A SLA Graph Model for Data Services

Katerina Stamou, Verena Kantere, Jean-Henry Morin
Institute of Services Science
University of Geneva, Switzerland

Michael Georgiou
Cyprus University of Technology
Cyprus

ABSTRACT

Cloud computing has given rise to on-demand service provisioning and massive outsourcing of IT infrastructures and applications to virtual, commoditized ones. Despite the broad Service Level Agreement (SLA) usage in scientific settings, their role in cloud markets is peripheral and misinterpreted.

The paper introduces a SLA graph data model that supports automated SLA formalization and data management through a property digraph. The data model is described as a directed graph (digraph). We elaborate on node and edge properties that indicate dependencies in the SLA data management flow. We sketch a realistic scenario of cloud data service provisioning to extract attributes that characterize the data service. The SLA graph model and data service attributes are used to demonstrate the formalization of a SLA template that is managed as a property graph. The graph structure enables the manipulation of SLA information in a modular, extensible way that considers the data flow and all inclusive data dependencies.

Categories and Subject Descriptors

H.3 [**Information Storage and Retrieval**]: Online Information Services; H.2 [**Database Management**]: Data models

Keywords

service level agreement; property graph; data management; distributed service management

1. INTRODUCTION

The cloud computing paradigm has enabled dynamic service management that is performed by cloud providers in a time- and cost-efficient manner. Cloud services are provisioned based on Service Level Agreements (SLAs), i.e. contracts between providers and customers that define agreed upon guarantees with respect to service provisioning details.

CloudDB'13, October 28, 2013, San Francisco, CA, USA.

http://dx.doi.org/10.1145/2516588.2516592 ...$15.00.

The SLA content assures the mutually agreed service levels between a provider and a customer [5]. SLAs describe obligatory service provisioning terms and encapsulate Quality of Service (QoS) characteristics. They may include a multitude of technical and business service-level objectives (SLOs) along with metrics that permit the calculation and measurement of service parameter values. Every customer needs to agree on a SLA in order to lease a new service. Traditionally, providers define SLAs, in which they guarantee explicit service-level bounds over a predefined, agreed period.

SLAs can be perceived as machine-readable documents, which are not yet standardized for the cloud computing domain. Such documents are typically semi-structured and their content varies according to service types and diverse business environments. So far, the scientific literature lacks of SLA data management approaches that enhance the automated handling of SLAs and enable their adoption by cloud markets.

Although the SLA data volume is not critical with respect to data operations, the SLA components introduce volatility and inter-dependencies that need to be addressed through a flexible data structure. This work presents a graph data model that is intended to be employed for the SLA data management of cloud services. The WSLA specification [13] is used as our reference to transform the SLA language schema into a property digraph.

The paper contributes a simple, structured way to keep and manage SLA information through a digraph that is modular, extensible and expressive with respect to data operational needs and dependencies. The SLA digraph specifics and the mapping of SLA information into nodes and edges are discussed along with the implementation details of a Python module that implements the proposed digraph model and generates SLA templates.

We sketch a realistic scenario of data service provisioning to frame the data service notion by extracting metrics and SLA parameters that comply with general data service characteristics. The description of the featured scenario summarizes attributes that are generic enough to apply in most types of provisioned data services. The scenario details are simplified to fit the paper demonstration scope. We apply the scenario characteristics to the SLA digraph module and illustrate the generated SLA template [18]. We analyze the SLA formalization as a digraph through subgraphs of the produced template, where we exercise an empirical, simple mapping process to transform the scenario information into SLA elements.

We elaborate on the immediate advantages of the graph structure for the SLA representation and data management. The graph enables the modular handling of enclosed information, where nodes and edges represent various SLA terms that can be handled separately. The proposed data model is extensible and can be adapted for any type of cloud service. The graph structure allows for clear information flow between technical and business SLA terms. The data model considers node and edge properties that are embedded in the SLA digraph definition and complete the SLA formalization.

Research questions with respect to the evaluation of the proposed SLA data model are introduced. Such questions summarize our current activities. They primarily deal with the accuracy and correctness of data queries to the proposed digraph model. Furthermore, as the proposed model is intended for the data management of SLAs over distributed repositories, challenges around scalability, efficiency and performance throughput are discussed.

The organization of the paper is the following: Section 2 introduces, the WSLA anatomy, its transformation into a digraph and the SLA digraph module implementation. Section 3 describes the data service provisioning scenario, where a data sevice provider offers deployed relational databases. The scenario description focuses on the analysis of service attributes and parameters that apply uniformly to data services. We then demonstrate and discuss two subgraphs of the generated scenario template. Section 4 provides a summarized analysis of the SLA digraph advantages and challenges. We conclude the discussion with related work.

2. SLA PROPERTY DIGRAPH

This section describes the SLA anatomy according to the Web Service Level Agreement (WSLA) specification [13] and its transformation into a property digraph. We elaborate on the characteristics of SLA components by representing them as element sets of the SLA tree. We discuss the implementation of a Python module that produces SLA digraphs, given the required data input and application criteria.

2.1 SLA anatomy

So far, the WSLA [13] and WS-Agreement [1] language specifications have been used by computer scientists to express SLAs in a machine-readable format. Both language specifications are expressed in XML and allow for full automation of the SLA document as well as for its manipulation by backend processing systems. Both specifications use a tree data structure to represent the SLA information. Tree branches illustrate separate SLA sections and tree leaves inner section terms.

According to [13], a SLA complements a service description. In contrast to other contracts for IT services, properties within a SLA have to be monitored during workload execution to audit potential service-level violations and to verify adherence to the agreed SLOs. Figure 1 illustrates a high level view of the WSLA language anatomy, according to which the core elements of a SLA are the following:

Parties: Signatory parties consist of one service provider and one service customer. Supporting parties represent third parties that operate on behalf of either or both signatories. Every SLA is associated with a single provider-customer pair to represent the one-to-one agreement relationship. Customer and provider can be affiliated with one or more third

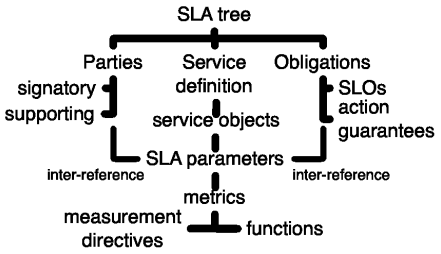

Figure 1: Web Service Level Agreement (WSLA) structure [13]

SLA$\supset$ $\{Signatory_2, Obligations_1, \{ServiceDescription_i\}\}$, where $i \in [1,n]$
$Signatory_2 \supset SupportParty_i$, where $i \in [0,n]$
$Obligations \supseteq SLO_i, ActionGuarantee_i$, where $i \in [0,n]$
$ServiceDefinition_i \supseteq ServiceObject_i$, where $i \in [1,n]$
$ServiceObject_i \supset SLAparameter_i$, where $i \in [1,n]$
$CompositeMetric_i \supset ResourceMetric_i, CompositeMetric_j, Function_j$, where $i \in [1,n]$ and $j \in [0,n]$

and i, j indicate the cardinality of element subsets and $n \in \mathbb{N}$.

Table 1: SLA language components - Set representation

parties that support service operations. Both signatories can be associated with the same third party [13].

Service definition: Service objects represent description terms and include SLA parameters that contain properties and indicate quantitative as well as qualitative metrics.

Obligations: A provider defines guarantees in the form of obligations either as service level objectives (SLOs) or as action guarantees. SLOs represent measurable targets that a provider promises to fulfill during service execution. SLO values are verified through real-time measurement. Action guarantees signify tasks that the provider, one or more supporting parties or, in some cases, the customer will take to establish the promised service levels of one or more SLA parameters. A well-defined description of service and resource parameters is prerequisite for the specification of QoS attributes in the form of either SLOs or action guarantees.

The core elements of the language are further divided into granular sub-elements. In the case of service objects, the granularity of information reaches the level of URI sources through which measureable values are monitored. In the case of service obligations, the flow of the information content involves business level objectives, as well as SLO evaluation functions and expressions that typically follow first order logic.

2.2 WSLA into digraph transformation

To represent the WSLA specification as a digraph we first transform primary language components into element sets by taking into account their cardinalities and relationships. Table 1 summarizes the identified sets. We use a subscript to denote the cardinality of elements in any SLA graph instance.

We construct a digraph that follows the WSLA guidelines with respect to primary language components. The digraph

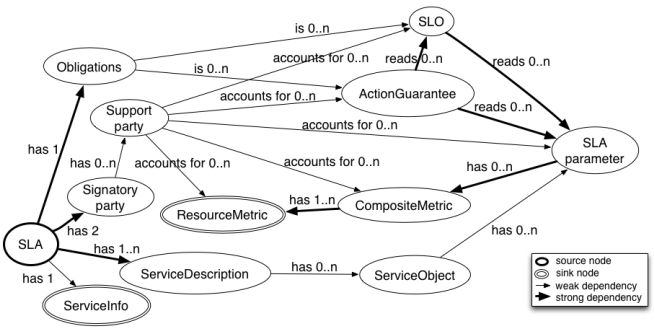

Figure 2: SLA representation as a digraph

nodes represent SLA terms and sub-terms. We add an additional node, the service-information one, to include data that identify overall service properties, which are not related with obligations of any involved party. Motivation for the inclusion of such node is derived from the WS-Agreement specification [1] that contains a similar section named "Context". The service-information node denotes SLA attributes like the location of the provider infrastructure and customer stored data, the legislation that the service provider adheres to, as well as service monitoring and configuration options for customers, if such exist.

We use directed edges to illustrate **unidirectional dependencies** between SLA terms in the digraph information flow. With the term unidirectional dependencies we indicate operational and management functions or attributes between diverse SLA components. The "SLA" represents the source node and the skeleton digraph contains two sink nodes. The digraph nodes and edges contain properties. A node or edge property is formed as a key-value pair that is embedded in the node or edge definition.

Figure 2 illustrates the SLA digraph. The primary model consists of 12 nodes that are interconnected through 19 edges. The skeleton graph can be extended to include any type of nodes or edges in any valid graph orientation, e.g. diverse service-description subgraphs with respect to application or business needs.

The graph structure provides a simple but powerful way to order semi-structured SLA data into structured context. The digraph representation highlights the logical flow between service dependencies and guarantees in the SLA content. Alternative graph representations will also yield reasonable SLA structures, given business domain constraints or application specific needs.

2.3 SLA digraph description

The skeleton digraph is implemented as a Python 2.7 module using the Networkx [8] scientific library. The SLA skeleton is defined as a DiGraph. An acyclic attribute is derived by the unidirectional connection between nodes that eliminates cycles. In the model definition, the "Obligations", "Signatory party" and "ServiceDescription" nodes are declared as subgraphs of the SLA skeleton graph.

The "Obligations" node represents a single node in the skeleton digraph. We define it as a subgraph since it may include numerous service level objectives ("SLO") and action guarantees ("ActionGuarantee") that may not apply uniformly for a given SLA instance. The "ServiceDescription" node may have many instances in a given SLA graph. Each service description instance is defined as a separate subgraph of the skeleton graph and may include numerous "ServiceObjects", "SLAparameters" and metric nodes.

Similarly, the "Signatory party" node has exactly two instances in the SLA graph. The party instances are defined as customer and provider subgraphs and may include any key-value paired information relevant to each signatory. Every signatory may have multiple support parties that are obliged in the service/SLA management.

Nodes and edges in the digraph represent the SLA content and are accompanied with properties like description, measurement functions as well as measurement type and unit. For example, edges that connect SLA parameters to composite or resource metrics, include a property on the measurement and evaluation schedules of the respective metrics. Such service details are important for SLA management since they enable the accurate execution of the service contract.

In Figure 2 a thick line denotes edges that indicate strong dependencies between nodes, while a thin line denotes weak dependencies. A **strong dependency** signifies that the content of the antecedent node is subject to frequent updates or that is necessary for the definition and measurement of the precedent node. **Weak dependencies** indicate semantic interrelations between nodes.

In the model declaration we address dependencies using weights, which are assigned to nodes according to their content criticality. The content criticality is SLA, time and application specific. In graph terms, it can be depicted in the weights of either node or edge attributes and according to the desired traversal path. The NetworkX library comes in hand with several graph algorithms (e.g. shortest path, breadth first search) that utilize the assigned weight values for graph traversals.

The SLA digraph is implemented as a Python module that enables its easy integration with backend compliant systems and diverse applications. Our intention is to utilize the module as the backbone of an SLA management service that generates, stores and handles SLAs in a graph representation. At the time of this writing, the NetworkX library lacks of a backend persistence layer to allow for graph data management operations. The SLA digraph module is currently rebuilt to communicate with the Titan [21] graph database.

3. PROVISIONING SCENARIO

We sketch a data service provisioning scenario to extract metrics and parameters that represent general data service characteristics. The approach enables the modeling of a data service as a collection of attributes that can be customized to adjust in various data service provisioning types.

In the illustrated scenario, a provider offers deployed relational databases. Customers interact with options of available service offers through a web interface that is connected with the provider's service management system. The web interface communicates to prospective customers available levels of service provisioning. Offered database schemas comply with alternative configurations in terms of data sharing, administration and storage partitioning. During the service selection and SLA negotiation phase, customers choose the level of database isolation.

In the following we describe the cloud infrastructure of the illustrated scenario and discuss attributes of the provi-

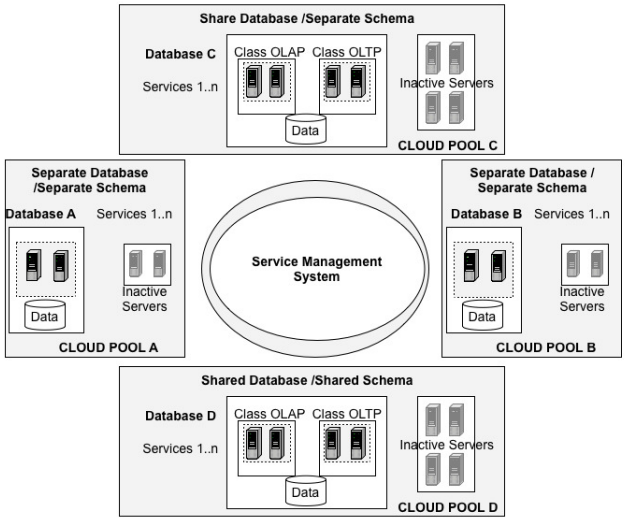

Figure 3: Data service cloud infrastructure

sioned data services. The process of specifying, generalizing and ordering such attributes with respect to the provisioned service types is necessary for their mapping into service level provisioning terms. The scenario is simplified to fit the paper demonstration scope.

3.1 Cloud Infrastructure description

Figure 3 illustrates the cloud architecture of the scenario infrastructure. Each cloud pool is configured as a server-cluster to host one or more database (db) types according to a predefined multi-tenancy model. Servers are deployed to operate for the optimized execution of diverse workload tasks. Every cloud pool scales-out by using idle processing power from inactive servers. The cloud infrastructure implements two types of database architectures: i) **share nothing**, where active database instances and persisted data are kept on the same server; ii) **shared disk**, where one or more database instances are running on each server and data are shared among all available servers.

The provider infrastructure includes a central service management system that is responsible for the administration of cloud pools. The administration unit controls the levels of server capacity for each cloud pool. A running process automatically adds to or removes servers from a cluster according to workload volume and scale out demand. The management system is configured to administrate database sessions for every running workload (e.g. connection rebalancing) and to handle monitoring and auditing processes of active SLAs. Moreover, the management system enables customers to login via a web portal to configure and monitor their leased services.

3.2 Data service attributes

We list and summarize attributes that apply to most data-service types and that influence the cost, performance, security and data isolation of the provisioned service.

The term **existing workload** denotes a running workload that is processed in the cloud infrastructure, remotely from customer local premises.

The term **complementary workloads** represents workloads that have similar performance characteristics and that run on the same server cluster, regardless of service or db type. Mixing non-complementary workloads may lead to missed SLA objectives, outages and poor consolidation of customer-specific deployment.

The **database (db) type** determines the level of service operations offered by the cloud provider. In the illustrated scenario we differentiate between **production** and **development** database. A production database has strict operational requirements that are related to its response time, availability targets (based on resource capacity) and security objectives. A development database has relaxed requirements with respect to service level targets.

Customer **workload requirements** represent a combination of read/write transactions that are processed by the provisioned database instance. In the scenario we assume that direct negotiation is abstracted and customers specify applicable service level values through a web interface [19]. Customer requirements primarily deal with transaction performance, especially with respect to execution time and processing throughput (e.g. transactions per second).

The **workload duration** is typically measured in seconds or milliseconds and represents the total workload execution time. It is described by a timeframe during which a planned number of transactions is executed.

The **transaction volume** indicates the number of processed transactions per second or millisecond. It represents the accumulative transaction number (read/write) that is processed during the workload duration from all active users.

The combination of workload duration and transaction volume signifies the **transactions per second/millisecond** (tps/tpm) metric and indicates the number of transactions executed by any workload. Transactions must be processed in a predefined timeframe. Workloads with the same tps can be merged and executed on the same server-cluster. The tps metric is used to define QoS parameters for every workload.

The **total workload IO** is measured in MB per second (MBps). Customer and data service provider estimate the workload IO volume and agree on the service cost and configuration parameters. Otherwise, the workload IO value is determined by the workload IO consumption during run time.

The **average execution time** (seconds or milliseconds) of workload transactions is a property metric that is driven merely by the **workload class**. In the scenario, the provisioning infrastructure supports two workload classes: the **OLTP** that is characterized by a large number of short online transactions (INSERT, UPDATE, DELETE) and the **OLAP** that is characterized by relatively low volume of transactions. Queries of the OLAP class are often very complex and involve aggregations.

The **max number of concurrent users** represents the maximum number of users that are connected in the system concurrently. This configuration property is used to manage user sessions and to avoid system overloads.

The **max memory per user connection** is measured in MB and signifies the maximum memory that is required in each database session.

The **max execution time** (seconds or milliseconds) of workload transactions is a property metric that is used by the provider to determine the workload peak time. If the

Resource/System characteristics	ST1	ST2	ST3
OS platform: Windows, Unix, Linux (32/64)	⋆		
CPU (family type/frequency)	⋆		
processor (nr/frequency)	⋆		
Database/Schema			
db instance/number of schemas	⋆	⋆	
db memory (MB)	⋆		
db memory per connection (MB)		⋆	⋆
db/schema size (MB/GB)	⋆	⋆	
db/schema growing ratio	⋆	⋆	
db/schema administration	⋆	⋆	
Database/Schema operations			
db shutdown/start-up	⋆		
on-demand db/schema/table backups	⋆	⋆	⋆
db/schema/table restores	⋆	⋆	⋆
db/schema/table error fixes	⋆	⋆	⋆
db admin option	⋆		

Table 2: Service properties I

Security	ST1	ST2	ST3
DB connection encrypted	⋆	⋆	⋆
Data encrypted	⋆	⋆	⋆
Tables/columns encrypted	⋆	⋆	⋆
DB/schema backup encrypted	⋆	⋆	
DBA access control	⋆	⋆	
DB/schema auditing/alert reports	⋆	⋆	⋆
Security profile (per schema)	⋆	⋆	
Availability			
Server/data (provider guarantee)	⋆	⋆	⋆
Stand by option DB/schema	⋆	⋆	
Back up option DB/schema	⋆	⋆	
Downtime during DB/schema migration option	⋆	⋆	
Scheduled downtime	⋆	⋆	⋆

Table 3: Service properties II

gap between the average and max transaction execution time is small, then the workloads are complementary, otherwise they are not.

3.3 Ordered attributes by data service provisioning type

In the illustrated scenario, the data service provider employs three data service types that are categorized as following:

Separate Database/Separate Schema (ST1), where customer workloads are executed on an isolated database that contains data from one customer only.

Shared Database/Separate Schema (ST2), where multiple customers execute their workloads in the same database and their data are stored in different database schemas.

Shared Database/Shared Schema (ST3), where multiple customers execute their workloads in the same database and their data are stored under the same schema.

Data service attributes accompany the description of each provisioning type and represent customizable or fixed SLA terms for a prospective tenant. Tables 2 and 3 demonstrate indicative attributes that are classified according to the data service provisioning type.

The first column of Table 2 lists attributes of resource and system characteristics as well as database, schema characteristics and operations. The other columns represent the three service provisioning types and their matching with listed characteristics. In Tables 2 and 3 the ($\star$) symbol indicates that an attribute is included in the service-type description.

The primary characteristic of the ST1 provisioning type is the database isolation offered to tenants. Customers have full control over their leased database and their data are physically isolated from the data of other tenants. Every leased database instance is considered as a separate server node (physical or virtual). Tenants of the first type can choose the operating system platform that will host their database. Additionally, they may adjust the values of system and resource attributes to their desired ones. If a customer prefers to avoid the risk of taking the complete db administration responsibility, a SLA arrangement can include alternative options, e.g. external administration services (supporting parties) and/or delegation of db administration priviledges to the service provider.

Customers of the first provisioning type can specify in the SLA the number of required db instances, the db memory size as well as the processing power and volume that is needed with respect to processor and CPU units. If applicable, a prospective tenant can indicate the initial size of each db instance and provide an estimation on the db growing ratio. All three provisioning types provide customers with configurable options regarding database, schema or table operations such as automatic db shutdown/start-up, on-demand db/schema/table backups, etc. A customer is provided with the SLA option to determine time-schedules for data recovery, e.g. recover an accidentally dropped table, rollback a single transaction etc.

In the ST2 provisioning type db resources are shared and their assignment depends from customer-provider arrangements. Tenants have no control over the db administration and configuration, but they fully manage their schemas. Customer data are isolated from the data of other tenants. A customer can estimate the maximum memory size to be consumed by each db session (e.g. max 5MB per connection). Additionally, a tenant can adjust the initial schema size and growing ratio i.e. the size of operating system files that are used to store db data and the percentage of schema space that is added per workload execution.

In the ST3 provisioning type, tenants share the same db instance and manipulate their data through the same schema. Tenants have no control over the database or the provided schema and their data are only partially isolated from the data of other tenants.

Customers of all service types have configuration options with respect to security and availability guarantees. A tenant can ensure through the SLA that all db connections as well as schemas, tables and table columns are encrypted. Tenants receive auditing and security reports in real time. Such reports specify controls with respect to when and how data and system modifications may occur. Customers can create security profiles for each application schema that is used by the database. Security profiles enable tenant control over database resources and schema security options (e.g. "failed login attempts", "password lock time", "password complexity"), if such are applicable.

In the scenario context, availability refers to the continuous database operation over a scheduled time interval. Availability characteristics are classified into two categories: server and data availability. Server availability specifies conditions of server failures, while data availability deals with data processing malfunctions. The service management system implements an adequate scale-out strategy to stay protected against server failures (e.g. timed-out servers are replenished by new ones that have already been active or are activated after the server failure).

A standby database provides disaster protection and protection against data corruption. A frequent database backup on pre-agreed time intervals (e.g. every 12 hours) eliminates data losses and guarantees service continuity. Database availability represents a series of SLA description and guarantee terms that the cloud provider offers to customers. Tenants are able to choose the type of database, schema or data backup (online or offline, export dump, logical backup) and customize or at least be alerted of timeframes between db, schema and data migration. Last but not least, scheduled db-downtimes (e.g. during upgrades, server patching etc) should be specified in the prospective SLA.

3.4 SLA template formalization

The SLA module that was described in Section 2, is used to generate a skeleton SLA template for the data service provisioning scenario. A SLA template represents a pre-instantiated agreement that is reviewed by customers or can be in an one-to-one negotiation process between a provider and a customer. Typically, SLA templates are aligned with the provider's resource provisioning plan.

We follow the element-set description of Table 1 and map data service attributes into the appropriate SLA term type. Elements that are granular and can be decomposed into finer service parameters are mapped as service- description or object terms. For example, the three data service provisioning types represent service-description subgraphs in the skeleton template as their formulation encloses many of the defined service attributes. There is no limit with respect to what and how many service-objects may describe a service-description node. Service-objects illustrate generic service attributes and typically require multiple SLA parameters, composite or resource metrics for their formulation.

Nodes and edges are accompanied with properties that are formed as key-value paired dictionaries in the node/edge definition. By default, each node in the produced graph includes as properties: uid, name and weight. Service description nodes that are lower in the SLA hierarchy tree include in addition: measurement unit, type and value. Edges include as properties the uid of the precedent node and a default weight that can be adjusted according to given application needs.

Figure 4 illustrates a service description subgraph of the complete SLA template, which can be found in [18]. The "SLA" node of the produced digraph is connected with three service description subgraphs, one for each service type. "Resource and System characteristics" represent a service-object node that applies only to the ST1 provisioning type. The service-object node is connected with three SLA parameters: the "Operating System", "Processor" and "CPU" nodes.

The "Operating System" node embeds as properties available OS options. The "Processor" and "CPU" nodes are connected with composite and resource metrics that estimate the SLA parameter values with respect to the number of available processors and processor frequency as well as CPU type and CPU efficiency. Composite metrics use the value of resource metrics for their definition [10]. The description of resource metrics includes the port or query interface, where the value of a metric is measured. Connecting edges define the measurement and evaluation schedules of composite and resource metric values.

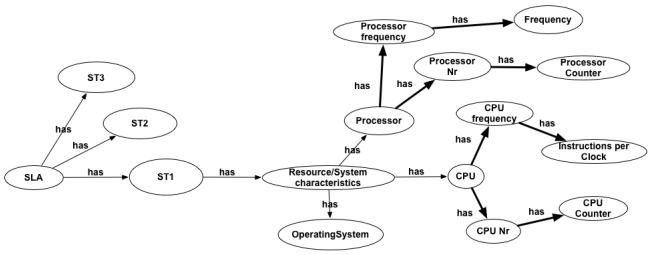

Figure 4: Service description subgraph - Data service scenario SLA template [18]

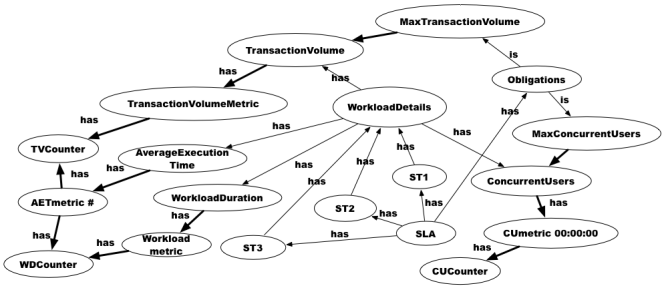

Figure 5: Service, gurantees subgraph - Data service scenario SLA template [18]

Figure 5 depicts the relationship of the "WorkloadDetails" node with neighboring nodes from the obligations subgraph. "WorkloadDetails" represents a service-object node that applies to all service types. This service object contains SLA parameters like the transaction volume, workload duration and average execution time. Such parameters are accompanied by metrics that help to determine the parameter value ranges. Moreover these parameters are used by guarantees in the form of SLOs to specify limits and objectives of service levels.

In total the produced template contains 137 nodes that are connected through 109 edges. The graph size is indicative as in a real business setting such a graph may contain thousands or millions of subgraphs, nodes and connected edges. Moreover, numerous SLA instances can be derived from a single SLA template [1], given application criteria and provisioning optimization needs.

4. SLA GRAPH CHALLENGES

The SLA formalization and management through a graph structure provides three immediate advantages with respect to data manipulation:

Extensibility: There are numerous forms that a SLA graph may take to meet application and business requirements. Instead of handling SLA information through schema-

less log files and diverse data structures, a single graph or a graph pool can be used to store and manage all relevant SLA data. In this work we presented a skeleton SLA digraph and demonstrated its simple instantiation given a realistic data service provisioning scenario.

Modularity: Graph nodes are naturally inter-connected, which allows for data interaction and for a multitude of data exchange operations. SLA information can be treated separately and combined dynamically. Levels of information granularity may differ within the same SLA graph as the latter may include subgraphs and customized representations of SLA data. Information modularity is an important advantage for SLA management. SLA data is typically volatile and their values may influence the values of nearby nodes. A modular structure and a granular, while customizable, data schema can be supported by graph traversals and coordinated data operations over multiple node-sets.

Data mapping and clear data flow: The mapping of SLA information into nodes and edges enables the storage and management of data in a structured way. Formalizing the SLA structure as a digraph allows for data flow clarity and permits the classification of data dependencies as well as their automatic retrieval through, for example, regular path queries (RPQs). This is important for data intensive applications as the SLA processing can be treated as a workflow that is scheduled to perform operations over semi-structured, dynamical data-series (e.g. value measurements, updates, alert notifications). When a massive volume of information has to be processed concurrently, an adequate workflow strategy can handle the efficient execution of data tasks. The realization of the graph as directed permits the orchestration of workflow operations that consider dependencies between antecedent and precedent nodes or node-sets.

The data quality and semantic accuracy of the SLA digraph have to be justified with respect to the type of queries that are posed and processed through a graph-supportive DBMS. In this context, the process of evaluating the query expressiveness, accuracy and presumably semantic correctness can be supported by a comparative mapping between natural language and graph query formulation. Moreover, a comparison with existing approaches for SLA manipulation (e.g. XQuery) can justify the appropriateness of the graph data structure for distributed service management.

The efficiency of the proposed data model with respect to system or application scalability and throughput can be evaluated by executing parallel data queries over distributed virtual hosts. Complex SLA queries can be processed using graph algorithms. Our interest concentrates on the SLA data management performance over the HTTP layer, where exchange of information is massive and immediate. From this perspective, scalability deals with the distribution of SLA data over remote graph repositories and with traversals to retrieve and filter information from the remote locations simultaneously.

The SLA graph representation allows for multiple uses given alternative application scenarios. Edge and node properties become of great importance as they indicate how to define the graph in terms of efficient path classification and optimal query processing. For example, vertices can be ordered according to their SLA operations (e.g. special vertices for SLA violations and signatory compensations). Moreover, the graph can be extended to consider a cost model [9] that introduces revenue paths to a provider based on available resource capacity and satisfactory SLA packages to customers given their requested provisioning criteria.

5. RELATED WORK

Research on SLA management is seasoned in terms of formalization [1, 13], semantic ontologies for metric categorization [6, 17] and monitoring [3, 15, 20]. Scientific results have also contributed in resource scheduling for computational workloads [4, 7], distributed service management [10, 16] and SLA negotiation [2]. In the cloud computing context, data-relevant scientific efforts have augmented on the SLA utilization for efficient manipulation of data intensive applications [11, 12, 14]. To our knowledge, very limited scientific work focuses entirely on SLA data management.

In [22] the authors elaborate on SLA semantic graphs using an in-depth analysis of service guarantees and objectives that are derived from real service offers. The authors define a SLA as a legal contract that specifies the minimum expectations and obligations between a provider and a customer. They refer to their model as the necessary means to design database schemas for the SLA information. Their SLA semantic model is based on UML and is formed as a relationship diagram that indicates the logical information flow among principal SLA elements.

We consider the approach in [22] complementary to ours. The authors label identified graph relationships with semantic, unidirectional attributes to describe the primary data flow among business SLA components. Their model includes the notion of service package graphs and resembles a hypergraph of service bundles that contains transition triggers from one service package to another according to predefined conditions. The SLA model in [22] signifies business perspectives related to service provisioning. Our model proposes a modular and extensible approach to structure and manage the SLA information flow.

6. CONCLUSIONS, ONGOING WORK

The paper discussed the formulation of a SLA data model as a property digraph. In the first part of the work we elaborated on the SLA anatomy according to the WSLA specification [13] and on the WSLA transformation into a digraph for SLA data management. We described the implementation of a minimal Python module that uses the NetworkX library [8] to generate SLA templates in the form of digraphs. Our analysis concentrated on the internal dependencies between SLA components and on how the digraph structure addresses such functional attributes through unidirected connections.

The second part of the paper described a realistic scenario of data service provisioning, where a provider offers deployed relational databases. The scenario details (e.g. explicit property values) were simplified for the paper demonstration scope. We used the scenario to extract service metrics and SLA parameters that comply with general data service characteristics. The data service attributes were ordered according to the provisioning types of the featured scenario. The digraph module was used to illustrate a SLA template for the scenario specifics. We described an empirical methodology for the mapping of data service attributes into SLA terms and analyzed two subgraphs of the produced SLA template, which can be found in [18].

The discussion continued with an analysis on the immediate advantages, but also on the research questions that the graph schema generates for the efficient management of SLA information. The graph data structure provides an extensible, modular schema that enables the transparent data flow of enclosed graph elements. At the same time, the proposed data model needs to prove its validity with respect to semantic accuracy, completeness and scalable handling, which can be introduced with the processing of SLA information over distributed virtual hosts.

The identified research challenges summarize our current efforts, where the proposed SLA digraph is deployed for the Titan [21] graph database with the backend support of Cassandra[1] as persistence layer.

7. ACKNOWLEDGMENTS

This work is supported by the Swiss National Science Foundation (SNSF), grant number 200021E-136316/1.

8. REFERENCES

[1] A. Andrieux et al. Web Services Agreement Specification (WS-Agreement), 2005.

[2] K. Czajkowski et al. Snap: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. In *Revised Papers from the 8th International Workshop on Job Scheduling Strategies for Parallel Processing*, JSSPP '02, pages 153–183, London, UK, 2002. Springer-Verlag.

[3] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *High Performance Distributed Computing, 2001. Proceedings. 10th IEEE International Symposium on*, pages 181–194, 2001.

[4] K. Czajkowski, I. Foster, and C. Kesselman. Agreement-based resource management. *Proceedings of the IEEE*, 93(3):631–643, 2005.

[5] A. Dan, H. Ludwig, and G. Pacifici. Web service differentiation with service level agreements. *White Paper IBM Corporation*, 2003.

[6] G. Dobson and A. Sanchez-Macian. Towards unified qos/sla ontologies. In *Proceedings of the IEEE Services Computing Workshops*, SCW '06, pages 169–174, Washington, DC, USA, 2006. IEEE Computer Society.

[7] I. Foster, A. Roy, and V. Sander. A quality of service architecture that combines resource reservation and application adaptation. In *Quality of Service, 2000. IWQOS. 2000 Eighth International Workshop on*, pages 181–188, 2000.

[8] A. Hagberg, D. Schult, and P. Swart. NetworkX. `http://networkx.github.io/`. Accessed: March, 2013.

[9] V. Kantere. A framework for cost-aware cloud data management. In *On the Move to Meaningful Internet Systems: OTM 2012*, volume 7565 of *Lecture Notes in Computer Science*, pages 146–163. Springer Berlin Heidelberg, 2012.

[10] A. Keller and H. Ludwig. Defining and monitoring service-level agreements for dynamic e-business. In *Proceedings of the 16th USENIX conference on System administration*, LISA '02, pages 189–204, Berkeley, CA, USA, 2002. USENIX Association.

[11] T. Kraska, M. Hentschel, G. Alonso, and D. Kossmann. Consistency rationing in the cloud: pay only when it matters. *Proc. VLDB Endow.*, 2(1):253–264, Aug. 2009.

[12] Z. Liu et al. PMAX: tenant placement in multitenant databases for profit maximization. In *Proceedings of the 16th International Conference on Extending Database Technology*, EDBT '13, pages 442–453, New York, NY, USA, 2013. ACM.

[13] H. Ludwig et al. Web Service Level Agreement (WSLA) Language Specification, 2003.

[14] O. Papaemmanouil. Supporting extensible performance slas for cloud databases. In *Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on*, pages 123–126, 2012.

[15] O. Rana, M. Warnier, T. B. Quillinan, and F. Brazier. Monitoring and reputation mechanisms for service level agreements. In *Proceedings of the 5th international workshop on Grid Economics and Business Models*, GECON '08, pages 125–139, Berlin, Heidelberg, 2008. Springer-Verlag.

[16] R. Rodosek and L. Lewis. A user-centric approach to automated service provisioning. In *Integrated Network Management Proceedings, 2001 IEEE/IFIP International Symposium on*, pages 273–276, 2001.

[17] S. Seo, A. Kwon, J. Kang, and J. Hong. OSLAM: Towards ontology-based SLA management for IPTV services. In *IFIP/IEEE International Symposium on IM*, 2011.

[18] K. Stamou. SLA digraph, complete RDBaaS template. `http://thinkcloud.unige.ch/SLAgraph`. Created: April, 2013.

[19] K. Stamou, V. Kantere, and J. Morin. SLA template filtering: a faceted approach. In *4th International Conference on Cloud Computing, GRIDs, and Virtualization*, 2013.

[20] P. Stelling et al. A fault detection service for wide area distributed computations. In *High Performance Distributed Computing, 1998. Proceedings. The Seventh International Symposium on*, pages 268–278, 1998.

[21] ThinkAurelius. Titan Distributed Graph Database. `http://thinkaurelius.github.io/titan/`. Accessed: May, 2013.

[22] C. Ward, M. J. Buco, R. N. Chang, and L. Z. Luan. A generic sla semantic model for the execution management of e-business outsourcing contracts. In *Proceedings of the Third International Conference on E-Commerce and Web Technologies*, EC-WEB '02, pages 363–376, London, UK, 2002. Springer-Verlag.

[1]Cassandra, `http://cassandra.apache.org/`