2012 IEEE 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops

Scalable Architectures for SaaS

Wei-Tek Tsai*† and Yu Huang*
*School of Computing, Informatics, and Decision Systems Engineering
Arizona State University
Tempe, AZ, USA
Email: {wtsai, yu.huang.1}@asu.edu

Xiaoying Bai†
†Dept. of Computer Science and Technology, INLIST,
Tsinghua University
Beijing, China
Email: baixy@tsinghua.edu.cn

Jerry Gao‡
Dept. of Computer Engineering
College of Engineering
San Jose State University
San Jose, CA, USA
Email: jerrygao@email.sjsu.edu

***Abstract*—An important issue faced by Software-as-a-Service (SaaS) application is scalability. Each SaaS application is typically shared by multiple (tens or hundreds) organizations (tenants). Each tenant may have hundreds or thousands of users. Thus, the number of concurrent accesses is high. Handling a large number of user requests effectively is critical for SaaS applications. Various aspects of SaaS can have a significant impact on its scalability, including levels of scalability mechanisms, automated migration, tenant awareness, workload support, fault-tolerance and recovery, software architecture and database access. This paper identifies scalability factors and discusses their impacts on the scalability of SaaS applications. Existing approaches for addressing the scalibility of SaaS applications are also analyzed, and this paper suggests some alternatives to improve SaaS scalability based on the factors identified.**

I. Introduction

Software-as-a-Service (SaaS) is a new software delivery model where users access the software via a Web browser. The software together with the associated data are hosted in a cloud. Scalability refers to the ability of a system to handle growing amount of work with stable performance with proportional new resources. For example, a scalable online shopping application needs to provide acceptable response time under an increased workload, such as during a holiday season, and furthermore adding resources incrementally to support growing workloads should be easy.

In MTA (Multi-Tenancy Architecture) SaaS, only a single code base is maintained to support all the users of the SaaS, while users feel like having their custom-made software. However, such architecture also posts challenges to its scalability, because each SaaS application instance is shared by multiple (tens or hundreds) organizations (tenants). Each tenant may have hundreds or thousands of users. The number of concurrent accesses from the users is high. Handling large amount of tenants / users effectively is critical for SaaS applications.

There are generally two solutions to scale a software system: *scale-up* and *scale-out*. Scale-up (aka. vertical scaling) means running the application on a machine with a better configuration, including more computing resource, more memory, higher disk bandwidth and larger disk space. Scale-out (aka. horizontal scaling) means running the application distributed on multiple machines with similar configurations. Because the resources of a single machine cannot be increased infinitely, and increase of the cost is not proportional to the increase of the resources, scale-up is not feasible in an Internet scale. Scale-out is usually adopted in a cloud.

Scalibility in parallel and distributed computing has been studied in the past. Specifically, scalability in PaaS has also been addressed. However, SaaS scalability has not been adequately studied. This paper identifies key factors that affect SaaS scalability, including levels of scalability mechanisms, automated migration, tenant awareness, workload support, recovery and fault-tolerance, software architecture, and database access.

This paper is organized as follows: Section II introduces SaaS and reviews scalability techniques; Section III presents factors that affect SaaS scalability and the interactions between these factors; Section IV reviews the scalability of the existing solutions with respect to the identified factors; Section V concludes this paper.

II. Related Works

A. Software-as-a-Service

SaaS systems often run on top of a PaaS (Platform as a Service). It leverages the datastore API provided by the PaaS systems for data storage and cache frequently used data for efficient data access. SaaS is essentially software plus datastore, as shown in Figure 1. However, different PaaS systems provide different support for SaaS. The design, performance, and scalability of a SaaS system running on top of a PaaS will be significantly affected by the underlying PaaS. If the PaaS is a closed system, i.e., if it does not open up control options for SaaS designers, and essentially the performance and scalability will be mostly controlled by the PaaS. However, if the PaaS system opens up control options, the SaaS designers can design different scheduling, customization, MTA, and scalability approaches. In this case, the PaaS is more like IaaS (Infrastructure as a Service), and allow SaaS designers to custom made the SaaS architecture. For example, GAE (Google App Engine) provides transparent fault-tolerance and scalability, without the interference of the applications, while EC2 provides a generic infrastructure where SaaS developers need to handle various aspects of the system including scalability. Furthermore, some SaaS systems run on top of a traditional or modified relational database system.

978-0-7695-4669-8/12 $26.00 © 2012 IEEE
DOI 10.1109/ISORCW.2012.44

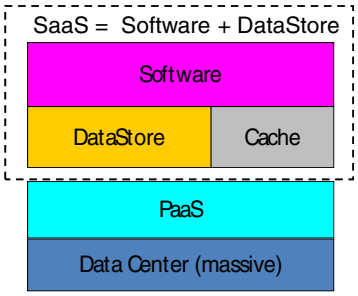

Fig. 1. SaaS Architecture

Note that as most PaaS systems were designed for scalable Internet applications, and they often have different system features from traditional OS and databases systems. For example, each write will be carried out three times (such as in GAE) for fault-tolerant computing. Writers have higher priority than readers (such as in Dynamo). NoSQL databases are used as well as relational databases.

A SaaS system also has its own unique features such as customization via metadata, MTA, and scalability. It can have its own fault-tolerant mechanisms, in addition to the fault-tolerant mechanisms offered by the underlying PaaS, such as having redundant information in various metadata tables and data tables [1]. The system design for MTA is also a source for design optimization and scalability. Some MTA is designed for easy development and scalability, but with limited customization, e.g., Corenttech.com. While some MTA designs offer full customization and scalability. Some database schema design will favor OLAP, but some for OLTP.

In summary, different SaaS designs on different PaaS systems will result in distinct customization, MTA, and scalability strategies. For those more closed PaaS systems, SaaS systems have less freedom and control in execution. It is easy to develop SaaS applications but many SaaS design options will be fixed by the underlying PaaS, and thus making the SaaS operations less than optimal as the same PaaS was designed to support all different SaaS systems.

A recent approach to develop SaaS is to adopt the OMG's approach by first developing a platform-independent model, and then map the model into concrete platforms or PaaS. EasySaaS [2], takes this approach as shown in Figure 2 and 3, and it has two major components, one for SaaS application development, one for execution on a PaaS. The SaaS application development is divided into two stages, the first stage develops a platform-independent model that consists of GUIs, workflows, services, and data, and a composed model can be analyzed and simulated before it will be deployed to a specific platform. Platform-independent design and optimization such as model consistency and DB design can be done at the model level. The second stage translates the model into executable code to run on top of specific platforms such as GAE, EC2 or Azure. In the second stage, platform-dependent design and optimization such as inserting monitoring points and automated provisoning can be performed.

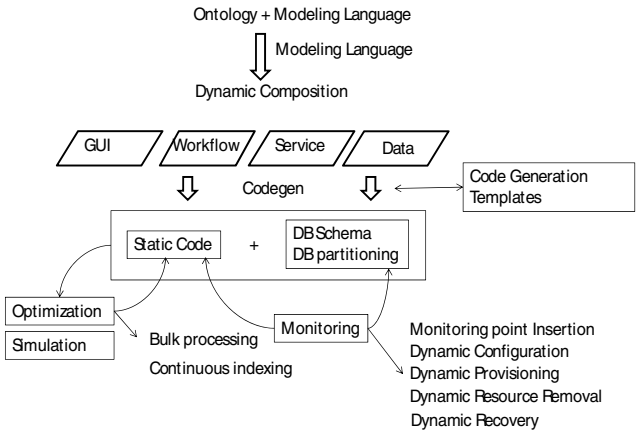

Fig. 2. EasySaaS Architecture

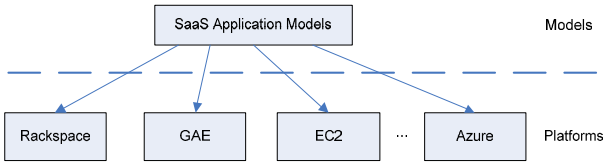

Fig. 3. Code Generation Architecture

B. Scalability Techniques

Scalability has been studied in parallel computing, distributed computing, databases, and service-oriented architecture (SOA).

Scalable design principles for application servers include divide-and-conquer, asynchrony, encapsulation, concurrency, and parsimony [3]. Divide-and-conquer means the system tasks should be divided into smaller tasks with single functions, and a system should be well partitioned into components; Asynchrony means work can be done on a resource-available basis, and this may imply distributed and/or self scheduling, background processing; encapsulation means system components and architecture such as layers are well encapsulated; concurrency means tasks can be done in parallel taking advantages of the distributed nature of hardware and software; parsimony means that the design considers the cost efficiently.

Other common techniques include system partitioning, service-based layered architecture, pooling and multiplexing, queuing and background processes, data synchronization, distributed session tracking, and intelligent load distribution. For example, SEDA (Staged Event-Driven Architecture) [4] shows how these principles can be used to develop a scalable architecture. The original complex event processing is divided into different stages (divide and conquer) and encapsulated with queues for interacting each stage, and each stage can be executed asynchronously and can be tracked and monitored.

According to Ebay [5], the following are key scalability principles: 1) partition by function or grouping those functions

that are related and split those functions that are related to each other, and this can be applied to code, procedure, architecture and databases; 2) split horizontally and this can be applied to both software and databases. When applied to software, it means that code should be *stateless* so that they do not interfere with each other and they can be developed in parallel. When applied to databases, it means data are partitioned so that processing can be done in parallel, and data can be repartitioned easily; 3) avoid distributed transactions; 4) decoupled functions asynchronously so that each part can be scaled independently ; 4) move processing to asynchronous flow; 6) virtualize at all levels; 7) cache appropriately.

While in [6], scalability is achieved by decreasing processing time by collocation, caching, pooling, parallelization, partitioning, and remoting, and concurrency. In [7], IBM proposed some scalability techniques,such as using appliance servers, segmenting the workload, batch requests, aggregate user data, manage connections, and cache. Furthermore, self-managing servers, pervasive computing devices are mentioned as new scalability techniques.

To evaluate the scalability of a system, many scalability metrics have been proposed in distributed computing and high performance computing environment [8]. However, most of these metrics do not capture the new factors brought in by cloud computing, such as cost and resource utilization. The scalability metric proposed in [8] takes resource usage into consideration for measuring the scalability of applications run in a cloud environment using statistical techniques because resources are shared and thus individual run may have different performance.

III. SaaS Scalability Factors and their Interactions

A. Scalability Factors in SaaS

Levels of Scalability Mechanisms. Most SaaS applications adopt a tiered architecture. A typical SaaS application has three tiers: the storage, the application and the presentation. Scalability mechanisms can be applied to a single tier, or across multiple tiers. Scaling within single tier means that the scaling of this tier is independent from the scaling of other tiers. For example, the application tier may run on five machines, while the storage tier runs on three machines. Scaling across tiers means that the whole SaaS application stack is scaled together. Due to the high cost of duplicating the whole application stack, cross-tier scaling is rarely adopted. However, a hybrid approach is typically used, which duplicates the whole SaaS application stack and leverages single tier scaling within the application stack.

Scalability mechanisms at different tiers may use different techniques as each tier has its own constraints and objective. For example, a SaaS can be scaled on both the tenant tier and the application server tier. At the tenant tier, the objective is to divide tenants to different groups and direct tenant requests to specific groups. While at the application layer, it objective is to balance the workloads of application servers, and one common technique is to make these server stateless.

Automated Migration. There are cases where tenant data must be migrated for better performance, and thus automated migration of data is critical for scalability. Several issues need to be considered. First, it is necessary to determine if the migration will be done online or offline. Online migration, i.e., the migration will take place while the application will be still in operation, is more difficult than offline migration where the migration takes place when the SaaS shuts down its services for maintenance. Though online migration can provide 24/7 services for SaaS customers, the added complexity of online data migration is high. Second, it is necessary to determine the data to be moved for scalability. One strategy is to move the minimum amount of data (to minimize the bandwidth demand) and to the closest node (to minimize latency delay), but other approaches are also possible. For example, one strategy is to move the indices with the tenant data, but an opposite strategy is to move the tenant data only. The second option requires reconstruction of indices after data migration. Third, the design of the storage format should have tenant data migration in consideration. For example, all data belonging to the same tenant should be grouped together for easy migration.

Tenant Awareness. The design of the storage tier needs to be tenant-aware to support customization, maintain tenant isolation, and facilitate data migration. Tenant awareness can be supported by two types of traceability: *forward* and *backward* traceability. Forward traceability means that given a data item to be queried or inserted, the system knows which tenant owns this data item and where to query/insert this data item by tracing forward from the tenant table. Backward traceability means given a stored data item, the system can trace back from the data item to the tenant table. If the system supports only forward traceability, the system can perform data migration by tracing the tenant table. However, if the system also provides backward traceability, data may be moved autonomously and asynchronously upon detection of the need for data migration. Proving both types of traceability for each data item is expensive, but providing both for a group of data items, especially those belonging to the same tenant will be useful for autonomous data migration.

Workload Support. Different workloads require different scalability mechanisms. For an OLAP workload, a high portion of the requests are reading data from the system. In this case, the system should be able to scale in case of high volume of read operations. For the an OLTP workload, write operations are dominant. In this case, the system should be able to distribute the write operations to avoid bottleneck at a single node. For a mixed workload where portion of read and write operations are close, the architecture needs to be designed to ensure there is no bias towards either type of operations because the bias may result in poor scalability.

Recovery and Fault-tolerance. Recovery and fault-tolerance mechanisms also affect the system scalability. First, the system should be able to detect the failures of nodes. When a node fails, the system should automatically scale down without a significant performance downgrade. When the failed node comes back to the system, it should automatically scale

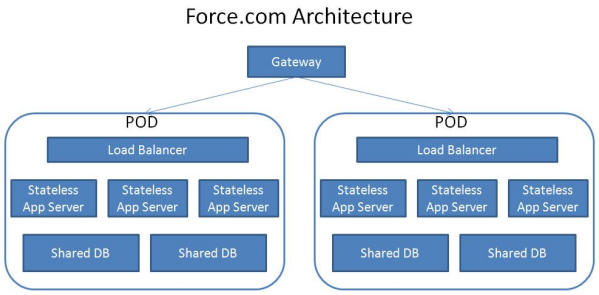

Fig. 4. The Force.com Multi-tenant Architecture

up and recover to previous working status. This process needs to be autonomous without intervention of the engineers and the scalability should be incremental when the number of nodes changes in the system.

Software Architecture. To scale up, it is necessary to avoid coupling in the architecture so that each part of the application can be independently scaled, without affecting other parts of the software system. SOA can be leveraged to decouple such large software system to provide excellent scalability. Many scalable systems adopts SOA such as DynamoDB [9].

Database Access. Accessing a database is often time-consuming, and thus is likely to be the bottleneck of a SaaS application. Access to a database can be either direct or indirect. Direct access to a database allows applications to connect directly to the database, and perform necessary operations. Indirect access to database goes through APIs exposed by certain database services wrapped on the underlying database. Indirect access allows the software and database to have its own scalability mechanism.

B. Relationships between Factors

These factors are inter-related and may affect each other. Specifically:

Levels of scalability mechanisms and tenant awareness: Each level of scalability mechanisms allow a SaaS to deal with the scalability one level at a time, but for MTA SaaS, the most critical issue is tenant identification and isolation. Thus, each level may need to deal with the tenants.

Levels of scalability mechanisms and automated migration: If the system has two or more levels of scalability mechanisms, automated migration needs to deal with each level. For example for a system with two levels of scalability, data migration can happen at each level, and sometimes at both levels if significant data need to be moved with new data partitioning and/or clustering.

Automated migration and tenant awareness: If the system has forward traceability only, automated migration must be initiated from the controller to make a coordinated plan for execution. If the system also supports backward traceability, migration can be executed in an asynchronous manner.

Workload support and database access: Different workloads need different database access patterns for optimal performance. A SaaS system can choose the traditional row-oriented storage, column-oriented storage, or a hybrid approach [1] for different tenants.

Software architecture and database access: One important system partitioning approach is SOA, and it helps to decouple components within a system. Decoupling database access layer with the clients that access the database directly makes it possible for the two parts to be scaled independently.

IV. Existing Scalable Architectures

A. Salesforce.com

Levels of Scalability Mechanisms. The Force.com platform architecture is as shown in Figure 4 and it has two levels of scalability mechanisms. At the gateway level, tenants are distributed to different PODs in the system. A tenant is designated to a specific POD and all the requests from this tenant are directed to the same POD afterwards. The second level uses stateless server to serve user requests. Consecutive requests from the same user can be directed to different servers in a POD, according to the workloads of these servers.

Automated Migration. Tenant data in a POD are stored in a heap data table, the heap table needs to be scanned to read and remove the records belonging to a specific tenant. Scanning and delete data from the heap table is expensive because the number of tuples can be huge.

Tenant Awareness. Each record of the tenant data is uniquely identified by the tenantID field in the record. Each tenant request is associated with a tenantID so that the system knows where to store the tenant data.

Workload Support. For the OLTP workload, the platform provides a restricted version of transactions. A single Apex transaction can make a maximum of 10 callouts to an HTTP request or an API call. For the OLAP workload, the index (pivot) table design used in the Force.com platform provides an efficient access for data analysis.

Recovery and Fault-tolerance. As the Force.com uses Oracle DB as the underlying storage engine, recovery and fault-tolerance may be handled by running a distributed version of the Oracle DB. However, if a complete POD fails, tenants applications allocated on this POD may be down until the whole POD is recovered from the failure.

B. Yahoo! PNUTS

Yahoo! proposed PNUTS [10] - Yahoo!'s hosted data serving platform, and it is part of its cloud platform Sherpa [11]. PNUTS focuses on the scalability of the storage tier.

Levels of Scalability Mechanisms. The PNUTS system has one level of scalability and it is divided into multiple "regions". Each region contains a set of storage units that stores data partitions called "tablets". Duplicate copies of data records are stored in different regions, with one of them being the master record. The mastership of the record is determined adaptively according to the read/write workload of the record. However, with this design, it is possible that the workload is not balanced between regions if the users are not evenly distributed geographically.

Automated Migration. PNUTS is designed with geographical locality in mind. Migration of tenants can be achieved by first synchronizing the records and then changing the mastership of the tenant data records.

Tenant Awareness. PNUTS is designed to be a general data serving platform. Tenant information can be incorporated into a SaaS to run on top of PNUTS during the SaaS development.

Workload Support. PNUTS assumes serializable transactions are normally impractical to achieve and unnecessary, while eventual consistency may be too weak and hence inadequate for Web applications as described in [10]. Thus, it provides a consistency model that is configurable. Application developers can choose the consistency level they need according to the applications. This approach can be good for handling different types of workloads, but also increases the development complexity.

Recovery and Fault-tolerance. When a region fails, user requests can be directed to other regions that contain the user data, and the mastership can be adaptively changed. Recovery is done by copying latest tablets from another replica.

C. DaaS - Database as a Service

One design is to treat a multi-tenancy database as a service like a SaaS [12], i.e., a third party hosts a database as a service (DaaS) and provides its customer the ability to create, store and access their data seamlessly. Two optimizations are proposed to improve scalability: Bitmap Interpreted Tuple (BIT) and Multi-Separated Index (MSI). It uses BIT to reduce the NULLs stored due to sparse table and uses MSI more efficient indexing to facilitate data access.

Tenant Awareness. Unique identifiers are used to represent the ownership of the data for the tenants.

Workload Support. The MSI approach proposed in this paper is similar to the pivot tables used in Force.com. The difference is that the pivot tables store another copy of data for each attribute and builds a separate index for each tenant. Also, due to the BIT optimization, though less NULL values are stored in the database and some spaces are saved, the reading of the data record and specific columns may be expensive.

D. Multi Dimensional Clustering (MDC)

This is a data layout scheme [13]–[15] that allows a relational table to be clustered on one or more orthogonal clustering attributes of a table. It also provides for a continuous and automatic maintenance of the clustering, and thus reduces the need to reorganize the table to regain clustering. Efficient optimization techniques like MDC can be used to improve the scalability of the system.

Workload Support. System performance can be greatly improved using well-designed indexing techniques. MDC is designed to for online analytic processing and data warehousing applications, which process a table or tables in a database using a multi-dimensional access paradigm [13]. For OLTP workload, it also presents promising performance [16].

E. Amazon DynamoDB

DynamoDB is a new cloud computing service added in the Amazon's Web Services family. It is a "fast, reliable and cost-effective NoSQL database service designed for internet scale applications" [17]. It is fully managed and can be scaled up or down on demand with automated provisioning and database management. According to [17], DynamoDB "taking the best ideas of Dynamo and SimpleDB".

Levels of Scalability Mechanisms. All nodes in the Amazon Dynamo [18] system are equally important, that means it has a one level scalability architecture. All the nodes form a ring, that represents all the possible keys produced by the hashing function. Each node is responsible for a portion of the key ranges.

Automated Migration. Tenant migration may not be easily done in Dynamo because the data is assigned to the nodes in the system by the hashing function.

Tenant Awareness. Similar to PNUTS, Dynamo is a general purpose data serving platform, that provides a key/value data model. Tenant identifier might be associated with each object to uniquely identify the ownership. However, to supports queries that need to retrieve all data objects related a tenant, the hash function might need to carefully designed so that tenant data can be efficiently retrieved.

Workload Support. The Dynamo system is mainly designed and used to support its e-commerce application. Thus, it has good support for the OLTP workload and transactions. However, for the OLAP workload, the Dynamo system might not be sufficient because it provides a key/value data model only.

Recovery and Fault-tolerance. Recovery and fault-tolerance is efficient in Dynamo. It also provides incremental scalability by leveraging consistent hashing. Adding new nodes or node failures will affect a few neighbors of the failed nodes on the ring only, and the workload of the failed node is about evenly distributed to other nodes.

F. BigTable Family

BigTable [19] is a distributed storage system, and several similar datastore systems have been developed including HBase [20], an open-source implementation of BigTable.

Levels of Scalability Mechanisms. There are a single master and many tablet servers in the system. The tablet servers are divided into multiple regions. This architecture is similar to that of PNUTS.

Automated Migration. The chunk size in the underlying chunk servers is 64MB by default. Migration of tenant data will be easy and efficient if the tenant data is clustered in these chunks.

Tenant Awareness. Tenant support is not included in the tablet schema. A unique identifier field can be added for the support of tenants.

Workload Support. BigTable targets more for OLTP applications. It also supports the OLAP workloads, as all the Google analytics data are stored in BigTable. However, the OLAP support can be optimized in BigTable [21].

Database Access. Accesses to the database can be conducted through the Google datastore API.

G. Discussion

Existing solutions might be improved for better scalability. For example for Salesforce.com, one way to improve tenant migration is to partition tenant data (sharding) in the PODs, specifically on the heap data table [22] so that when a migration needs to be performed, a group of tenants in the same partition within the POD can be moved together. Database sharding is not new but it presents new optimization opportunities. For example, because all the columns of the heap data table are of type string, that are used to store heterogeneous data according to tenant customization, the sharding scheme can be optimized. Instead of merely partitioning the data table by tenant's unique identifier, the partitioning should also consider the homogeneity of data type in the columns. Special indexing schemes might also be developed for more efficient access for table holding heterogeneous data using single data type.

V. Conclusion

This paper focuses on the scalability of a SaaS application. It identifies the factors that have significant impacts on system scalability, including levels of scalability mechanisms, automated migration, tenant awareness, workload support, recovery and fault-tolerance, software architecture and database access. For a complex software system like SaaS, multiple levels of scalability mechanisms with different scalability mechanisms at each level can scale well however with increased system complexity. Automated migration of tenant data should be taken into consideration, especially in the data access level. Tenant awareness is critical for SaaS, as multi-tenant support is one of the keys in SaaS, and also because it affects all other modules in SaaS. Decoupling system functions and avoid direct access to database facilitate system scalability so that each can be scaled independently, and with each stateless server can be developed in parallel by different development teams without interfering each other. A SaaS system needs to know its workloads so that it can scale well for its specific workload, and the scalability mechanisms may interact with the system fault-tolerant mechanisms, for example, with multiple copies available in different nodes, system recovery is easier but each write may create additional workload to make copies at different nodes.

Acknowledgment

This project is sponsored U.S. National Science Foundation project DUE 0942453 and the European Regional Development Fund and the Government of Romania under the grant no. 181 of 18.06.2010. This project is also supported by National Science Foundation China (No. 61073003), National Basic Research Program of China (No. 2011CB302505), and the Open Fund of the State Key Laboratory of Software Development Environment (No. SKLSDE-2009KF-2-0X). It is also supported by Fujitsu Laboratory.

References

[1] W. Tsai, Y. Huang, Q. Shao, and X. Bai, "Data partitioning and redundancy management for robust multi-tenancy SaaS," *International Journal of Software and Informatics*, vol. 4, no. 4, pp. 437–471, 2010.

[2] W. Tsai, Y. Huang, and Q. Shao, "EasySaaS : A SaaS development framework," in *Proceedings of SOCA*. IEEE, 2011.

[3] C. Roe and S. Gonik. (2002) Server-side design principles for scalable internet systems. [Online]. Available: http://www.cs.princeton.edu/courses/archive/spr11/cos448/web/docs/week8_reading2.pdf

[4] M. Welsh. (2001) The staged event-driven architecture for highly-concurrent server applications. http://www.eecs.harvard.edu/ mdw/proj/seda/. [Online]. Available: www.eecs.harvard.edu/~mdw/papers/quals-seda.pdf

[5] R. Shoup. (2008) Scalability best practices: Lessons from ebay. [Online]. Available: http://www.infoq.com/articles/ebay-scalability-best-practices

[6] R. Bradford. (2011) Successful scalability principles. [Online]. Available: http://effectivemysql.com/downloads/SuccessfulScalabilityPrinciples-Part1-2011-05.pdf

[7] W. Chiu. (2001) Design for scalability - an update. [Online]. Available: http://www.ibm.com/developerworks/websphere/library/techarticles/hipods/scalability.html

[8] W. Tsai, Y. Huang, and Q. Shao, "Testing the scalability of SaaS applications," in *Proceedings of SOCA*. IEEE, 2011.

[9] highscalability.com. (2007) Amazon architecture. [Online]. Available: http://highscalability.com/amazon-architecture

[10] B. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H. Jacobsen, N. Puz, D. Weaver, and R. Yerneni, "PNUTS: Yahoo!'s hosted data serving platform," *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1277–1288, 2008.

[11] R. Rmakrishnan, "Sherpa: Cloud computing of the third kind," in *Data-Intensive Computing Symposium*, 2008.

[12] M. Hui, D. Jiang, G. Li, and Y. Zhou, "Supporting database applications as a service," in *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*. IEEE, 2009, pp. 832–843.

[13] IBM. (2003) Multi dimensional clustering (MDC) in DB2. [Online]. Available: MultiDimensionalClustering(MDC)inDB2

[14] B. Bhattacharjee, S. Padmanabhan, T. Malkemus, T. Lai, L. Cranston, and M. Huras, "Efficient query processing for multi-dimensionally clustered tables in DB2," in *Proceedings of the 29th international conference on Very large data bases-Volume 29*. VLDB Endowment, 2003, pp. 963–974.

[15] S. Padmanabhan, B. Bhattacharjee, T. Malkemus, L. Cranston, and M. Huras, "Multi-dimensional clustering: a new data layout scheme in DB2," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '03. New York, NY, USA: ACM, 2003, pp. 637–641. [Online]. Available: http://doi.acm.org/10.1145/872757.872835

[16] Z. Wang, C. Guo, B. Gao, W. Sun, Z. Zhang, and W. An, "A study and performance evaluation of the multi-tenant data tier design patterns for service oriented computing," in *e-Business Engineering, 2008. ICEBE'08. IEEE International Conference on*. IEEE, 2008, pp. 94–101.

[17] W. Vogels. (2012, January) Amazon DynamoDB. Amazon Inc. [Online]. Available: http://www.allthingsdistributed.com/2012/01/amazon-dynamodb.html

[18] D. Hastorun, M. Jampani, G. Kakulapati, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazons highly available key-value store," in *In Proc. SOSP*. Citeseer, 2007.

[19] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber, "BigTable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, p. 4, 2008.

[20] Apache. HBase. [Online]. Available: http://hbase.apache.org/

[21] highscalability.com. (2008) How I learned to stop worrying and love using a lot of disk space to scale. [Online]. Available: http://highscalability.com/how-i-learned-stop-worrying-and-love-using-lot-disk-space-scale

[22] Force.com. The Force.com multitenant architecture: Understanding the design of Salesforce.coms internet application development platform.

[23] W. Tsai, Y. Huang, and X. Bai, "Grapevine model for template recommendation and generation in SaaS applications," in *Proceedings of The Third Asia-Pacific Symposium on Internetware*, 2011.