updates to less important data will be applied when no queries are waiting [16]. Finally, in applying updates on demand, the execution of pending updates is coupled with the arriving of queries, where all the data objects read by a certain query are refreshed on demand before the execution of that query.

In order to guarantee on deadline miss ratio (i.e., maximize QoS) and data freshness (i.e., maximize QoD), Kang et al. [17] proposes QMF (QoS-sensitive approach for Miss ratio and Freshness guarantees) that considers the relationship between updates and queries and utilizes the information to apply the updates required frequently by queries immediately and apply other updates on demand. While that policy considers the dependency between queries and updates, it prioritizes queries based on EDF, which is clearly not an effective policy, especially when system load is high. After that, Kang et al. [18] employs admission controller to improve QMF. When overloaded, update can be relaxed within the specified range of QoD to reduce the update workload if necessary. The modified QMF is just suited for the scenario with firm deadlines, although the performance is improved.

2.2.2 Quality-Aware Scheduling in DSMS

Data stream management system (DSMS) [19] is used to manage continuous data streams and widely adopted in applications such as telecommunication network and finance. It is similar to a database management system (DBMS), which is, however, designed for static data in conventional databases. A DSMS also offers a flexible query processing so that the information needed can be expressed using queries. However, in contrast to a DBMS, a DSMS executes a continuous query that is not only performed once, but also is permanently installed. Therefore, the query is continuously executed until it is explicitly uninstalled. Since most DSMS are data-driven, a continuous query produces new results as long as new data arrive at the system [20].

To reduce query response time (i.e., optimize QoS), Urhan and Franklin [21] proposes the rate-based (RB) policy to schedule a single multistream query with join operators for a query plan in pipeline execution. Sharaf et al. [22, 23] extends RB to schedule multiple continuous queries and proposes HNR (highest normalized rate) so as to reduce the total query response time. In particular, for multiple aggregate continuous queries, Guirguis et al. [24] proposes Weave Share to improve a globe performance by reordering the sequence and sharing partial aggregations. After that, Guirguis et al. [25] employs TriOps to model aggregate continuous queries and proposes TriWeave to further improve the performance on multiple aggregate continuous queries.

To reduce the data staleness (i.e., improve QoD), Golab et al. [26] discusses updating a data warehouse that collects near-real-time data streams from a variety of external sources and proposes EDF-P (Prioritized EDF), MB (Max Benefit) MBL (Max Benefit with Lookahead), etc. The objective is to keep all the tables and materialized views up-to-date as new data arrive over time. Sharaf et al. [27] proposes FAS-MCQ (freshness-aware scheduling of multiple continuous queries) to improve the freshness of results from multiple continuous queries.

In order to balance the trade-off between QoS and QoD, Sharaf et al. improves FAS-MCQ [27] by adding β ($0 \leq \beta \leq 1$) to propose FAS-MCQ(β) [28]. In particular, FAS-MCQ(β) becomes the original FAS-MCQ when $\beta = 0$, whereas FAS-MCQ(β) behaviors like HNR [22, 23] when $\beta = 1$.

2.2.3 Quality-Aware Scheduling in RDBMS

Relational database management system (RDBMS) is built on the foundation of relational data model and exploits the concepts and techniques on algebra of sets to process data. Relational database is widely adopted in many scenarios. For instance, RDBMS serves as a data platform for various web applications and employed for construction of data warehouse.

Generally, transactions in web applications typically employing SLA to specify QoS are associated with soft deadlines which express the performance expectations

of the end user and beyond which transactions are not dropped but are still processed to completion. In this context, minimizing transaction tardiness (i.e., amount of deviation from deadline) rather than the number of miss deadlines is a more appropriate performance goal. As shown in Sect. 2.2.1, EDF is suited for light workload, whereas SJF (shortest job first) performs better than EDF in heavy workload. Sharaf et al. [29] combines EDF with SJF to propose ASETS which is adaptive to the variability of workloads without additional parameters. Guirguis et al. [30] extends ASETS from web transaction level to workflow level including a sequence of web transactions and proposes ASETS* strategy. Sharaf et al. [31] proposes TraQIOS to optimize I/O-intensive transactions in highly interactive applications based on the trade-off between EDF and SJF. In addition, online view materialization is a common approach to process web transaction. Labrinidis and Roussopoulos [9] proposes OVIS(θ) algorithm for the view selection problem, in order to optimize QoS given QoD.

Typically, data replication improves system performance or availability [32]. For instance, data warehouse copies the remote transaction data locally so as to do analysis quickly and web search engine replicates partial web pages to accelerate searching. Hence, how to synchronize the different replicas is very important to optimize QoD. In order to capture users' expectation on QoD, Guo et al. [33] extends SQL by adding C&C constrains which allows users to specify the requirement on data freshness as well as consistency. In addition, Cho and Garcia-Molina [34] studies the update scheduling strategies such as FO (fixed order), RO (random order), and PR (purely random) to maximize the freshness of local replicas. Further, updates influence both the freshness of local replicas and the one of derived data such as materialized views. In addition, Labrinidis and Roussopoulos [8] explores the impact of updates on materialized views and proposes QoDA (QoD-Aware) to find an optimal sequence to refresh materialized view.

To specify users' expectation on query latency and data freshness, Bright and Raschid [35] schedules the transition of web data according to LRP (latency/recency profile). To further specify users' requirement of QoS and QoD, Qu et al. [36] explores USM (user satisfaction metric) to combine users' satisfaction on the query response (i.e., QoS) and data freshness (i.e., QoD) and proposes UNIT framework to manage transaction scheduling. However, USM depicts the QoS and QoD from system aspect and is oblivious to the expectation or preference of individual user. Hence, Labrinidis and Roussopoulos [37] proposes Quality Contracts (QC) which is a framework based on the microeconomic paradigm and provides a very powerful way for users to specify their preferences for query response (QoS) and data freshness (QoD) by specifying how much benefits he/she thinks the server should get at various levels of quality for the posed query. Based on QC model, Qu and Labrinidis [38] proposes a scheme named QUTS (Query Update Time Sharing) that allocates CPU resources to queries and updates based on proportion of QoD and QoS profits assigned by users to maximize total profit.

The classical view of a distributed DBMS is that it should behave just like a centralized DBMS from the point of view of a user; issues arising from distribution of data should be transparent of the user, although, of course, they must be addressed at the implementation level [39]. Hence, it usually employs strong consistency which

means that once a write request returns successfully to the client, all subsequent reads of the object—by any client—see the effect of the write. Differently, key-value stores often adopt weak consistency. For instance, Dynamo is designed to be an eventually consistent data store where all updates reach all replicas eventually; whereas, PNUTS provides a timeline consistency model where all replicas will go through the same sequence of updates such that they will eventually converge to the latest update made by the application. Hence, the data provided by API of key-value stores might be stale. For example, if the settings of parameters, i.e., R (read quorum), W (write quorum), and N (the number of replicas), in Dynamo do not meet $R + W > N$, then the result from `get()` might be stale. When consistency level of `get()` in Cassandra is set to be ONE, this request would read any replica of a data object so as to return a stale version. Similarly, Read-any in PNUTS might read a stale data object.

2.2.4 Quality-Aware Scheduling in Key-Value Stores

Similar to the aforementioned data management systems, optimizing QoS and QoD is also very important to key-value stores. In order to optimize QoS, Chi et al. [40] proposes SLA-tree which supports SLA-based scheduling toward benefit optimization in cloud data management system such as key-value stores. Here, SLA is represented as a stepwise function. After that, Chi et al. [2] employs piecewise linear function to describe SLA and provides cost-based iCBS algorithm to support query scheduling in cloud environment. In particular, the cost is evaluated by the mean of the population as the expected running time for a query. However, the execution time distributions might be far from each other. Hence, Chi et al. [41] proposes Shepherd algorithm to exploit distributions of expected running times when doing SLA-aware query scheduling.

In order to improve the performance on query response, key-value stores typically adopt data replication so as to incur data inconsistency potentially which represents as QoD issues at end user. Bailis et al. [42] introduces PBS (probabilistically bounded staleness) to measure the consistency for practical quorum systems such as Dynamo. Then, Bailis et al. [43] illustrates the function of PBS in practice by a microblogging demonstration named DBSoc similar to Twitter. In addition, for weak consistency key-value stores, Zhu et al. [44] studies how to maximize replica consistency (i.e., QoD) within the specified latency (i.e., QoS) constrained by a prototype named RECODS. Golab et al. [45] considers how to detect a consistency violation as soon as it happens as well as quantifying the severity of such violations.

In the context of modern NoSQL key-value data stores, studying the trade-off between latency and consistency has gained special attention and attracted several research efforts. Abadi [46] studies CAP [47] theory in distributed system and rewrites CAP as PACELC. That is, if there is a partition (P), how does the system trade-off availability and consistency (A and C); else (E), when the system is running normally in the absence of partitions, how does the system trade-off

Table 2.1 The design choice of key-value stores under PACELC

System	Partition		Else	
	Availability	Consistency	Latency	Consistency
Dynamo	✓		✓	
Cassandra				
Riak				
BigTable		✓		✓
HBase				
MongoDB	✓			✓
PNUTS		✓	✓	

latency (L) and consistency (C)? Table 2.1 lists the design choices of some key-value stores according to PACELC which emphasizes the importance of the trade-off between latency/consistency(L/C). Ramakrishnan [48] argues that the L/C trade-off has become a key factor in the design of large-scale data management systems. Wada et al. [49] investigates what consumers observe in terms of consistency/latency properties, i.e., the ability to optimize QoS and QoD, under various implementations of key-value data stores such as SimpleDB and S3.

All this work, however, the latency versus consistency trade-off at a global-level is studied and platform-level techniques to model and control such trade-off (e.g., update propagation models, quorum settings, etc.) is employed. While those techniques might be effective in directing the incoming queries and updates to the right nodes at the right time, it is still crucial to efficiently allocate the resources necessary to execute such operations in that final stage in which they have arrived at their destination nodes, especially at times of high load. In particular, it is important that the resource allocation taking place at the node-level works in synergy with the global-level platform design and enforces meeting its expectations and specifications for latency and consistency, which is exactly the focus of the work presented in this paper.

2.3 Summary

In this chapter, we first illustrate the metrics for quality-aware scheduling, including QoS and QoD metrics (Sect. 2.1). Then, we introduce the quality-aware scheduling in data management system (Sect. 2.2), including real-time database management system (Sect. 2.2.1), data stream management system (Sect. 2.2.2), relational database management system (Sect. 2.2.3), and key-value data stores (Sect. 2.2.4). In particular, there are plenty of algorithms, theories, and prototypes with respect to quality-aware scheduling in data management systems. Figure 2.2 depicts them according to the conference or journal published, year, types of data management

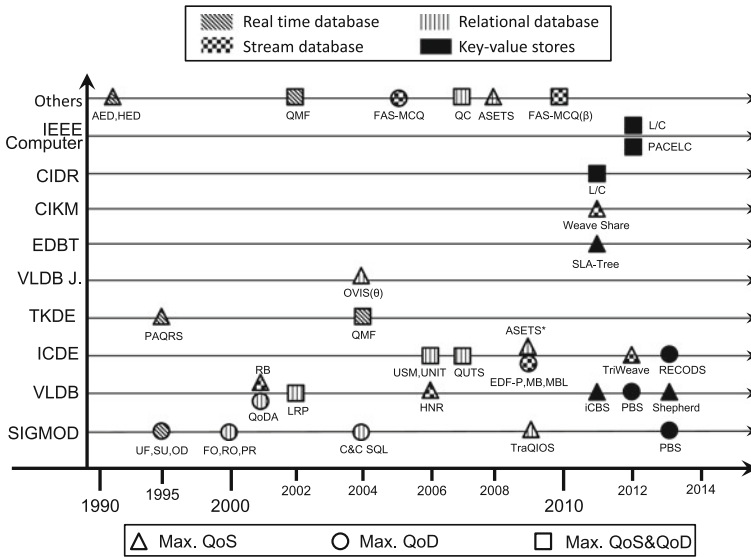


Fig. 2.2 The research progress of quality-aware scheduling in data management systems

systems, and optimization goal. Clearly, there are many papers published on prestigious conferences (e.g., SIGMOD, VLDB, and ICDE) of database community. Hence, quality-aware scheduling is a important issue in data management systems and it calls for further studies for the one in key-value stores.

References

1. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Voshall, P., Vogels, W.: Dynamo: Amazon's highly available key-value store. In: Proceedings of SOSP, pp. 205–220 (2007)
2. Chi, Y., Jin Moon, H., Hacigümüş, H.: ICBS: Incremental cost-based scheduling under piecewise linear SLAS. PVLDB **4**(9), 563–574 (2011)
3. Haritsa, J.R., Livny, M., Carey, M.J.: Earliest deadline scheduling for real-time database systems. In: Proceedings of RTSS, pp. 232–242 (1991)
4. Chenyang, L., Stankovic, J.A., Son, S.H., Tao, G.: Feedback control real-time scheduling: framework, modeling, and algorithms. Real-Time Syst. **23**(1), 85–126 (2002)
5. Wei, Y., Son, S.H., Stankovic, J.A., Kang, K.-D.: QoS management in replicated real time databases. In: Proceedings of RTSS, pp. 86–97 (2003)
6. Roebuck, K.: Data Quality: High-Impact Strategies—What You Need to Know: Definitions, Adoptions, Impact, Benefits, Maturity, Vendors. Emereo Pty Limited (2011)
7. Pipino, L., Lee, Y.W., Wang, R.Y.: Data quality assessment. Commun. ACM **45**(4), 211–218 (2002)
8. Labrinidis, A., Roussopoulos, N.: Update propagation strategies for improving the quality of data on the web. In: Proceedings of VLDB, pp. 391–400 (2001)

9. Labrinidis, A., Roussopoulos, N.: Exploring the tradeoff between performance and data freshness in database-driven web servers. *VLDB J.* **13**(3), 240–255 (2004)
10. Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Lee, S., Seidman, G., Stonebraker, M., Tatbul, N., Zdonik, S.B.: Monitoring streams—a new class of data management applications. In: *Proceedings of VLDB*, pp. 215–226 (2002)
11. Shah, S., Ramamritham, K., Shenoy, P.J.: Maintaining coherency of dynamic data in cooperating repositories. In: *Proceedings of VLDB*, pp. 526–537 (2002)
12. Cho, J., Garcia-Molina, H.: Effective page refresh policies for web crawlers. *ACM Trans. Database Syst.* **28**(4), 390–426 (2003)
13. Wikipedia: Real-time database. http://en.wikipedia.org/wiki/Real-time_database#cite_note-1
14. Buttazzo, G.C., Spuri, M., Sensini, F.: Value vs. deadline scheduling in overload conditions. In: *Proceedings of RTSS*, pp. 90–99 (1995)
15. Pang, H.H., Carey, M.J., Livny, M.: Multiclass query scheduling in real-time database systems. *IEEE Trans. Knowl. Data Eng.* **7**(4), 533–551 (1995)
16. Adelberg, B., Garcia-Molina, H., Kao, B.: Applying update streams in a soft real-time database system. In: *Proceedings of SIGMOD Conference*, pp. 245–256 (1995)
17. Kang, K.D., Son, S.H., Stankovic, J.A., Abdelzaher, T.F.: A QoS-sensitive approach for timeliness and freshness guarantees in real-time databases. In: *Proceedings of ECRTS*, pp. 203–212 (2002)
18. Kang, K.D., Son, S.H., Stankovic, J.A.: Managing deadline miss ratio and sensor data freshness in real-time databases. *IEEE Trans. Knowl. Data Eng.* **16**(10), 1200–1216 (2004)
19. Koudas, N., Srivastava, D.: Data stream query processing: a tutorial. In: *Proceedings of VLDB*, p. 1149 (2003)
20. Wikipedia: Data stream management system. http://en.wikipedia.org/wiki/Data_stream_management_system
21. Urhan, T., Franklin, M.J.: Dynamic pipeline scheduling for improving interactive query performance. In: *Proceedings of VLDB*, pp. 501–510 (2001)
22. Sharaf, M.A., Chrysanthis, P.A., Labrinidis, A., Pruhs, K.: Efficient scheduling of heterogeneous continuous queries. In: *Proceedings of VLDB*, pp. 511–522 (2006)
23. Sharaf, M.A., Chrysanthis, P.K., Labrinidis, A., Pruhs, K.: Algorithms and metrics for processing multiple heterogeneous continuous queries. *ACM Trans. Database Syst.* **33**(1), 5 (2008)
24. Guirguis, S., Sharaf, M.A., Chrysanthis, P.A., Labrinidis, A.: Optimized processing of multiple aggregate continuous queries. In: *Proceedings of CIKM*, pp. 1515–1524 (2011)
25. Guirguis, S., Sharaf, M.A., Chrysanthis, P.K., Labrinidis, A.: Three-level processing of multiple aggregate continuous queries. In: *Proceedings of ICDE*, pp. 929–940 (2012)
26. Golab, L., Johnson, T., Shkapenyuk, V.: Scheduling updates in a real-time stream warehouse. In: *Proceedings of ICDE*, pp. 1207–1210 (2009)
27. Sharaf, M.A., Labrinidis, A., Chrysanthis, P.K., Pruhs, K.: Freshness-aware scheduling of continuous queries in the dynamic web. In: *Proceedings of WebDB*, pp. 73–78 (2005)
28. Sharaf, M.A., Chrysanthis, P.K., Labrinidis, A.: Tuning QoS in stream processing engines. In: *Proceedings of ADC*, pp. 103–112 (2010)
29. Sharaf, M.A., Guirguis, S., Labrinidis, A., Pruhs, K., Chrysanthis, P.K.: Poster session: ASETS: a self-managing transaction scheduler. In: *Proceedings of ICDE Workshops*, pp. 56–62 (2008)
30. Guirguis, S., Sharaf, M.A., Chrysanthis, P.K., Labrinidis, A., Pruhs, K.: Adaptive scheduling of web transactions. In: *Proceedings of ICDE*, pp. 357–368 (2009)
31. Sharaf, M.A., Chrysanthis, P.K., Labrinidis, A., Amza, C.: Optimizing i/o-intensive transactions in highly interactive applications. In: *Proceedings of SIGMOD Conference*, pp. 785–798 (2009)
32. Gellersdörfer, R., Nicola, M.: Improving performance in replicated databases through relaxed coherency. In: *Proceedings of VLDB*, pp. 445–456 (1995)
33. Guo, H., Larson, P., Ramakrishnan, R., Goldstein, J.: Relaxed currency and consistency: how to say good enough in SQL. In: *Proceedings of SIGMOD Conference*, pp. 815–826 (2004)
34. Cho, J., Garcia-Molina, H.: Synchronizing a database to improve freshness. In: *Proceedings of SIGMOD Conference*, pp. 117–128 (2000)

35. Bright, L., Raschid, L.: Using latency-recency profiles for data delivery on the web. In: Proceedings of VLDB, pp. 550–561 (2002)
36. Qu, H., Labrinidis, A., Mossé, D.: Unit: user-centric transaction management in web-database systems. In: Proceedings of ICDE, p. 33 (2006)
37. Labrinidis, A., Qu, H., Xu, J.: Quality contracts for real-time enterprises. In: Proceedings of BIRTE, pp. 143–156 (2006)
38. Qu, H., Labrinidis, A.: Preference-aware query and update scheduling in web-databases. In: Proceedings of ICDE, pp. 356–365 (2007)
39. Ramakrishnan, R., Gehrke, J.: Database Management Systems. The McGraw-Hill Companies Inc, New York (2003)
40. Chi, Y., Moon, H.J., Hacigümüs, H., Tatemura, J.: SLA-tree: a framework for efficiently supporting SLA-based decisions in cloud computing. In: Proceedings of EDBT, pp. 129–140 (2011)
41. Chi, Y., Hacigümüs, H., Hsiung, W.-P., Naughton, J.F.: Distribution-based query scheduling. PVLDB **6**(9), 673–684 (2013)
42. Bailis, P., Venkataraman, S., Franklin, M.J., Hellerstein, J.M., Stoica, I.: Probabilistically bounded staleness for practical partial quorums. PVLDB **5**(8), 776–787 (2012)
43. Bailis, P., Venkataraman, S., Franklin, M.J., Hellerstein, J.M., Stoica, I.: PBS at work: advancing data management with consistency metrics. In: Proceedings of SIGMOD Conference, pp. 1113–1116 (2013)
44. Zhu, Y., Yu, P.S., Wang, J.: RECODS: replica consistency-on-demand store. In: Proceedings of ICDE, pp. 1360–1363 (2013)
45. Golab, W.M., Li, X., Shah, M.A.: Analyzing consistency properties for fun and profit. In: Proceedings of PODC, pp. 197–206 (2011)
46. Abadi, D.: Consistency tradeoffs in modern distributed database system design: cap is only part of the story. IEEE Comput. **45**(2), 37–42 (2012)
47. Gilbert, S., Lynch, N.A.: Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News **33**(2), 51–59 (2002)
48. Ramakrishnan, R.: Cap and cloud data management. IEEE Comput. **45**(2), 43–49 (2012)
49. Wada, H., Fekete, A., Zhao, L., Lee, K., Liu, A.: Data consistency properties and the trade-offs in commercial cloud storage: the consumers’ perspective. In: Proceedings of CIDR, pp. 134–143 (2011)



<http://www.springer.com/978-3-662-47305-4>

Quality-aware Scheduling for Key-value Data Stores

Xu, C.; Zhou, A.

2015, XI, 97 p. 32 illus., Softcover

ISBN: 978-3-662-47305-4