Data Management Research at NEC Labs

Data Management Research Group
NEC Laboratories America
http://www.nec-labs.com/dm/
{hakan}@sv.nec-labs.com

1. INTRODUCTION

In 2009, NEC Laboratories of America started a data management research department to create a world-class team and a research program with the dual goal of research excellence and direct contributions to the company's global business. Our organization gives the researchers opportunities to maintain a balanced mix of fundamental and applied research as they focus on innovations which are motivated by the real needs faced by the company's large service and product businesses. In this report, we present an overview of the research program of the Data Management Research group, which currently includes Yun Chi, Hakan Hacıgümüş, Wang-Pin Hsiung, Bin Liu, Ziyang Liu, Hyun Jin Moon, Oliver Po, Jagan Sankaranarayanan, Junichi Tatemura, and our close collaborators, Michael Carey from University of California, Irvine, Hector Garcia-Molina from Stanford University, Jeffrey Naughton and Jignesh Patel from University of Wisconsin, Madison. The group is also engaged with numerous academic organizations through our University Relations program.

The current focus of the group is data management in the cloud. Cloud computing has emerged as a promising computing and business model. By providing on-demand scaling capabilities without any large upfront investment or long-term commitment, it is attracting a wide range of users. It is obvious that cloud computing presents challenges and opportunities for data management services. For instance, database service providers may face heterogeneous customer workloads with widely varying characteristics. To serve such workloads, they may have to use a diverse set of specialized database products and technologies in an elastic manner to ensure that customers observe the benefits of those products specifically tailored for their needs. The goal of our group is to understand and analyze those challenges and opportunities by identifying and solving the relevant research problems.

2. THE CLOUDDB PLATFORM

Our current research projects are built around a large platform, called *CloudDB*, which is envisioned as a comprehensive data management platform in the cloud [2, 10]. *CloudDB* would provide data management capabilities as a service to transparently and efficiently support diverse application workloads with identifiable SLA guarantees and end-to-end system management functions. The system would be able to employ heterogeneous underlying storage models to effectively meet applications query and scalability requirements. The CloudDB platform has the following guiding principals in its design:

- *One size does not fit all*, hence the platform should embrace the heterogeneity by leveraging specialized database technologies to serve diverse business needs and workloads.
- The *profit (money)* and the customer *Service Level Agreements (SLAs)* should be the main metrics for all system management and optimization decisions.
- The system should maintain the declarative nature of data processing while leveraging diverse set of specialized database technologies underneath.

The effort to build the *CloudDB* platform with these guiding principles has enabled us to identify numerous challenging, exciting, and relevant research and system problems in data management. Our progress in building the CloudDB platform has already generated various significant technologies, which are being evaluated, further developed, and deployed in real business settings. As the CloudDB platform includes a large number of projects, here we only report on the selected subset, which is a good representative of major areas including query processing and optimization over heterogeneous data stores, intelligent resource and workload management for cloud database services, and application areas that leverage the cloud data management capabilities.

3. MICROSHARDING: ELASTICITY FOR OLTP WORKLOADS

The goal of our Microsharding project is to establish a declarative approach to support OLTP type of workloads elastically based on the relational model and to claim that developers do not have to abandon SQL and relational models just because existing RDBMSs are not as elastic as key-value stores [14]. Cloud computing is expected to enable an application to dynamically adapt to growing workloads by increasing the number of servers. Such an approach is often called *scaling in/out*, and the property of systems that enables this is called *elasticity.* Elasticity is an important property for hosting web applications since workloads from web users are often unpredictable, and can change dynamically over short periods of time. However, it remains challenging to deliver elasticity to interactive and data-intensive applications that handle a large number of read and write operations on shared data.

An approach to deliver elasticity to OLTP workloads is to use a family of key-value stores, which enable seamless scaling out (e.g., live data re-partitioning). However, these data stores provide limited query and data manipulation APIs that are much simpler than SQL. Whereas SQL provides a declarative way to query and manipulate data, those APIs require an application developer to code data manipulation logic in a more procedural manner. Such an approach lacks data independence: Change in data organization on a data store (e.g., introducing secondary index objects) involves change in the application code that access the data.

The Microsharding proposes a relational alternative to the industrial state-of-the-art approaches. We believe that the idea of entity groups [3] is practical and effective to achieve elastic OLTP workloads. If the workload and entity groups are designed appropriately, the system can mostly avoid distributed transactions, and scaling out could be made easier. However, the code that accesses data is written in a proprietary and procedural way, making it difficult to take a principled approach to design elastic workloads. Developing such a declarative and principled solution is the main focus and the novel contribution of microsharding.

To realize a relational approach as an alternative for the existing procedural approaches, we identify the following key questions: *1)* How can we define constraints on transactions similar to entity groups in the relational model? *2)* How can we benefit from the relational model to design and analyze elastic OLTP workloads? *3)* How can we implement it?

For the first question, we propose to extend the concept of transaction class [5], which is a description of the behavior of transactions. Our vision is to introduce transaction description language (TDL) to describe various restrictions on ACID properties in the form of transaction classes. We first introduce a primitive transaction class, which is a building block of complex transaction classes. A primitive transaction class defines the smallest unit of logical data partitioning, which is a relational version of entity groups. We call this logical partition a *microshard* and the entire scheme *microsharding.*

For the second question, we examined how TDL enables a principled approach to design databases and application workloads. The physical design (data layout) of the database takes not only schema into account but also transaction classes.

For the third question, we have implemented microsharding methodology as a relational middleware, called *Partiqle*, on top of open source distributed data stores, HBase [1] and Vodemort [4], which demonstrates the benefits of the data independence.

Transaction Class is a key concept in microsharding. We introduce a *transaction class* as a way of declaratively specifying the constraints on the ACID properties of a transaction. The concept of transaction classes was introduced to a distributed database system SDD-1 [5], where a transaction class is to specify data set accessed by a transaction. A transaction class was used as an input of static conflict analysis to optimize transaction protocols without sacrificing global consistency. Here, we use a similar specification to restrict the power of transactions the application can use. Our transaction class is a tool for the developer to design trade-off between consistency and elasticity. Here, we only discuss a basic transaction class, called *primitive transaction class.* More detailed discussion on transaction classes can be found in [14].

A primitive transaction class defines a logical scope of the data where serializability must be maintained. Its specification looks similar to specification of entity groups. Let us take an example from TPC-W benchmark data. Consider supporting a transaction that updates a purchase order, which consists of one record in ORDERS table and records in ORDER_LINE table, which have parent-child (foreign key) relationship. A transaction class for this transaction can be stated by specifying as follows:

```
CREATE TRANSACTION CLASS t1 AS
    ORDERS BY O_ID, ORDER_LINE BY OL_O_ID
```

This statement classifies records of ORDERS and

ORDER_LINE together into groups by the values of O_ID (the primary key) and OL_O_ID (the foreign key to ORDERS), respectively.

We call columns of a table specified in a primitive transaction class *transaction keys*. A value of this key identifies a specific group of data.

This specification is similar to data partitioning in RDBMSs based on reference [7]. The objective of traditional data partitioning is to provide physically partitioned but logically seamless (consistent) data. However, our partitioning defines *logical* boundaries that have an impact on the data consistency semantics. In addition, this logical partitioning is independent from physical partitioning (or layout on distributed data stores). To distinguish this difference, we refer to the logical partitioning as *sharding*. We call the unit of transaction scope a *microshard* and our approach *microsharding*, since the granularity of shards is very small relative to the entire database [1].

4. MAESTRO: RESOURCE AND WORKLOAD MANAGEMENT FOR CLOUDDB

The Maestro project aims at developing a family of technologies to manage very large heterogeneous database clusters that are deployed in cloud service delivery infrastructures. More specifically we look into resource and workload management and optimization based on the key metrics that are relevant for cloud service delivery. Obviously managing the resources and the workloads in a cloud service delivery infrastructure in an optimal manner involves a number of decisions that have to be made in the lifecycles of the systems. The key issue is the identification of the metric on which the system optimizes. The metric should be relevant to database systems characteristics and the cloud computing model. We choose to use *SLA-based profit* as the metric, which is the ultimate goal of service providers, rather than low-level system metrics, such as average query response time. SLA-based profit is identified by two parts: i) the *revenues* (money) and ii) the *operational costs*. The revenue is what service clients pay to the service provider based on the delivery of the services according to the SLAs in the contract between the service provider and the customer. The revenue is not fixed and it may change with potential reduction in payments or even penalties; depending on the service quality, e.g, too high query latency. Operational cost is the cost of resources used to run the service. Hence, the research question in Maestro is how to manage resources and workloads in the system to maximize SLA-based profit, which is SLA-based revenue minus operational cost. We believe applying SLA-based profit optimization to all system components opens up many interesting research challenges and opportunities.

In the Maestro project, we have been developing techniques to achieve our goal of *SLA-based profit optimization* in key areas such as query dispatching, query scheduling, admission control, capacity planning, provisioning, and multitenant database management. There have been many proposals to these problems in the literature. However, most of them addressed the problems by considering lower level system-oriented metrics, such as minimizing average response time or minimizing average slow down. That is, they do not explicitly consider profit optimization. It may seem like average response time minimization will implicitly lead to profit optimization, but it is not the case in general: some jobs may have more expensive SLAs to violate than others, so the cloud provider would want to give a higher priority to those jobs for better overall profit. Our approach, in contrast, explicitly considers SLA revenue function of each job at the core of those listed problems to achieve an overall profit optimization. We experienced that distinctively optimizing the individual system components that correspond to those key areas with a global objective in mind gave us a greater degree of freedom to customize our methods. This approach yielded higher degrees of performance, customizability based on variable business requirements, and end-to-end profit optimization.

The Profit and the SLA Models: The total profit, P, of the cloud service provider is defined as $P = \sum_i r_i - C$, where r_i is the revenue that can be generated by delivering the service for a particular job i and C is the operational cost of running the service delivery infrastructure. We define the revenue, R, for each job class in the system. Each client may have multiple job classes based on the contract. We use piecewise linear functions to characterize the SLA revenue as discussed in [6]. Intuitively, the clients agree to pay varying fee levels for corresponding service levels delivered for a particular class of requests, i.e., job classes in their contracts. For example, the client may be willing to pay a higher rate for lower response times. This characterization allows more intuitive interpretation of SLAs with respect to revenue generation.

[1]We do not use the term *entity* group either in order to avoid confusion between logical schema design and transaction design: For instance, we also plan to introduce *vertical sharding*, making a logical *entity* no longer an atomic unit of transaction scope.

The intuition is that, if the level of services changes, the amount that the provider can charge the client also changes according to the contract. Due to the limitations on the availability of infrastructure resources, the cloud service provider may not be able or choose to attend to all client requests at the highest possible service levels. Dropping/Increasing service levels cause loss/increase in the revenue. The loss of potential revenue corresponds to SLA function. Likewise, increasing the amount of infrastructure resources to increase service levels results in increased operational cost. As a result, the key problem for the provider is to come up with optimal service levels that will maximize its profits based on the agreed upon SLAs.

We note that the quantile-based SLAs are more commonly used in practice – especially for availability measures. If this is preferred, there exist techniques (e.g., [9]) that directly map quantile-based SLAs to per-query SLAs. However, based on our extensive interactions with numerous business organizations that provide services to real clients, they desire to be able to manage SLAs at the finest granularity level (i.e., per query) with multiple levels of delivery defined in the SLAs (i.e., step functions). The observation is that currently majority of the service providers only give availability SLAs to their clients represented in the quantile form but not other types of SLAs such as latency, throughput etc. Also, lack of formal models and tooling to enable finer granularity level SLA management is a major inhibitor for businesses to adopt various types of SLAs and also varying levels. Our research aims at advancing the state-of-the art in that area and helping service providers by working with them.

4.1 iCBS: Efficient Cost-Based Scheduling

In the cloud computing environment, service providers offer vast IT resources to large sets of customers with diverse service requirements. Obviously the service provider may derive different revenues from different clients as the clients may have varying SLA and price agreements. Then one of the key questions the service provider has to answer is: "How should I prioritize the queries in the system for execution in way that will maximize the profits?" Naturally, the answer to this question should consider the constraints, such as available resources, and other inputs, such as customer SLAs etc. The prioritization of queries can be defined as a query scheduling problem, which is our focus in efficient cost-based scheduling project.

The scheduling is a mature problem and has long been extensively studied in various areas such as compute networks, database systems, and Web services. There exist many different scheduling policies and there is a vast amount of theoretical results and practical analysis on various scheduling policies. However, in the majority of existing work on scheduling, the performance metrics are low-level performance metrics (e.g., average query response time or stretch). More recently, researchers and practitioners have paid more and more attention to metrics other than the system-level ones. One such new metric is the cost-based metric. For example, instead of optimizing query response time, the main target of a scheduling policy can be to reduce the total query cost, while for each query there is a certain mapping between its response time and the corresponding query cost. Such a cost-based metric is well suited for cloud computing where profit plays a central role and so we believe it is worth investigating the cost-based scheduling problems in cloud services. We mainly focus on two requirement areas while designing a scheduling algorithm for this purpose. First, we need a competitive computational complexity as we expect a cloud service delivery system has to manage very large query queues, where efficiency is crucial. Second, the scheduling algorithm should be capable of handling comprehensive set of SLA functions that are representative of typically business contact cases. With those requirements in mind, we developed an efficient cost-based scheduling algorithm, called *iCBS* [6] by building on the foundation of a previous work in this area [11, 12]. iCBS has the following main characteristics and advantages: *1)* It has a competitive time complexity of $O(\log^2 N)$ over other SLA-aware scheduling algorithms. The competitive complexity is achieved by using certain techniques in computational geometry thereby making the cost based scheduling feasible for query scheduling in the cloud-based systems. *2)* It can handle wide range of SLA function families.We study cost-based scheduling for a special family of cost functions; namely piecewise linear SLAs. Piecewise linear SLAs are easy to describe in natural language and so preferred for business contracts. We implemented and demonstrated that for many special types of piecewise linear SLAs, iCBS can achieve $O(\log N)$ time complexity. Other than capturing various rich semantics, piecewise linear SLAs also make many computations in cost-aware scheduling more tractable, which is a key to our work.

4.2 SmartSLA: Virtualized Resource Management for Cloud Databases

In a cloud computing environment, multi tenancy is a common practice where resources are shared among different clients. SmartSLA[2] project [16] focuses on intelligently managing and allocating resources among various clients in a multi tenanted cloud database environment. SmartSLA consists of two main components: the system modeling module and the resource allocation decision module. The system modeling module uses machine learning techniques to learn a model that describes the potential profit margins for each client under different resource allocations. Based on the learned model, the resource allocation decision module dynamically adjusts the resource allocations in order to achieve the optimum profits.

More specifically the problem is that he service provider should intelligently allocate limited resources, such as CPU and memory, among competing clients. On the other hand, some other resources, although not strictly limited, have an associated cost. Database replication is such an example. It is known that adding additional database replicas not only involves direct cost (e.g., adding more nodes), but also has initiation cost (e.g., data migration) and maintenance cost (e.g., synchronization). We view the successful management of resources as follows:

Local Analysis : The first issue is to identify the right configuration of system resources (e.g., CPU, memory etc.) for a client to meet the SLAs while optimizing the revenue. Answers to such a question are not straightforward as they depend on many factors such as the current workload from the client, the client-specific SLAs, and the type of resources.

Global Analysis : The second issue that a service provider has to address is the decision on how to allocate resources among clients based on the current system status. For example, how much CPU shares or memory should be given to the gold clients versus the silver ones and when a new database replica should be started, etc.

In the SmartSLA system architecture the system modeling module mainly answers the Local Analysis questions, and the resource allocation decision module is responsible for the Global Analysis questions.

4.3 ActiveSLA: Profit-Oriented Admission Control

Compared to traditional database systems, the databases systems hosted in the cloud usually serve more diverse clients (e.g., through multi-tenancy) and therefore face more unpredictable workloads. Due to economic considerations, cloud database providers try to avoid resource overprovisioning while accounting for simultaneous peak workloads from a large number of clients. Consolidating multiple clients in shared infrastructures is a very commonly used approach by the cloud providers. Such a consolidation affords greater economies of scale and fixed cost distribution. However, managing the overloading becomes a much more crucial problem in such environments.

The admission control has been proposed to resolve the system overloading problem. With admission control, when the system is near an overloading condition, new queries are either throttled (e.g., [8]) or rejected (e.g., [15]) until the system condition improves. Although existing admission control techniques are helpful to alleviate system overloading, they do not work directly toward the main goal of cloud service providers—namely to maximize their profits by satisfying different SLAs.

We have designed and implemented an admission control framework, called ActiveSLA[3], for making prediction-based and profit-oriented admission control decisions, with a target of maximizing the expected profit of the database service providers [17]. The ActiveSLA framework is an end-to-end solution that consists of two main modules: a *prediction module* and a *decision module*. When a new query arrives, the query first enters the prediction module. The prediction module uses machine learning techniques and considers both the characteristics of the query and the current system conditions. The prediction module outputs the probability of the query meeting its deadline. The calculated probability and the query's SLA are sent to the decision module. The decision module decides either to admit the query or to reject the query up-front. Finally, the result of each admitted query is returned to the client and the actual execution time is piggybacked to the prediction module in real time. This piggybacked information can further help the prediction module to improve the accuracy of its future predictions by introducing new training data. Further details can be found in [17]. We believe the consideration of profit maximization in admission control presents new challenges that we address in our solution.

5. COSMOS: SEAMLESS MOBILITY BY CLOUDDB

The mobility of today is defined by the multitude of apps, which while working in isolation, can

[2]SmartSLA stands for "Re<u>s</u>ource <u>M</u>anagement for Resource-Sh<u>ar</u>ing Clien<u>t</u>s based on <u>S</u>ervice <u>L</u>evel <u>A</u>greements".

[3]ActiveSLA stands for: <u>A</u>dmission <u>C</u>ontrol for Profi<u>t</u> <u>I</u>mpro<u>v</u>ing und<u>e</u>r <u>S</u>ervice <u>L</u>evel <u>A</u>greements.

achieve a variety of tasks for the mobile user. The mobility of tomorrow is envisioned as one where mobile apps work together by sharing information to create a seamless mobile experience, where the focus is the mobility of the user but not the device.

In the COSMOS (stands for *Clouddb fOr Seamless MObile Services*) project, we are developing a multitenant, SLA-aware, cloud-based PaaS on top of CloudDB to provide the necessary support for *seamless mobility* [13]. The core component of COSMOS is the Sharing MIddLEware (SMILE), which provides the infrastructure for mobile apps residing on COSMOS to share data actively with one another. SMILE allows for management of SLAs on the shared data, which means that some serious technical challenges will have to be overcome in order to guarantee the desired level of access on the shared data to all those who access it. The key challenge is in ensuring that SLA guarantees are provided in the face of multiple users with diverse workloads and SLA requirements, while providing performance guarantees for the data owners in sharing data with others using performance isolation policies.

The ultimate goal of mobility is to ensure that the experience of a mobile user is a rich one. This means that individual apps (i.e., mobile services) should interact with the mobile user as if they exist in an ecosystem whose collective objective is to ensure that the mobile user can interact with the digital world in a *seamless* fashion. Mobile users frequently change their context as they navigate in their fast-paced daily lives. The desire is to ensure that mobile users stay connected to the digital world through mobile services as they move forth from one context to another – regardless of the kind of mobile device and sometimes even without a mobile device, which is largely irrelevant here. In this environment, the main constraints of mobility are the limited time, patience, and attentive span of the mobile user who is *on the go*. Although there have been recent efforts to significantly advance the capabilities of mobile devices to improve the user experience, more substantial improvements in user experience can be achieved if individual apps are cognizant to the limitations of the mobile user. In some sense, we want to move beyond traditional arguments that solely attribute the challenges of mobility to the limitations of the mobile device, but instead focus on how apps can provide a much better user experience. Apps that constantly adapt to the current context of the mobile users are said to exhibit "*seamless mobility.*"

The seamless mobility can be achieved by apps working together to help the mobile user achieve tasks on the go. In that case the apps could greatly benefit from sharing information with one another. The mobility as we envision is far from what currently exists. Even though there is a rich variety of apps on mobile devices, they generally do not talk to one another or they try to do it an ad-hoc manner let alone share information to create a seamless mobile experience. The problem with most of todays connections among the mobile apps is that it is unidirectional and not scalable in the sense that every app needs to implement the API individually. In general, APIs are expensive to create and maintain, not to mention that they may not be expressive enough for most sharing needs between apps. Moreover, as apps are often hosted in the same cloud infrastructure or on different cloud infrastructures that can communicate in standard ways, there are other less expensive ways of enabling communications between apps hosted in the cloud.

COSMOS PaaS System: Mobile apps can be viewed as front ends driven from remote services, which are typically hosted on the cloud. For example, a weather app on a mobile device is essentially a front end that queries a data store on the cloud infrastructure for the weather conditions at a certain zip code. PaaS provides hosting, processing and querying of data for any mobile app that wishes to use its services. A PaaS is the most appropriate place to build a service for sharing as it typically hosts data from several other apps. Sharing between the apps can be provided as a service with little or no overhead to the apps that use it. Note that one critical aspect of sharing in this context that we believe would make sharing successful is the consistent nature in which mobile apps can refer to a mobile user using a small set of *key identifiers* (e.g., phone number, device id, simcard id). Sharing, in our context, adds value to all the parties involved by providing access to richer information on the mobile user.

COSMOS would be a PaaS offering for mobile apps built on NEC CloudDB [10], with the goal of supporting seamless mobility by enabling wide scale sharing between apps. We consider a RDBMS model of data storage, such that access to the hosted data is typically using SQL. In the PaaS setting, mobile apps that use the PaaS to host their databases are referred to as *tenants*. Usually a *PaaS provider* hosts several tenants in the same cloud infrastructure. In other words, the PaaS provider usually resorts to *multitenancy* for good resource usage and spreading of the operation cost among several tenants. To ensure that all the tenants get an acceptable level of service, in spite of sharing the infrastructure with several others, tenants negotiate SLA with the PaaS provider. SLA is a contract that describes the level of service a tenant requires on the data hosted with the PaaS. For example, an SLA could specify that the tenant would pay 10 cents for queries responded within 300ms, while the tenant would penalize the PaaS $1 if the execution time for the query exceeds 300ms. The PaaS provider, whose objective is to maximize profits, ensures that sufficient resources are available so that tenants do not miss their SLA deadlines too often as that results in a loss of revenue.

SMILE: Data Sharing in the Cloud: The key component of COSMOS is a middleware for sharing data. The Sharing MIddLEware, known as SMILE, enables sharing between a tenant t, who is the *owner* of the data and another tenant, referred to as a *consumer*, who wants access to t's data. Consider a scenario of two apps, say App-A and App-B, hosted on COSMOS that agree to share data. In particular, let us only consider the case of App-A, who is the data owner, agreeing to share data with App-B who is the consumer. For instance, App-A could be a calendar service, while App-B could be an airline ticket booking service that wants to query calendar appointments to determine if the user is traveling in the near future. If App-A wants to share some of its data with App-B, in a traditional scenario, App-A would create an API and share the details of the API with App-B. Now, App-B would use the API to access App-A's data. The advantage of this model is that App-A is *loosely coupled* with App-B in the sense that App-A is free to change its data layout without really affecting App-B as long as the API is suitably updated. However, App-A must setup the necessary infrastructure to create the API as well as keep updating it whenever its data layout changes. In our context, an API is an inefficient way to access App-A's data, especially if both App-A and App-B reside in the COSMOS system.

An extreme solution would be if App-A allows App-B to access its data directly. As App-A and App-B are both tenants in COSMOS, this can be trivially achieved. The drawback of this arrangement is that it leads to a tight coupling between App-A and App-B in the sense that if App-A changes its data layout, it has to coordinate with App-B. Other issues with sharing that are specific to a PaaS system like COSMOS are as follows. Imagine the scenario that App-B is a *hard-hitter* (i.e., issues queries at a high rate) of App-A's data, which would lead App-A to frequently miss SLA deadlines on its own data. Moreover, if App-A extensively shares its data with several other consumers, the access on the shared data may be exceedingly poor for all the parties involved without the PaaS investing additional resources to ensure that everyone gets reasonable access. Sharing may require substantial investment of resources from the COSMOS provider, where the COSMOS provider has setup a *materialized* shared space for App-B, which ensures that App-B's access on the shared space will not significantly affect App-A's queries. As materialized shared space takes up storage and is expensive to maintain so this solution, while attractive, must be used intelligently.

6. REFERENCES

[1] Apache HBase. `http://hbase.apache.org/`.

[2] CloudDB:A Data Store for All Sizes in the Cloud. `http://www.nec-labs.com/dm/CloudDBweb.pdf`.

[3] Google App Engine. `http://code.google.com/appengine/`.

[4] Project Voldemort. `http://project-voldemort.com/`.

[5] P. A. Bernstein, D. W. Shipman, and J. B. Rothnie, Jr. Concurrency control in a system for distributed databases (SDD-1). *ACM Trans. Database Syst.*, 5(1):18–51, 1980.

[6] Y. Chi, H. J. Moon, and H. Hacıgümüş. iCBS: Incremental cost-based scheduling under piecewise linear slas. *PVLDB*, 4(9), 2011.

[7] G. Eadon, E. I. Chong, S. Shankar, A. Raghavan, J. Srinivasan, and S. Das. Supporting table partitioning by reference in oracle. In *SIGMOD '08*, pages 1111–1122, 2008.

[8] S. Elnikety, E. Nahum, J. Tracey, and W. Zwaenepoel. A method for transparent admission control and request scheduling in e-commerce web sites. In *Proc. of WWW*, 2004.

[9] D. Gmach, S. Krompass, A. Scholz, M. Wimmer, and A. Kemper. Adaptive quality of service management for enterprise services. *ACM Trans. Web*, 2008.

[10] H. Hacıgümüş, J. Tatemura, W. Hsiung, H. J. Moon, O. Po, A. Sawires, Y. Chi, and H. Jafarpour. CloudDB: One size fits all revived. In *IEEE World Congress on Services (SERVICES)*, 2010.

[11] J. M. Peha and F. A. Tobagi. A cost-based scheduling algorithm to support integrated services. In *INFOCOM*, 1991.

[12] J. M. Peha and F. A. Tobagi. Cost-based scheduling and dropping algorithms to support integrated services. *IEEE Transactions on Communications*, 44(2):192–202, 1996.

[13] J. Sankaranarayanan, H. Hacıgümüş, and J. Tatemura. COSMOS: A Platform for Seamless Mobile Services in the Cloud. In *IEEE MDM*, 2011.

[14] J. Tatemura and H. Hacıgümüş. Microsharding: A declarative approach to support elastic OLTP workloads. In *The 5th Workshop on Large Scale Distributed Systems and Middleware (LADIS)*, 2011.

[15] S. Tozer, T. Brecht, and A. Aboulnaga. Q-cop: Avoiding bad query mixes to minimize client timeouts under heavy loads. In *Proc. of ICDE*, 2010.

[16] P. Xiong, Y. Chi, S. Zhu, H. J. Moon, C. Pu, and H. Hacıgümüş. Intelligent Management of Virtualized Resources for Database Systems in Cloud Environment. In *ICDE*, 2011.

[17] P. Xiong, Y. Chi, S. Zhu, J. Tatemura, C. Pu, and H. Hacıgümüş. ActiveSLA: A profit-oriented admission control framework for database-as-a-service providers. In *SoCC*, 2011.