# Sparse Glider Datasets: A Case Study for NoSQL Databases

Michael Lindemuth and Chad Lembke
College of Marine Science Ocean Technology Group
University of South Florida
St. Petersburg, FL 33701
Email: mlindemu@usf.edu and clembke@usf.edu

***Abstract*—Multi-sensor platforms like buoys and gliders produce one or more readings per sensor on varying, discrete time frequencies. The resulting datasets are a matricies with rows containing readings from sensors that reported at a moment in time and NULL for missing readings from sensors that did not. Traditional Relational Database Management Systems (RDBMS) are already well suited for the dense matricies in which NULL values are infrequent. The efficiency of these systems deteriorates though as data becomes more sparse.**

**The University of South Florida College of Marine Science Ocean Technology Group (COT) operates four gliders. Each glider produces dynamic, different sparse datasets. Other data management solutions exist, but they are based on a RDBMS. COT has been investigating an alternative without using and RDBMS.**

**Glider Database Alternative with Mongo (GDAM) is a data management system for gliders built on the MongoDB NoSQL database engine. It is live in production at COT. GDAM is a collection of scripts which parse, process and store real-time glider datasets. Data is parsed as soon as it is transmitted via satellite to our shore-based servers. The system has been tested during two Slocum G1 glider deployments in September and October of 2012. Archival datasets dating back to March of 2009 have also been uploaded into this system. Records are indexed by time, GPS, and depth with the ability to add more indexes as necessary.**

**The paper outlines dataset problems identified using data from COT glider operations in 2012. These problems inform a discussion of design decisions and possible options considering both RDBMS and NoSQL systems. The paper concludes by discussing the current implementation of GDAM.**

## I. Introduction

Field deployed multi-sensor platforms introduce a unique challenge for database management systems. Due to factors like inclimate weather, cloud cover, or curious fauna sensors may fail. Throughout the life cycle of the platform sensors may be upgraded or added. Platform managers may enable or disable sensors due to funding or power constraints. As a platform is more dynamic, creating a schema for a database management system becomes more difficult.

Some platforms are so dynamic that their datasets can be classified as sparse. Sparse datasets are datasets where given a set of fields, the majority of fields are not filled or NULL values. Sparse datasets exist when platforms have failing sensors, sensors with different sampling frequencies, or sensors with event-based sampling rather than scheduled sampling.

Relational Database Management Systems (RDBMS) struggle in handling sparse datasets. The initial design of an RDBMS requires a schema. This schema can be adjusted as needed, but it still must account for all possible values. Schemas that account for sparse datasets can be space inefficient due to NULLs populating most of the database or require complex queries due to a non-traditional design.

NoSQL offers a schemaless alternative. NoSQL databases are designed to handle the high volume of and adjust to the dynamic nature of web-based data. Most do not require schema design. Without a schema, NoSQL databases can efficiently store records of varying sizes without wasting space storing NULL values.

Glider Database Alternative with Mongo (GDAM) is a system developed at the University of South Florida College of Marine Science Ocean Technology Group (COT). It is built on MongoDB, a NoSQL database. Scripts to parse and store data from Teledyne Webb Gliders have been developed. These scripts have been deployed in production and used during deployment of gliders.

The paper outlines the design decisions and implementation of GDAM. It begins in section II by demonstrating how sparse glider datasets can be. Next, section III outlines why sparse data is difficult for RDBMS. Section IV discusses the current state of NoSQL systems and tradeoffs between them and relational database systems. Section V outlines the implementation of GDAM. Lastly, section VI discusses the state of GDAM and how it is an improvement over RDBMS solutions for glider data.

## II. Gliders

Autonomous underwater profiling gliders have been in development by a number of research groups for over two decades. They are tailored to efficiently collect water column data, from the surface to depth, over moderate temporal and spatial scales with minimal time investment by the operator. And over the past decade, their use has grown tremendously, resulting in thousands of deployments by a number of users. The gliders traverse the water column using buoyancy generated propulsion, allowing them to repeatedly cycle the water column in a saw-tooth pattern while transiting hundreds of kilometers over a period of weeks. They periodically surface to

978-0-933957-40-4 

communicate via satellite communications, allowing for adaptive missions. They can carry sensor packages that measure a multitude of sensors.

### A. Operations

The University Of South Florida College Of Marine Science's fleet of Slocum gliders, produced by Teledyne Webb Research and operated by the College's Ocean Technology Group, enhance ocean observing efforts. To date, all of the USF glider deployments have been in the eastern Gulf of Mexico between Pensacola and Naples, FL. Since 2009, there have been over 20 missions, benefiting a variety of scientific missions such as monitoring red grouper [1], understanding red tide [2]–[5], validating circulation models [6], and monitoring for dispersed oil during the Deepwater Horizon oil spill. It is hoped that these efforts continue and expand in support of coastal observations efforts.

USF's five gliders are each equipped with a CTD, and most have enhanced optics and acoustic payloads, including fluorometers, dissolved oxygen sensors, radiance / irradiance sensors, and passive acoustic recorders. Four of the gliders are rated for 200m and the fifth is rated for 1000m.

### B. Datasets

While they collect thousands of parameters nearly continuously, the user receives this data via several methods. Slocum gliders can communicate data through an RF link or a satellite link, each with their own bandwidths. They store data on internal memory cards for collection following a deployment. There are varying levels of data packets for the user to customize for their operation. These vary from very sparse and limited sensor value files optimized for satellite communication, to complete data sets of every sensor recorded. Additionally, different data types are stored on different storage disks maintained by the gliders independent flight and science computers.

Merging data stored on the two independent computers along time columns results in a very large sparse matrix. Each row in the matrix represents a record in time where a subset of sensors is sampled. Each column is a sensor reading type (e.g., depth, conductivity, altimeter). Sensor sampling frequencies and output can be changed by glider operators before or during a deployment.

One USF glider deployment in October, 2012 produced a matrix with 698,313 records (rows) and 1,890 unique sensor fields (columns). Figure 1 shows the frequency distribution of the number of fields[1] in each of the records. 34.1% of the records contain only 6 out of 1,890 possible fields. The next most common number of fields is 196 which accounts for 10.4% of records. The most filled records populate 1,836 fields, but account for 6 records total. The median number of fields in a record is 191.

The data from this last glider deployment is not an outlier. Every deployment dataset is like it and may have a different number of unique sensor fields. A rigid relational database schema is a difficult system to maintain in this environment.

[1]300-1,890 omitted on the y-axis due to negligible frequencies

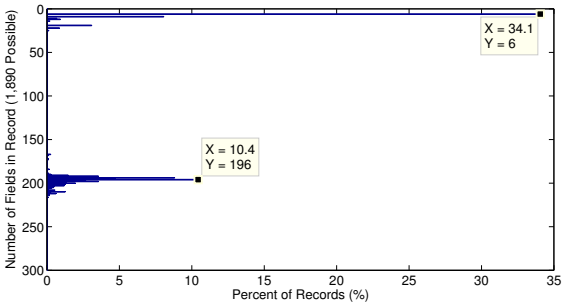


Fig. 1. Glider Record Size Frequency Over One Entire Deployment in 2012

TABLE I
COMMON RANGE QUERIES

| Query | Parameters | Request |
|---|---|---|
| Time | Start,End | Return readings between start and end timestamps. |
| Sensor | Sensor Output Type,Min,Max | Return readings taken at the same time a sensor output type (e.g., depth) was between min and max values. |
| Spatial | GPS Bounding Rectangle | Return readings taken inside a specific geographic area. |

### C. Common Queries

Glider datasets are of interest to a diverse set of users which can include academic researchers, policy makers, modelers, fisherman, and the general public. Despite differences in their objectives, most users can be served by three main queries: time, sensor, and spatial range queries. The ideal database system would be able to handle all three types of queries in a time efficient manner.

Table I provides a list of required parameters and short request examples for each query type. All range queries return a subset of glider data as a sparse matrix where every column is a sensor output type and every row is the value of the sensor output at that instant in time. The returned sparse matrix can be further reduced by allowing users to specify fields of interest.

No relational based glider database system could handle all three types of queries. An alternative is necessary.

## III. RELATIONAL DATABASE MANAGEMENT OF SPARSE DATASETS

Commonly used Relational Database Management Systems (RDBMS) include MySQL [7], PostgreSQL [8], Microsoft SQL Server [9], and Oracle Database [10]. Every RDBMS requires that data be structured in a relational schema consisting of tables with pre-defined columns that each have a name and data type such as text, integer, or float. Tables store objects with columns that represent object attributes (e.g, name, description, cost) and rows that represent new instances of that object type. Object instances each have a primary

key which is usually an integer. Objects stored in one table can reference objects in another by storing the primary key value rather than a copy of the other object. This reference creates a relation between tables. Sparse data performs poorly in standard relational schema. A relational table storing just data from the October, 2012 deployment would need to have at least 1,890 columns and efficiently store NULL values.

### A. Column Limits

Most relational databases cannot store a large number of columns. MySQL[2], Oracle and SQL Server standard table types are limited to around 1,000 columns [11]–[13]. PostgreSQL, on the other hand, only has a 1.6TB per row limit which, in most cases, can store the glider deployment table [14].

Agrawal et. al. propose a vertical schema to circumvent the column limitation of any relational databases [15]. Vertical schemas store 3-ary tuples in a single table that consist of an object id, key, and value. Each row in the table references unique objects by id, an attribute of that object, and the value of that attribute. Object NULL attributes are not stored. Values are all string types (VARCHAR). Most glider output is a float or integer. Storing numbers as strings rather than binary is space inefficient for both tables and indexes.

A variant of this idea called a binary representation splits each attribute out to separate tables containing object id and value pairs where the value can be typed correctly and efficiently stored [15]. A glider operations schema similar to a binary representation does exist and is used in production. Both vertical and binary representations descrease query simplicity and performance [16]. The similar glider operations schema suffers from these same problems.

### B. NULL Value Storage

Relational database schemas often include fields that can either contain a value or a NULL. RDBMS deal well with a few NULLS that may be present in a dense dataset. They are space inefficient for sparse datasets though especially if fixed size types like integer or floats are present in rows. In the case of the example glider deployment table, 90% of columns are NULL. Beckmann et. al. describe a more efficient row storage method called interpreted storage that does not use storage on NULLs [16]. Oracle and SQL Server both offer a wide table type that can efficiently store NULL values. Unfortunately, neither are open source or freely available.

## IV. Document-oriented NoSQL

NoSQL refers to a wide set of database systems that depart from some fundamentals of RDBMS. For simplicity, this section focuses on Document-oriented NoSQL databases but touches on topics relevant to other types. Cattell et. al. gives a broad overview of what NoSQL stores are available along with a comparison [17]. The majority of NoSQL database developers relax RDBMS requirements and focus on flexibility and scalability to handle the dynamic and high volume of web based data [17], [18].

[2]Using the default InnoDB table storage engine

### A. Flexible Record Storage

All NoSQL databases offer flexible storage. Not every type to be stored must have a column defined. Instead, values can consist of any number of fields. Document-oriented NoSQL databases specifically store data points in documents that each have a unique identifier (key) assigned and their own data structure (value) with one or more fields. Relations between documents can be made by referencing the keys of other documents. Care must be taken in using relations with NoSQL databases because, unlike all RDBMS, not all NoSQL databases guarantee atomicity and isolation.

### B. ACID vs. BASE

RDBMS guarantee atomicity, consistency, isolation, and durability (ACID) properties on operations or sets of operations marked by a transaction block [19]. Each term makes the following guarantees:

- *Atomicity* guarantees that all operations in a set occur or none occur; Partial transaction execution is not allowed.
- *Consistency* guarantees that each transaction preserves the state of the database.
- *Isolation* guarantees that one transaction cannot affect another.
- *Durability* guarantees that in the event of a system failure all completed transactions will persist in the database.

ACID properties are absolutely necessary in databases where real-time accounting is important such as stock exchanges or banks.

Most NoSQL systems do not guarantee all ACID properties. They are better described by consistency, availability, and partition tolerance (CAP) properties. As originally proposed by Brewer, these properties are defined as [20]:

- *Consistency* guarantees that all database clients see the same set of data.
- *Availability* guarantees that the database is able to handle a high volume of operations.
- *Partition* tolerance guarantees that database cluster clients will still be able to read and write to the database when nodes fail. When a node comes back online, it receives all writes performed while it was offline.

Unlike RDBMS and adherence to ACID, NoSQL systems decide between either strong consistency or high availability [21]. To offer strong consistency a system must go offline for short intervals to synchronize. To offer high availability, the system will continually be in an 'eventually consistent' state.

Eventually consistent system guarantees are best described by yet another acronym: BASE [22]. BASE stands for 'Basically Available Soft state Eventually consistent'. It guarantees that the system - barring some catastrophic failure - will be available, but all clients accessing different database nodes may not see the same state of data. After data is written or updated on one node, the change it will eventually be reflected on the other nodes. BASE offers a good tradeoff for database

management systems that do not require a real-time consistent state.

### C. Scalability

Relaxing ACID in favor of BASE allows many NoSQL systems to scale well across multiple servers (i.e., horizontal scaling). RDBMS can scale across multiple servers, but they require complex schemas that include table partitioning schemes. Most RDBMS focus on scaling by moving to a better server (i.e., vertical scaling) rather than to a cluster of average servers.

### D. Performance

Flexibility and scalability usually come at a cost. A recent comparison on a single server show that MongoDB, a document-oriented database, is faster than SQL server at reading and writing data points [23]. As databases scale horizontally though, the overhead of managing schemaless data becomes apparent. Two others studies between horizontally scaled RDBMS and NoSQL servers show how the former can still be faster [24], [25]. All three benchmarks use dense datasets with very few NULL values. No benchmarks could be found for sparse datasets.

## V. Implementation

Glider Database Alternative with Mongo is a glider data management system that can be used in tandem or in place of the manufacturer's data solution. It uses 10Gen's MongoDB, a NoSQL Document-oriented database, as it's back-end. Users can access the system via a web front-end. Details about the MongoDB back-end and web front-end follow.

### A. MongoDB Back-end

MongoDB is an open-source Document-oriented NoSQL database. It requires no schema to begin handling data, can scale horizontally across up to 12 heterogeneous machines, and can store up to 16MB in a document [26]. Documents are stored in an efficient Binary Javascript Object Notation (BSON) format.

BSON stores each field to it's closest binary representation rather than storing everything as a string. BSON document sizes can vary. Only fields with values are stored in BSON documents. No NULL-valued fields are stored. Another great feature of BSON is that it quickly translates to the popular JSON web format used in many RESTful APIs. BSON documents are assembled into logical groups called collections.

Collections usually contain similar documents, but, unlike RDBMS tables, do not require it. Similar to RDBMS tables though, all query, insert, index, and update operations are performed on collections. Collection joining is not supported by MongoDB. Sets of collections can be separated into individual databases. Proper planning and organization of the database, collection, and document storage can make queries simpler and faster.

GDAM organizes collections based on glider and upload type. Upload type allows the subset of data uploaded while the glider is in the field to be separated from the much higher resolution data that is processed from a retrieved glider. Figure 2 shows this split between glider, and upload type. Scripts in the system are run as soon as all data is transmitted to shore. These scripts parse and merge glider flight and science data based on timestamps. The merged data is then inserted into the proper MongoDB collection.

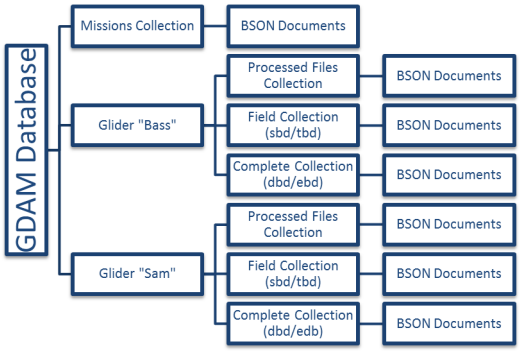


Fig. 2. GDAM Mongo Collection Organization Based on Glider and Upload Type

Indexes are built on each collection glider field and complete collections to efficiently handle time, sensor, and spatial queries as discussed in II-C. The only field assumed to be in every document of the collections is 'timestamp'. A unique index over timestamp prevents duplicate record insertion and speeds time range queries. A sparse index over depth speeds up depth range queries. Other sparse index fields can be added as needed on individual collections. A sparse, spatial index over GPS coordinates greatly speeds up any spatial queries.

Sparse indexes are used on depth and GPS because not all records have these fields. A drawback to having to use sparse indexes is that they require two queries to retrieve all the relevant data. The first query is on the sparse indexed field to find the time minimum and maximum. These time extents are then used in a second query to retrieve the set of documents in that time range. Currently, the depth and spatial query methods only works for a single deployment at a time, but work is being done on extending it to retrieve data across multiple deployments.

### B. Web Front-end

GDAM implements a read-only RESTful API. Developers can query for glider data by time, sensor, or spatial area by generating and sending a GET request to a web-facing server. The server performs the query on the GDAM MongoDB and returns results as a JSON document by default. CSV and KML formats are available if requested in the GET URL.

COT uses the GDAM API to display glider data on it's Centralized Remote Observations Website (CROW) [27]. CROW uses the Google Charts and Earth API to display gilder deployment paths and data since the last surfacing. To begin users specify a time range in UTC. As the user enters their desired time range, the site generates a GDAM API request and starts an AJAX request for KML data. When the AJAX

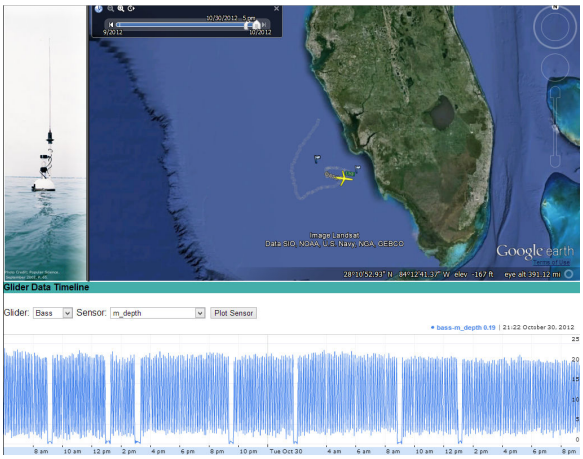


Fig. 3. USF CROW Web Interface

request is successful, a path is shown in the Google Earth panel with a time range control in the top left. Any adjustments to the time range in Google Earth will also adjust the time axis of the data plot at the bottom.

## VI. Conclusion

GDAM provides a novel alternative to traditional RDBMS. The lack of a schema and dynamic nature allows it to map storage more closely to the real values without the inefficiency of storing empty or NULL values. As demand or storage needs increase, it can be scaled horizontally across heterogeneous machines rather than requiring a machine upgrade. The RESTful web-based API allows other users to easily and efficiently perform time, value, and spatial range queries to retrieve subsets of all glider deployment data.

The authors believe a similar solution may be much more difficult to achieve with RDBMS than NoSQL. Though more tried and proven, RDBMS do not accommodate sparse datasets efficiently. Research does exist on trying to improve RDBMS and the authors will continue to monitor for any improvements. For now though, in our opinion, NoSQL offers a better alternative for dynamic, sparse glider datasets.

## Acknowledgment

The authors would like to thank Drs. Chuanmin Hu, Frank Muller-Karger, and Bob Weisberg for their support of USF glider operations. Work on this paper was funded as part of Gulf of Mexico Coastal Ocean Observing System (GCOOS) grant #99-S120305 'West Florida Shelf Glider Deployments and Data Dissemination for GCOOS-RA'.

## References

[1] C. Wall, C. Lembke, and D. Mann, "Shelf-scale mapping of sound production by fishes in the eastern gulf of mexico, using autonomous glider technology," *Marine Ecology Progress Series*, vol. 449, pp. 55–64, 2012.

[2] D. English, C. Hu, C. Lembke, R. Weisberg, D. Edwards, L. Lorenzoni, G. Gonzalez, and F. Muller-Karger, "Observing the 3-dimensional distribution of bio-optical properties of west florida shelf waters using gliders and autonomous platforms," in *OCEANS 2009, MTS/IEEE Biloxi - Marine Technology for Our Future: Global and Local Challenges*, 2009, pp. 1–7.

[3] A. Jochens, M. Howard, L. Campbell, R. Mullins-Perry, G. Kirkpatrick, B. Kirkpatrick, C. Simoniello, C. Hu, R. Weisberg, C. Lembke, A. Corcoran, J. Ivey, and S. Wolfe, "Integrating observing systems to benefit stakeholders: A case study in the gulf of mexico," in *Oceans, 2012*, 2012, pp. 1–4.

[4] J. Zhao, C. Hu, J. Lenes, R. Weisberg, C. Lembke, D. English, J. Wolny, L. Zheng, J. Walsh, and G. Kirkpatrick, "Three-dimensional structure or a karenia brevis bloom: Observations from gliders, satellites, and field measurements," *Harmful Algae*, 2013, accepted for publication July 2013.

[5] R. H. Weisberg, L. Zheng, Y. Liu, C. Lembke, J. Lenes, and J. Walsh, "Why a red tide was not observed on the west florida continental shelf in 2010," *Harmful Algae*, 2013, in press.

[6] R. H. Weisberg, A. Barth, A. Alvera-Azcrate, and L. Zheng, "A coordinated coastal ocean observing and modeling system for the west florida continental shelf," *Harmful Algae*, vol. 8, no. 4, pp. 585 – 597, 2009, understanding the causes and impacts of the Florida Red Tide and improving management and response. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S156898830800156X

[7] (2013, Jul.) Mysql :: The world's most popular open source database. [Online]. Available: http://www.mysql.com

[8] (2013, Jul.) Postgresql: The world's most advanced open source database. [Online]. Available: http://www.postgresql.org

[9] (2013, Jul.) Business intelligence — database management — data warehousing — microsoft sql server. [Online]. Available: http://www.microsoft.com/en-us/sqlserver/default.aspx

[10] (2013, Jul.) Oracle — hardware and software, engineered to work together. [Online]. Available: http://www.oracle.com

[11] (2013, Jul.) Mysql::mysql 5.7 reference manual::d.10.4 limits on table column count and row size. [Online]. Available: http://dev.mysql.com/doc/refman/5.7/en/column-count-limit.html

[12] (2013, Jul.) Logical database limits. [Online]. Available: http://docs.oracle.com/cd/E16655_01/server.121/e17615/refrn0043.htm#i288032

[13] (2013, Jul.) Maximum capacity specification for sql server. [Online]. Available: http://msdn.microsoft.com/en-us/library/ms143432.aspx

[14] (2013, Jul.) Postgresql: About. [Online]. Available: http://www.postgresql.org/about/

[15] R. Agrawal, A. Somani, and Y. Xu, "Storage and querying of e-commerce data," in *VLDB*, vol. 1, 2001, pp. 149–158.

[16] J. Beckmann, A. Halverson, R. Krishnamurthy, and J. Naughton, "Extending rdbmss to support sparse datasets using an interpreted attribute storage format," in *Data Engineering, 2006. ICDE '06. Proceedings of the 22nd International Conference on*, 2006, pp. 58–58.

[17] R. Cattell, "Scalable sql and nosql data stores," *SIGMOD Rec.*, vol. 39, no. 4, pp. 12–27, May 2011. [Online]. Available: http://doi.acm.org/10.1145/1978915.1978919

[18] J. Pokorny, "Nosql databases: A step to database scalability in web environment," in *Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services*, ser. iiWAS '11. New York, NY, USA: ACM, 2011, pp. 278–283. [Online]. Available: http://doi.acm.org/10.1145/2095536.2095583

[19] R. Ramakrishnan and J. Gehrke, *Database Management Systems*, 3rd ed. New York, NY, USA: McGraw-Hill, Inc., 2003.

[20] E. A. Brewer, "Towards robust distributed systems (abstract)," in *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, ser. PODC '00. New York, NY, USA: ACM, 2000, pp. 7–. [Online]. Available: http://doi.acm.org/10.1145/343477.343502

[21] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *SIGACT News*, vol. 33, no. 2, pp. 51–59, Jun. 2002. [Online]. Available: http://doi.acm.org/10.1145/564585.564601

[22] D. Pritchett, "Base: An acid alternative," *Queue*, vol. 6, no. 3, pp. 48–55, May 2008. [Online]. Available: http://doi.acm.org/10.1145/1394127.1394128

[23] Z. Parker, S. Poe, and S. V. Vrbsky, "Comparing nosql mongodb to an sql db," in *Proceedings of the 51st ACM Southeast Conference*, ser. ACMSE '13. New York, NY, USA: ACM, 2013, pp. 5:1–5:6. [Online]. Available: http://doi.acm.org/10.1145/2498328.2500047

[24] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 143–154. [Online]. Available: http://doi.acm.org/10.1145/1807128.1807152

[25] A. Floratou, N. Teletia, D. J. DeWitt, J. M. Patel, and D. Zhang, "Can the elephants handle the nosql onslaught?" *Proc. VLDB Endow.*, vol. 5, no. 12, pp. 1712–1723, Aug. 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2367502.2367511

[26] (2013, Jul.) Mongodb. [Online]. Available: http://docs.mongodb.org/manual/reference/limits/

[27] (2013, Jul.) Centralized remote observations website. [Online]. Available: http://ooma.marine.usf.edu/CROW/