Cloud Data Denormalization for Platform-As-A-Service

Aspen Olmsted
Department of Computer Science
College of Charleston, Charleston, SC 29401
e-mail: olmsteda@cofc.edu

Abstract— **In this paper, we investigate the problem of representing transaction data in PAAS cloud-based systems. We compare traditional database normalization techniques with our denormalized approach. In this research, we focus on transactional data related to an organization's customers. Some optimization comes from the absence of a known customer object, which allows for the vertical merging of tuples. Instead of storing detail transactional data, data is stored in aggregate form. The journaling features of the data store allow for full audits of transactions while not requiring anonymous data to be materialized in fine-grained levels. The horizontal merging of objects is also deployed to remove detail lookup data instance records and one-to-many leaf node records.**

Keywords-web services; distributed database; modeling; cloud computing

I. Introduction

In this work, we investigate the problem of representing transactional data in a platform as a service (PAAS) cloud-based system. In traditional client-server architectures, database normalization is used to ensure that redundant data does not exist in the system. Redundant data can lead to update anomalies if the developer is not careful to update every instance of a fact when modifying data. Normalization is also performed to ensure unrelated facts are not stored in the same tuples resulting in deletion anomalies. Our earlier work [1] focused on the minimization of storage requirements for anonymous transaction data in PAAS cloud storage. This work extends that research, by increasing the optimizations to include enterprise integration, mobile integration and the modeling of the workflow and lifecycle of objects in a PAAS system.

Data representation in the cloud has many of the same challenges as data representation in client/server architectures. One challenge data representation in the cloud has that is not shared with client/server is the minimization of data. This challenge exists because the costs of cloud data storage are significantly higher than the costs for local storage. When we say higher costs, we mean the simple, measurable costs for the disk storage, not the true costs of managing and accessing the data over the life of the application. Organizations have traditionally budgeted the costs of disk drives for local storage which are in the tens of dollars per gigabyte. Similar cloud storage can be in the hundreds of dollars per gigabyte per month [1]. Often this storage is expressed as the number of tuples in the data store instead of the number of bytes on the disk drive holding the data. For example, force.com [2] charges for blocks of data measured in megabytes but they calculate usage as a flat 2KB per tuple. Zoho Corporation also tracks data storage by the tuple for serval of their cloud products including Creator [3] and CRM [4]. The tuple count method is used as it is easier to calculate in a multi-tenant system where the physical disk drives are shared by many clients.

In this paper, we present an algorithm that will minimize the number of tuples used to store the facts for a software system. We use a motivating example from a cloud software system developed by students in our lab. The algorithm performs three main operations:

- The horizontal merging of objects – several distinct relations are combined into one.
- The vertical merging of objects – several distinct instances of the same type of facts is combined into one.
- Business rule adoption – instead of storing tuples to represent availability of lookup data, we replace the tables with pattern based business rules

We apply our algorithm to a system in the humanities application domain and show an approximately 500% reduction in tuple storage.

Date [5] invests a good deal of his text on the definition of denormalization. He argues that denormalization is when the number of relational variables is reduced, and functional dependencies are introduced where the left-hand side of the functional dependency no longer is a super key. The practical realization of Date's denormalization is that the primary key does not directly determine attributes in the tuple, leading to update and deletion anomalies in exchange for better performance or storage. In our work, we perform many optimizations. When we horizontally merge relations, then we are performing a true denormalization in Date's definition. Other optimizations such as vertical merging do not fit Date's definition of denormalization. We choose to stay with the term denormalization algorithm as it is a set of steps taken after the normalization process to optimize an aspect of the data model.

The organization of the paper is as follows. Section II describes the related work and the limitations of current methods. In Section III, we give a motivating example where our algorithm is useful and describe our denormalization algorithm. Section IV describes additional enhancements through the design of business rule objects. Section V explores reporting from the denormalized objects utilizing the object version history stored in the journal. Section VI contains our comparison of the proposed method and the traditional database normalization method. We explore the denormalization when applied to an object's workflow and lifecycle in Section VII. In Section VIII, we add an additional optimization to handle one-to-many lookup storage of data. Section X investigates the denormalization algorithm when applied to mobile computing. Section XI considers our data

2017, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

model in the context of web-service database. In Section XII, we analyze the security vulnerabilities that are reduced through our data modeling technique. Section XIII gives a discussion of our experience through this work. We conclude and consider future work in Section XIV.

II. Related Work

Sanders and Shin [6] investigate the process to be followed when denormalization is done on relational data management systems (RDBMS) to gain better query performance. Their research was performed before the cloud database offerings became prevalent. In the cloud, database performance is less of an issue to storage requirements because the systems are designed to distribute queries across many systems.

Conley and Whitehurst [7] patented the idea of denormalizing databases for performance but hiding the denormalization for the end user. Their work focuses on merging two relations into one relationship to eliminate the processing required to join the records back together. Their work uses horizontal denormalization to gain performance. Our work uses both horizontal and vertical denormalization to minimize storage space and increase usability.

Most denormalization research work was done in the late 1990s and was focused on improvement in query performance. The performance was an exchange for a loss of correctness and usability of the data. Recently, folks like Andrey Balmin have looked at denormalization as a technique to improve the performance of querying XML data. Like the previously mentioned research, this work differs from our work in the desired end goal. Our end goal being the minimization of data storage and improvement in end user usability.

In Bisdas' [8] blog, he demonstrates ways that end users can improve data visualization by vertically merging hierarchical data in the Salesforce, data model. He takes advantage of the trigger architecture to create redundant data in the hierarchy. Taber [9] also recommends denormalization to improve data visualization. The problem with both solutions is that data storage requirements are increased while correctness is jeopardized by the redundant data.

In one of our previous publications [10], we study UML models from the perspective of integrating heterogeneous software systems. In this work, we create an algorithm to sort cyclical UML class data diagrams to enable transaction reformation in the integration. In the process, discoveries were made on the freshness of data at different layers in the UML graph. The knowledge is useful in this study when considering anomalies that can happen in response to data updates.

Additional semantics for models can be added by the integration of the matching UML Activity and Class diagrams. UML provides an extensibility mechanism that allows a designer to add new semantics to a model. A stereotype is one of three types of extensibility mechanisms in the UML that allows a designer to extend the vocabulary of UML to represent new model elements [11]. Traditionally the semantics were consumed by the software developer manually and translated into the program code in a hard coded fashion.

Developers have implemented business rules in software systems since the first software package was developed. Most research has been around developing expert systems to process large business rule trees efficiently. Charles Forgy [12] developed the Rete Algorithm, which has become the standard algorithm used in business rule engines. Forgy has published several variations on the Rete Algorithm over the past few decades. In this work, we focus on the representation of the business rules in the data model.

Our previous work [13] on data modeling for the cloud focuses on benefits gained by aggregating anonymous data. These benefits and the research behind that study is covered here along with further work to minimize data storage requirements for one-to-many data along with schema denormalization when using predefined object schemas.

III. Denormalization

We demonstrate our work using a Tour Reservation System (TRS). TRS uses web services to provide a variety of functionalities to the patrons who are visiting a museum or historical organization. We use the UML specification to represent the meta-data. Figure 1 shows a UML class diagram for an implementation of this functionality. The Unified Modeling Language includes a set of graphic notation techniques to create visual models of object-oriented software systems [13]. In this study, we use data collected by the Gettysburg Foundation on visitors to their national battlefield. The system is modeled and implemented on the force.com [2] cloud platform.

Figure 1 shows a normalized UML class model of reservation transactions of visitors to the Gettysburg National Battlefield. In the model, the central object ticket represents a pass for an entry that is valid for a specific date and time and a specific activity. Activities are itinerary items the visitor can be involved in while visiting the battlefield. In the normalized model, each ticket is linked to a specific activity

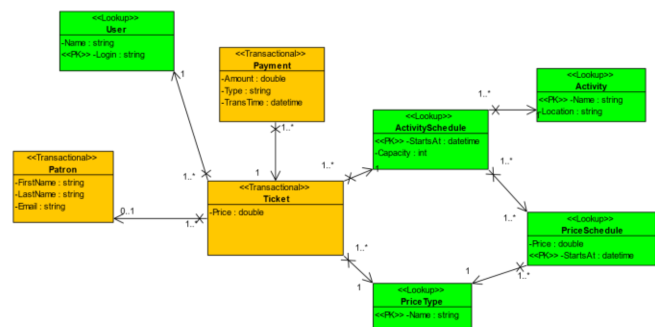

Figure 1. Normalized Transaction Model.

2017, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

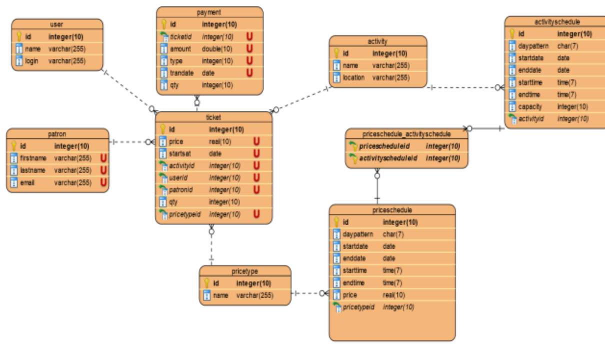

Figure 2. Denormalized Transaction Model.

schedule entry that will designate the date and time the pass is valid for entry. Each activity schedule is linked to an activity object that designates the name and location of the activity.

Each ticket is linked to a user in the Gettysburg organization who was responsible for the transaction. Each ticket can be linked to a patron object. In the case of advanced reservations, there will be a valid patron object linked to the ticket. Advanced reservations are transactions that take place through the organization's website or over the phone to a reservation agent. In the case of walk-up transactions, there will not be a linked patron. A walk-up transaction is a transaction that takes place when a visitor arrives on the site without a prior reservation and pays for the ticket at the front desk.

In Figure 1, the multiplicity of the association between the patron and the ticket is a zero or one to many. A multiplicity that can be zero represents anonymous data. Anonymous data is data that does not need to be specified in order for the transaction to be valid. In the example transaction, the patron can remain anonymous but still visit the battlefield and partake in the activities. In the case of the sample Gettysburg data, 60 percent of ticketing transactions were anonymous.

In the case of the force.com [2] PAAS, data storage is charged by the tuple. With an enterprise license to the platform, the organization is granted access to one gigabyte of data storage. The storage is measured by treating every tuple as two kilobytes. This form of measurement allows the organization 500,000 tuples in the enterprise data storage option. The data collected for the normalized data model would allow the Gettysburg organization to store around nine months' worth of data. With the anonymous transaction, the ticket and the payment data is only important on the original transaction level for auditing. For example, the accounting department may want to see the details behind a specific ticket agent's cash total for the day. Another example would be the marketing department wants to see the ticket price patterns within an hour of the day.

The force.com [2] platform uses an Oracle relational database to deliver the data storage services but adds a journal feature so history can be stored on all changes to an object over time. This journal can be used at no additional data storage cost. The field level changes stored in the journal would allow aggregate data to be stored for anonymous transactions and still have the detail to perform the audits mentioned earlier.

If an object is used between two other objects where the middle object is the "many" side of the one-to-many relationship and the one side of the other relationship, then the same data can be represented by moving the attributes to the object on the composition side of the relationship. The middle object is then able to be removed, reducing the number of tuples representing the same amount of data. In Figure 1, the "Activity Schedule" object fits this profile and can be horizontally merged with the "Ticket" object. In our previous work [10], we study UML data model freshness requirements and document the relationship between data changes and location in the UML graph. In our findings, we see that middle object nodes are less predisposed to changes than leaf nodes. The lower amount of data changes reduces the change of update anomalies.

In Figure 1, we also designate objects that are updated in transactions differently than objects that are navigated for transactional lookup values. Two stereotypes are added to the diagram:

- Transactional – The classes designated with the orange color and the <<Transactional>> tag are updated during transactional activities.
- Lookup – The classes designated with the green color and the <<Lookup>> tag are not updated

Algorithm 1. Denormalization Algorithm.

INPUT: normalizedObjects (XMI representation of UML class diagram)
OUTPUT: denormalizedEntities (XMI representation of denormalized entities)

```
foreach object in normalizedObjects
   add entity to denormalizedEntities
   foreach attribute in object
      add attribute to entity
      if object is transactional
         mark attribute as unique
   add id attribute as primary key
   mark id as autoincrement

foreach entity in denormalizedEntities
   if entity is both a many side and a one side of two
relationships
    and  a lookup object
      foreach attribute in object
         if attribute is PK
            add attributes to many side entity
            if attribute is a datetime type expand date pattern
      swap graph location entity of the one sides

foreach association in normalizedObjects
   if association is one-to-may and many side is transactional
       add foreign key to many side entity
       add quantity field to entity on many side
```

2017, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

during transactional activities. The data in these classes are created by administrative activities. During transactions, the data is searched for the proper values.

The Denormalization Algorithm, Algorithm 1, transforms a normalized model stored in a UML class diagram into a denormalized model represented as an entity-relationship diagram. The algorithm assumes input and output of the models in the XMI [14] format. XMI is a standard exchange format used to represent structural models in a non-proprietary way.

The algorithm first loops through each object in the normalized model and adds the object and attributes as entities in the denormalized entity-relationship diagram. If the object has the transactional stereotype, then the attributes are marked unique. Surrogate Identifier fields are added to each object's definition to be used as an auto-incrementing primary key.

The next pass of the algorithm is to find objects that can be eliminated from the middle relationship of two "one-to-many" relationships. The original model, in Figure 1, had an activity schedule object that consumed a lot of data space by storing a lot of tuples to represent the occurrences an activity can take place. We use a stereotype of "PK" applied to attributes in the original model to designate the primary identifier for instances of an object. This designation allows us to shift the attribute down the association and swap the positioning of the objects. In this iteration over the objects, we also look for date-time data types that are part of the primary key. When we locate an occurrence, we replace the attribute with a date-time specification occurrence. The date-time specification includes a starting date, ending date, starting time, ending time and day of the week pattern.

The final pass of the algorithm adds foreign keys and aggregation counters. The aggregation significantly reduces the count of tuples stored. An example of this is shown in Figure 1. Instead of having an instance of each ticket, we add the quantity field to store the aggregate count for the unique attributes.

Algorithm 2. History Creation Algorithm.

INPUT: object
OUTPUT: collection of object's version history

```
Set thisObject = newest version of object
Set objectVersions = empty list
Set fieldVersions = distinct saveDates values from object journal
Sort fieldVersions by saveDate descending
Set lastDate = maximum(saveDate)
Foreach version in fieldVersions
    If lastDate = version.saveDate
       objectVersions.add(thisObject)
   Set thisObject.[version/attribute] = version.value
   Set lastDate = version.saveDate
Return objectVersions
```

Figure 2 shows an entity-relationship diagram of a transformed model of Figure 1. Unique attributes have been applied where aggregations should be performed. The activity schedule entity has been shifted out in the graph, and the quantity fields have been added to the aggregated transactional objects.

IV. BUSINESS RULES

Business rule engines have sprung up to allow the separation of business rules from the core application code. The systems are designed to allow the end users to change the business rules freely without changing the original application code. In 2007, International Data Corporation implemented a survey where they asked 'How often do you want to customize the business rules in your software?'. Ninety percent of the respondents reported that they changed their business rules annually or more frequently. Thirty-four percent of the respondents reported that they changed their business rules monthly [15].

Figure 2 shows two tables that implement business rules:

- Activity Schedule – This table implements the date-time pattern mentioned earlier to store the business rules for when a particular activity is valid.
- Price Schedule – This table implements the date-time pattern mentioned earlier to store the business rules for when a particular price is available.

In each case objects in Figure 1, which inserted instances to represent availability, are replaced with rule instances to represent the availability. So instead of having a tuple per availability instance, a single tuple can represent the pattern. In the case of activity schedules, the example year had over 26,000 instances of availability that were replaced with 30 instances of the business rule.

V. OBJECT HISTORY ANALYSIS

One of the main reasons an enterprise develops or purchases a software solution is to allow the organization to increase their knowledge of their operations through the analysis of the data collected in the software solution. The denormalization solution presented earlier may limit the data

TABLE I. EMPIRICAL RESULTS.

Table	*Normalized Tuples*	*Denormalized Tuples*
user	*31*	*31*
patron	*17,610*	*17,610*
ticket	*738,981*	*157,780*
activity schedule	*26,697*	*30*
price schedule	*220*	*24*
activity	*17*	*17*
Total	*783,556*	*175,492*

2017, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

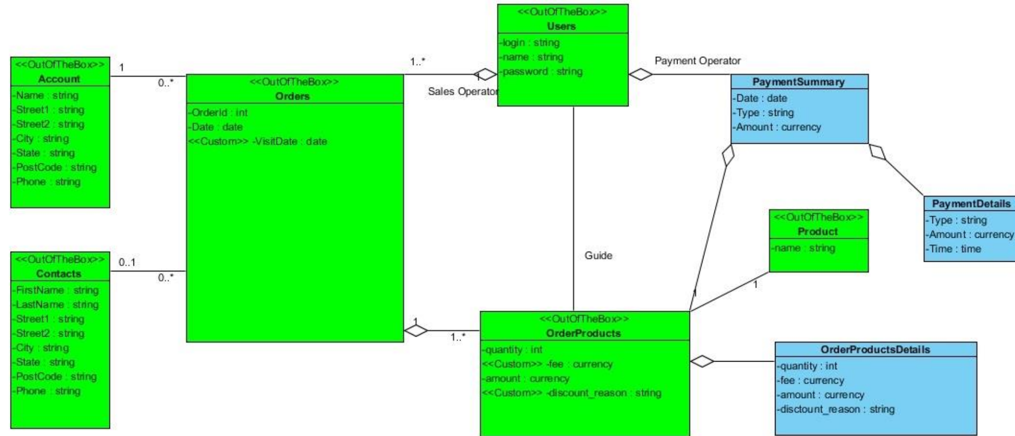

Figure 3. Order Data Model.

available from the denormalization process. The data is presented to the users through dashboards, reports or exports. A dashboard is presented as a graphical chart to measure where the organization stands compared to a goal. Examples of these would be sales to date compared to same period last year. A report has a set of input parameters that control the data displayed. The data displayed in the report tends to include tables with aggregated values. Exports allow for the exporting of data into a two-dimensional table saved as a comma separated value (CSV) format. In this format, attributes represent the columns of the data. Columns are escaped with double quotes and separated by commas. For our purposes, we will refer to all three categories generically as reports.

Current state and historical comparison are the two categories of reports a user may want to pull in their analysis. In current state reports, only the latest version of the object is needed. In historical comparison reports, all versions of an object may be needed depending on the level of aggregation. An example of a historical comparison report would be a report that compares sales for the month compared to sales last year in the same month.

In our work, we developed Algorithm 2 to create an in-memory copy of all historical versions of a specific object. We use code to generate the data and then generate the report output. If the organization wanted to allow end users to report on historical versions, they could modify Algorithm 2 to write records as temporary tuples and then call the reporting tool.

VI. Empirical Results

The empirical results demonstrate the success of representing the example transaction data with significantly lower cloud storage costs. TABLE I shows the tuple counts for the original data model and the denormalized data model. Both data models represent the complete 2014 calendar year of visitor transactions for the Gettysburg National Battlefield. The denormalized model creates a 78% reduction in the number of tuples. In the specific case of the force.com [2] platform, the reduction in the number of tuples allows the

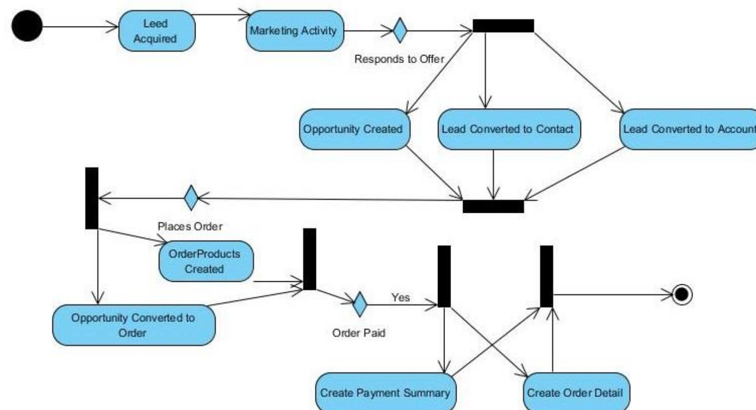

Figure 4. Advanced Reservation Workflow Model.

2017, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

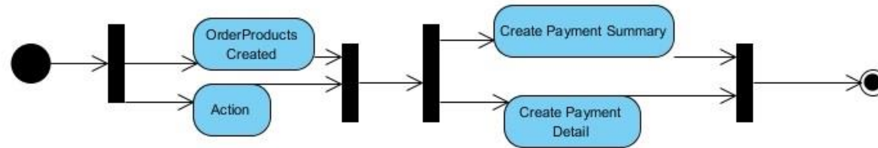

Figure 5. Frontdesk Workflow.

organization to store nearly three years of transaction data in the data storage provided without additional subscriptions costs. In the minimal data storage provided to an enterprise customer of force.com [2], the organization receives 500,000 tuples. Additional data storage is available to the organization for a monthly subscription price of 2,000 tuples per dollar. Using the normalized data model to represent the transactional data a complete year of cannot be stored without purchasing more data storage.

VII. MODELING WORKFLOW

The first phase of research in this paper modeled the reservation transactions using domain specific objects. Using domain-specific objects is the method to use if a designer is implementing greenfield engineering. The cloud PAAS platform we used in the first phase of the research, includes a full Customer Relationship Management (CRM) system to store interactions with customers Out of the box the CRM provides objects to hold sales orders and the workflow with the customer before they place an order. Figure 4 shows the model of the workflow for the advanced sales operations. Leads are acquired through public events such as conference tabeling and information sessions. After a lead is acquired a marketing activity takes place where the lead is related to the new activity. Marketing activities may include email marketing, phone calls, in-person meetings and online meetings using a technology such as Skype or Google Hangouts. When more information is gathered about the potential visit from the customer, the lead is converted into a contact object, account object, and opportunity object. The contact object holds an individual's biographical details such as first name, last name, email, address, and phone numbers. The account object holds details about the organization the customer is associated with. This data includes organization name, address, and phone number. The opportunity object

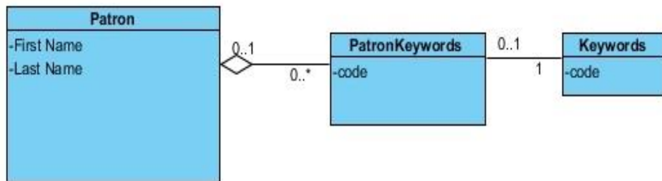

Figure 6. One-to-Many Relationship.

stores the details of the products the visitor is likely to order. These objects are shown in Figure 3.

The workflow continues when the customer makes a commitment about their visit. At this point, the opportunity is converted into two new out-of-the-box objects; Orders and OrderProducts. Finally, when the payment is made by the customer, a PaymentSummary record, and a PaymentDetail record is inserted into the system. The PaymentDetail record is stored for auditing purposes and is purged after the fiscal period is closed out. Individual payment details were only used by the individual operators to settle out their daily cash drawers and by management to audit individual operator cash drawers. Once the fiscal period has closed the details of multiple payments related to individual transactions is no longer needed.

Figure 4 models the workflow used with non-anonymous transactions. To model the collection of the anonymous data, we included Figure 5. The model of the collection of anonymous data is a much shorter workflow where less data is collected. Figure 3 included two detail objects; OrderProductDetails and PaymentDetails. When data is created in the anonymous workflow the summary level object is aggregated on the user and visitation date. The detailed data is required on anonymous transactions for auditing purposes until the fiscal period is reconciled. The payment and order detail objects are purged after the fiscal period is closed. TABLE II shows the tuple count of the normalized data and the denormalized data from this methodology. The table shows an eight-nine percent reduction of the tuple count.

TABLE II. EMPIRICAL RESULTS.

Table	*Normalized Tuples*	*Denormalized Tuples*
Accounts	7,550	7,550
Contacts	15,172	15,172
OrderProducts	519,422	110,675
Orders	176526	16,545
Payments	717,446	7,310
Products	38	38
Users	178	178
Total	1,436,332	157,468

2017, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

VIII. One to Many Lookup Codes

In traditional database design, one-to-many relationships are represented in separate objects. Figure 6 shows a relationship we observed in another organization. The New York Philharmonic stores keywords as a way to tag patrons with different attributes. This design is the output of the normalization process used to eliminate update and deletion anomalies [16]. Relational databases use b-tree indexes that allow a change in the lookup value to be populated down to the joining table in log N time.

An analysis of the data reveals that on average each patron had six keywords associated with their record. To represent five hundred thousand patrons with this design requires a little over three million five hundred thousand tuples. Bitmap indexes allow a database field to store bits to represent different discrete values in a single field. In the example above, a multi-select field stored in patrons to represent the keywords associated with patrons would reduce the tuples to around five hundred thousand. A single bit of the multi-select field uses a bit to represent the presence or absence of the lookup value. A bitmap index stores separate lists of the tuples where the bitmap is turned on to allow the search time for queries, updates, and deletes to be below N search time. The bitmap allows more data to be stored in fewer tuples but also allows fast retrieval time.

IX. Departmental functional integrations

Enterprise transaction processing systems support several different use cases to fulfill the entire set of requirements of an organization. An organization will partition an enterprise system at the department level for several different reasons. Two of these reasons are a simplification of the functional model and to enable geographic proximity to the users entering the transactional data.

The result of the departmental partitioning is a duplication of data across departmental systems, and the management of this duplication is a difficult problem. Often an organization will enter this data manually in each local system. The organization is forced to tolerate the data inconsistencies that come from the difference in human interpretation of the source data and transcription differences.

We sampled a few enterprise organizations data in search of duplication of biographical information (customers, addresses, and telephone numbers). Biographical information is easier to correct than other domains as there are standard algorithms to clean the data. These algorithms include address standardization using the postal service CASS database [1] and move update database [2]. We applied the cleaning of the biographical data to the sample. After, we found there was still a 17% duplication of biographical data collected in the individual departmental systems over a 10-year period.

Functional differences across enterprise departments make it difficult for a single system to meet all the needs of the organization. An organization could choose to relax functional requirements in exchange for better data quality, but this option is often not considered. We surveyed the 41 member organizations of CIO Arts [3] to find the threshold between functional requirement priority and system partition preference. The member organizations are performing arts centers that have three specialized departmental system needs (Box Office, Fund Development, and Event Management). In the study, it is clear that even low priority functional requirements take precedence over choosing a single enterprise system.

Disperse Geographic locations also require departments to use separate systems to enable each department to keep its data local. Localization avoids network partition problems and improves system performance. Cloud providers offering infrastructure as a service or platform as a service are an alternative to geographic partitioning. Unfortunately, these platforms are relatively new compared with the live cycle of vertical market enterprise systems. Typically, vertical market systems are built using a client-server architecture that requires low latency response time, making them inappropriate for cloud providers. In our CIO Arts survey, we found all organizations used a Microsoft Windows client-server application.

Often system integrations are built to enable data to be automatically exchanged between departmental systems. The integration is instead of having a single enterprise system. These processes can have high latency in the case of large volumes of transactional data updates. The latency problem can lead to incorrect data and improper decision-making. With cloud-based enterprise systems, the data needs to be pulled or pushed between systems.

TABLE III shows the bandwidth differences for synchronization of one year of our example data between two cloud systems. The denormalized data represents an eighty-eight percent reduction in bandwidth requirements and also represents a similar reduction in synchronization times.

X. Offline Mobile Applications

Mobile computing has become ubiquitous over the past decade due to the proliferation of smartphones, tablet

TABLE III. BANDWIDTH IN KB.

Table	*Normalized Bandwidth*	*Denormalized Bandwidth*
Accounts	1,888	1,888
Contacts	3,793	3,793
OrderProducts	25,971	5,534
Orders	35,305	3,309
Payments	53,808	548
Products	8	8
Users	18	18
Keywords	1,214	0
Total	122,004	15,097

2017, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

devices, and 4G mobile data coverage. Data storage and computation for the applications running on the mobile devices are accessed from distributed web services running in cloud computing environments. The software industry has responded by developing applications that require continuous connectivity, but this assumes safe computing environments and no user error. In reality, applications operate in a very different environment from the software developer's assumptions. Internet connectivity will come and go as the mobile device moves between cell and hotspot range. Malicious users may want to pollute the data collection process by replaying data packets or manipulating data in the original request. Application users may accidentally submit multiple times, forget to submit or submit incorrect data.

To solve the problem of providing mobile application access to data in the presence of a network partition, we developed a caching algorithm that persisted a local copy of the data on the mobile device as data was retrieved from the cloud. The data fetch operation attempts to call a web-service to retrieve fresh data when a user interacts with the application. If a network partition is discovered, then the locally cached data is used instead of web-service data. The reduced tuples required to store the data (shown in TABLE II) and the reduced bandwidth required to transfer the data (shown in TABLE III) allows more data to be cached locally. These reductions are important with mobile devices that have limited storage capacity and lower bandwidth available.

XI. FINE-GRAINED WEB-SERVICE CALLS

Many No-SQL databases handle tuple insertion via web-services. Each tuple create, read, update and delete (CRUD) operation requires a web-service call. When these databases are hosted in the cloud, the latency becomes an important bottleneck to minimize. In our testing, a back office web-service call can be fulfilled in one to three milliseconds on average. The same web-service call to a cloud provider requires between thirty and sixty milliseconds to fulfill on average.

The reduction of tuples of our design methodology assists in reducing the combined latency required in a typical software application. Over the workflow of a typical application the time the reduction is significant. The client reads data from the server to display each form. With our methodology, make fewer calls are made on the form setup, and there are also fewer calls made when the application updates data.

XII. LOSS OF CLOUD DATASTORE AVAILABILITY

The loss of availability to a system deployed in the cloud is a security risk that all enterprises must consider when migrating from a self- hosted solution to a vendor-hosted solution. Tuple-based licensing is a treat to availability or budget when overages occur. Tuple and data limit overage policies vary by cloud service provider, but, the threat of system availability loss should cause an organization to consider our modeling strategy to minimize the threat and the damage when an overflow occurs. There are three overage policies in use by cloud service providers:

- Bill for overage – The bill for overage policy allows continued use of the system after the overage occurs, but the organization is charged a fee based on the amount of overage. Often the fees are much higher per tuple then they are on the fixed price policies. The fixed price policy would be the pre-negotiated subscription fee the organization pays for their typical data. Salesforce uses the bill for overage policy for their cloud hosting data storage system. When an organization surpassed the tuple limits, a warning is displayed inside the administrative application for the platform to notify the user of the overage.
- Data insert or web-service call denial – This policy denies access to future writes, or system calls until the overage has been resolved. Salesforce uses this policy for web-service calls per day. In the out-of-the-box web services, a call is made per tuple affected. Once the daily limit is reached, the organization cannot invoke the web-service again until the next day. The possibly of no availability represents a large vulnerability for an organization and modeling must take this vulnerability into effect.
- Throttling – Throttling is a policy that slows access down once tuple limits have been reached. The throttling policy has been used by database systems for years in the back office, to ensure one thread does not overwhelm other concurrent threads on the server. In the cloud several data stores, such as MongoDB, throttle inserts per thread [22].

In all three of the licensing policies, a design methodology that reduced tuples will reduce the potential of downtime and over costs.

XIII. DISCUSSION

At first glance, the problem addressed by our research in this study appears to be a self-inflicted problem created by a licensing model used by several of the PAAS service providers. As we investigated the problem in depth, we quickly realized that traditional client-server database normalization models did not fit distributed cloud-based data store models. The normalization algorithms we teach and learn in database classes are tiered assuming an iterative approach to move down the tiered normalization level. The approach has a single goal of minimizing redundancy to alleviate update or deletion anomalies. The iterative approach includes a final phase in which the designer will denormalize the database structure to gain performance, concurrency or

2017, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

storage enhancements. In exchange for the better performance, concurrency or storage the designer or implementer is willing to accept the greater possibilities of deletion or update anomalies. Our initial problem was focused on the reduction of the storage requirements. The change was not aimed at less disk space usage, but less subscription cost for the data service consumed.

Over the course of the research project, we realized that the client-server model adds many other difficulties that can be eliminated in the denormalization phase. The client-server model has an innate preference for many smaller tables with a few attributes each over larger tables with many attributes stored per tuple. In practice, this has led to three major difficulties. The first challenge is experienced when an end user, of the application that generated the data, tries to gain access to the data stored in the database. The difficulty stems from the complexity of understanding the relationship between the many individual tables and the semantics of each table. Second, there is also complexity added to the development of integration systems that consume and write the data. With increased complexity comes increased development and maintenance costs. Finally, there is complexity in enforcing higher level constraints on data when the data is distributed across many smaller relationships. The need for correct data has increased as more and more data has become available for use in Management Information Systems (MIS) reports, data science prediction models and Executive Information Systems (EIS) visualizations. The best way to ensure correct data is to declare constraints as close to the database store as possible. The constraint declaration is simplified when the model is denormalized as we did in this project.

XIV. CONCLUSION

In this paper, we propose several algorithms for object denormalization when transforming an application domain object model to a data model used in a cloud PAAS data store. Our solution is based on navigating the relationships in a UML class diagram and horizontally compressing classes between multiple one-to-many relationships, aggregating relationships on anonymous relationships, using temporal offering patterns and rolling up one-to-many relationships.

The techniques used in our work met our goals of reduced tuple storage while increasing the usability of the data for the end users. Like the denormalization methods of the late 1990s, aimed at squeezing out a little more performance or a little more storage, our methods can be applied as individual strategies to save tuple space for a particular “use case.” For example, if a developer needed to model data passed between a mobile application and a cloud application, the denormalized model could be used to represent the data transferred. Likewise, if a developer had many one-to-many relationships and needed to reduce tuples, then our specific approach could be applied in that instance to reduce the storage requirements of the many sides of the relationship.

In this research, we studied a specific application domain related to humanities organizations. The algorithms can be applied to similar application domains that contain entity objects representing transactions and customers. Future work needs to test our algorithms in other application domains to ensure the work applies across different application domains.

REFERENCES

[1] A. Olmsted and G. Santhanakrishnan, "Cloud Data Denormalization of Anonymous Transactions," in *Cloud Computing 2016, The Seventh International Conference on Cloud Computing, GRIDs, and Virtualization*, Rome, Italy, March 20-24 2016, ISBN: 978-1-61208-460-2, pp 42-46.

[2] Brainsell blog, "Salesforce, SugarCRM and SalesLogix — Data Storage Costs Compared," 2016. [Online]. Available: https://www.zoho.com/creator/pricing-comparison.html. [Accessed 2017.5.5].

[3] Salesforce.com, inc, "Run your business better with Force.," 2006. [Online]. Available: http://www.salesforce.com/platform/products/force/?d=70130000000f27V&internal=true. [Accessed 2016.02.03].

[4] Zoho Corporation, "Creator Pricing Comparison," 2016. [Online]. Available: https://www.zoho.com/creator/pricing-comparison.html. [Accessed 2016.02.03].

[5] Zoho Corporation, "Compare Zoho CRM editions," 2016. [Online]. Available: https://www.zoho.com/crm/comparison.html. [Accessed 2016.02.03].

[6] C. J. Date, "Denormalization," in *Database Design and Relational Theory*, O'Reilly Media, 2012.

[7] G. L. Sanders and S. Shin, "Denormalization Effects on Performance of RDBMS," in *Proceedings of the 34th Hawaii International Conference on Systems Sciences*, 2001.

[8] J. D. Conley and R. P. Whitehurst, "Automatic and transparent denormalization support, wherein denormalization is achieved through appending of fields to base relations of a normalized database." USA Patent US5369761 A, 29 November 1994.

[9] A. Bisda , "Salesforce Denormalization Delivers New Power for Nurtures," DemandGen, 29 07 2014. [Online]. Available: http://www.demandgen.com/salesforce-denormalization-delivers-new-power-nurtures/. [Accessed 2015.5.5].

[10] D. Taber, Salesforce.com Secrets of Success: Best Practices for Growth and Profitability, Prentice Hall, 2013.

[11] A. Olmsted, "Fresh, Atomic, Consistent and Durable (FACD) Data Integration Guarantees," in *Software Engineering and Data Engineering, 2015 International Conference for*, San Diego, CA, 2015.

[12] Object Management Group, "Unified Modeling Language: Supersturcture," 05 02 2007. [Online]. Available: http://www.omg.org/spec/UML/2.1.1/. [Accessed 2015.5.5].

[13] C. L. Forgy, "Rete: A fast algorithm for the many pattern/many object pattern match problem," *Artificial Intelligence,* vol. 19, no. 1, p. 17–37, 1982.

2017, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

[14] A. Olmsted and G. Santhanakrishnan, "Cloud Data Denormalization of Anonymous Transactions," in *Cloud Computing 2016*, Rome, Italy, 2015.

[15] Object Management Group, "Unified Modeling Language: Supersturcture," 05 02 2007. [Online]. Available: http://www.omg.org/spec/UML/2.1.1/. [2015.5.5].

[16] Object Management Group, "OMG Formal Versions of XMI," 06 2015. [Online]. Available: http://www.omg.org/spec/XMI/. [Accessed 2015.11.11].

[17] Ceiton Technologies, "Introducing Workflow," [Online]. Available: http://ceiton.com/CMS/EN/workflow/introduction.html#Customization. [Accessed 2014.15.09].

[18] J. Ullman and J. Widom , A First Course in Database Systems, Pearson, 2007.

[19] "Certification Programs," United State Postal Service, [Online]. Available: https://www.usps.com/business/certification-programs.htm. [Accessed 2015.5.5].

[20] United States Postal Service, [Online]. Available: https://www.usps.com/business/move-update.htm. [Accessed 2015.5.5].

[21] CIO Arts, Inc, [Online]. Available: http://www.cioarts.org/. [Accessed 2015.5.5].

[22] Normally Pleasant Mixture, "mongo-throttle," [Online]. Available: https://www.npmjs.com/package/mongo-throttle. [Accessed 2017.01.27].

2017, © Copyright by authors, Published under agreement with IARIA - www.iaria.org