UNIVERSITY OF SOUTH FLORIDA
COLLEGE OF BUSINESS

[DRAFT:] 04/01/2012

DONALD J. BERNDT, RICARDO LASA, JAMES MCCART

SiteWit Corporation: SQL or NoSQL that is the Question[1]

Is the time right? The time is right! And here is why…

Ricardo Lasa, CEO and co-founder of SiteWit Corporation, was always chastising his technical team that the "biggest risk facing the company is the engine." SiteWit provides cross-platform services aimed at helping small (or even medium-sized) business customers effectively advertise on search engines like Google (AdWords) and Bing (adCenter), as well as other online social networking or display advertising venues. Essentially, SiteWit is a Web analytics company that tracks all the detailed organic and paid advertising traffic on client websites. SiteWit uses this very detailed data to deliver software-as-a-service (SaaS) products that handle a variety of tasks from automated keyword bidding to campaign optimization. These products rely on a foundation of website analytic data warehousing and automated data mining, so data quality is of paramount concern.

Lasa and his team faced a critical technology challenge in scaling the core database systems to meet rapidly escalating data volumes. Should he stick with well-known relational database technologies? His core team was well versed in the Microsoft technology stack and had worked together for more than a decade on software-as-a-service (SaaS) applications. Or should he re-implement core components in newer, highly distributed NoSQL databases in search of competitive advantages? So, the decision could be summarized as follows: SQL or NoSQL that is the question.

1. **Do nothing.** SiteWit Corporation is a lean startup with limited resources. Do we really need to add new technologies and more uncertainty at this stage?
2. **Proceed cautiously with NoSQL technology through limited experiments.** It might be reasonable to pick some component that could be implemented using NoSQL technology to gain experience and validate the technology.
3. **Develop a new product, alone or through a partnership, that makes use of NoSQL technologies.** A couple of potential SiteWit partners are experimenting with or even already resting firmly on NoSQL technologies, so one strategy might be to learn through collaboration.
4. **Take a leap of faith.** Again, SiteWit is an early stage company facing plenty of risk factors. Adding a few more for an important competitive advantage may be a reasonable tradeoff.

Whatever decision he reached, scaling the core database technology was an immediate need. New customers were arriving daily in response to online advertising campaigns. It would not get any easier.

Database Technologies

The heart of SiteWit's technological challenge revolved around the core technologies for managing its data. A database is an application that manages structured collections of data. Databases can be used either independently, or in conjunction with other programmed applications. Since the 1980s, databases had been dominated by a particular approach: the relational database. SiteWit was already up and running with a robust relational architecture using cloud-based infrastructure services. While relational database vendors had added many features to support very large databases, they struggled to achieve the type of scale and performance required by web applications. Companies like Google and Amazon had therefore developed whole new infrastructures to support their businesses. More importantly, some other Web analytic startups had embraced next-generation distributed database technologies that grew out of earlier efforts by Google and other "big data" pioneers. To better understand the situation, it is useful to examine how technology has evolved.

Early Database Approaches

Databases first began to appear in the late 1950s and early 1960s, spurred by two technological factors: the increasing reliability of computer processors and the expansion of secondary storage capacity in the form of tapes and disk drives. Early "databases" tended to be sequential or random access files that could be searched or processed by any program that knew the precise internal file structure. Eventually, standalone products that were application-independent evolved. Often these used a hierarchical data organization, whit data being stored in a category-subcategory-subsubcategory-etc… arrangement that, from a logical standpoint, looked like a tree. IBM's IMS system used this organization and, even today, the approach is used for some systems, such as the MS Windows Registry.

Many applications, such as the early airline reservation system, delivered high performance on hardware that would be considered primitive by today's standards using these early database designs. Using these designs, however, demanded significant concessions with respect to flexibility. Essentially, you had to know all the intended to use your data before designing the data organization. If a different type of search or transaction was later desired, it was likely to prove highly inefficient, or even impossible, to implement on an existing database. These early database models also required substantial skills from their users. They required each query to be written like a computer program. These languages used to form these queries were **procedural** languages, meaning that the details of how to produce the answer had to be specified in an unambiguous computer algorithm. While good database performance could be achieved, the effort involved was significant and demanded a highly skilled work force. These factors all drove demand for a more adaptable approach to data management.

Relational Databases: A Classic Success Story

E. J. Codd first proposed relational databases and the underlying theories in a 1970 paper [Codd 1970]. In a subsequent Turing Award lecture titled "Relational Database: A Practical Foundation for Productivity," he highlighted the overall objective: to improve the productivity of database programmers [Codd 1982]. The relational database model was a radical departure that rested upon a few powerful set-theoretic operations that combine separate data tables (or relations) to produce an answer set. The queries are specified using relational algebra or more commonly the standards-based Structured Query Language (SQL). SQL allows a database user to express his or her query in a **declarative** form, without any detailed programmatic instructions. That is, the form of the results and inputs are specified, without any concern for how the results will be computed. So database users no longer needed to be skilled programmers or spend their time writing complicated query code — hence the gain in productivity.

The problem with early relational databases was that, while technically elegant, they were horribly inefficient. Queries would simply take a long time to produce answers. This lackluster performance

encouraged a widespread research effort on database optimization techniques that exploit heuristic rules, indexing structures, and statistics to create more efficient query execution plans. These advanced approaches substituted computer-based analysis for handcrafted query design in order to generate the procedural steps necessary to efficiently answer the query. Over time, these research efforts were quite successful, leading to a classic laboratory to marketplace transfer of technology, with companies such as Oracle, IBM, and later Microsoft offering robust relational database products.

In 2012, over thirty years after their commercial introduction, relational database systems (SQL technologies) remained at the core of virtually all corporate data infrastructures and powered most day-to-day operational processes, from accounting systems to contact management on cell phones to comprehensive ERP systems. There was some indication, however, that this situation might ripe for change. In the emerging era of "big data" with an increasingly important role of data analytics, traditional RDMS approaches were again being performance challenged by the problems of *scaling* and *distributed transaction processing*. In these areas, alternative database (NoSQL) technologies offered some interesting advantages.

Scaling Database Systems

There are two fundamental approaches to scaling database systems: **vertical scaling** and **horizontal scaling** [Prichett 2008]. Vertical scaling is the more straightforward strategy, relying on increasingly powerful computing infrastructures to meet demand. Of course, this strategy can become expensive as progressively more exotic machines lock you into pricey vendors. This path may also take you into database server clustering, with an extra layer of software complexity allowing multiple machines to focus on a single database.

Horizontal scaling takes a different approach, partitioning the data across multiple databases. While this approach is more complex, there are gains in flexibility and the potential to scale for big data applications. Horizontal scaling can be achieved along two dimensions. One dimension involves grouping the data by function and then spreading the functions across multiple databases. The second dimension involves splitting the data within a function across multiple databases (or *shards*). NoSQL platforms often offered built-in support for database sharding as a scaling strategy.

Distributed Transaction Processing

The dominant commercial relational database engines all provide sophisticated transaction processing capabilities. A **transaction** is a user-defined unit of work that typically comprises multiple database operations, such as individual queries or update statements. For instance, a simple human resource function may entail reading from several database tables and writing new information to other tables as necessary, all of which should be considered a single transaction. Distributed databases add another layer of complexity to this process, as the required updates may involve multiple computers at different locations. For many years, transaction processing had been driven by the so-called ACID properties. A new philosophy, known by the acronym BASE, was now emerging, most widely supported on NoSQL platforms.

ACID Properties

Relational databases provide important transaction processing guarantees, including atomicity, consistency, isolation, and durability (the so-called ACID properties). These fundamental ACID properties can be briefly defined as follows.

- Atomicity – All of the statements within a transaction are completed (or none are performed); no partially executed transactions.
- Consistency – The database is left in a consistent state after a transaction is executed.

- Isolation – A transaction is executed as if it is the only unit of work being processed by the database.
- Durability – Once completed (or committed), a transaction will never be reversed.

While industrial quality relational database systems often provide additional functions to speed up transaction processing, there are architectural limits on the degree of parallelism possible. Therefore, very large Internet-scale applications have sought out other big data solutions.

In order to scale traditional relational databases, a few additional servers can be "clustered" to function as a single database engine, with protocols to keep the independent memory areas synchronized (the cache coherency problem). However, to scale beyond these special-purpose clustered solutions requires a truly distributed collection of database engines, with data spread across all the systems. What happens if we try to process a transaction across database boundaries? In this case, a higher-level protocol must be used to make sure the pieces of a transaction are handled appropriately within each database, still guaranteeing the ACID properties above. For instance, most relational databases make use of an approach called the two-phase commit (2PC) protocol, even though it has some issues with regard to failures of participating distributed databases. The two phases consist of a voting phase in which all participants must indicate (to the query coordinator) that the assigned portion of a transaction can be completed, followed by a commit phase in which the query coordinator issues a commit or abort (depending on the previous votes). All participants must send a "yes" vote for a transaction to reach a commit point. Of course, this ensures consistency after each transaction, though at the cost of a fairly expensive protocol that requires the ACID properties to be met. How can we tradeoff consistency to gain availability and enhanced performance in big data environments?

Basic Availability Soft-State Eventual Consistency (BASE)

An alternative to the traditional ACID properties is captured by the acronym BASE, for basic availability soft-state eventual consistency. Rather than stark opposites, these different models of transaction processing anchor a continuum. Software architects can choose points along this continuum that best suite systems and associated business models. A BASE approach removes the strict focus on consistency after every detailed transaction in favor of achieving "eventual consistency" within a reasonable timeframe. In other words, approximations are fine and need not be based on every single data item.

Think of accounting systems, which keep track of transactions as a business runs, but lag reality until the books are formally "closed" and reconciled for a given period. During much of the time, these accounting systems are used to produce management reports that are reasonable approximations rather than completed financial reports.

Emerging Developments in NoSQL

Just as RDMS technology emerged out of a mix of theory and practice, NoSQL approaches were rapidly evolving from the same two influences. On the theory side, the CAP Theorem was making clear the tradeoffs that needed to be made when distributed "big data" was involved. On the practice side, companies, such as Google, were applying NoSQL approaches to great effect and tools for constructing NoSQL databases, such as MongoDB, were being applied for commercial purposes.

CAP Theorem

Eric Brewer at the 2000 Symposium on Principles of Distributed Computing (PODC) first proposed the CAP theorem as a conjecture. There have been many different discussions of his conjecture from both academic and practitioner perspectives, especially as it relates to NoSQL databases. These discussions almost always begin with a re-cap of CAP, highlighting three desirable properties of distributed systems: **consistency**, **availability**, and tolerance of network **partitions** (hence CAP). The conjecture is that distributed systems can embrace only two of the three properties, yielding three combinations that

describe the underlying tradeoffs: consistent and available (CA), consistent and partition tolerant (CP), and available and partition tolerant (AP) as in Exhibit 1. Two MIT researchers, Seth and Gilbert, published a proof of the conjecture establishing it as a theorem, though in a somewhat restricted form [Seth and Gilbert 2002]. It turns out, however, that the design tradeoffs pursued in many NoSQL databases were somewhat subtler than a straightforward choice between consistency and availability.

Taking a somewhat simple perspective of these highly complex discussions, scaling based on any type of distributed system involves partitioning the data across machine boundaries, and therefore requires partition tolerance (P). Thus, highly scalable systems are typically trading off consistency or availability, giving us the CP or AP categories shown in Exhibit 1 (with some associated NoSQL systems listed). The consistent-available (CA) systems include the traditional relational database management systems (RDMSs), including offerings from companies such as Oracle and Microsoft (e.g., the SQL Server engine being used by SiteWit).

Google's BigTable

Google's BigTable provided an early and excellent example of these new highly distributed database systems [Chang et al. 2006]. Because of its immense scale and innovative philosophy, Google traditionally relied on custom-built infrastructure, including innovative data centers, inexpensive servers, and big data toolkits. BigTable was one of the first generation "Internet scale" highly distributed database systems. BigTable provided a simple data model for storing structured data across a large collection of commodity servers, thereby providing an efficient and cost effective data store at web scale. By 2006, BigTable was the data store for many recognizable Google projects, such as Google Analytics, Google Finance, Orkut, Writely, and Google Earth. As described by several of the developers, "BigTable has achieved several goals: wide applicability, scalability, high performance, and high availability." What is not to like? In fact, BigTable had been directly re-incarnated in projects such as Apache Cassandra (cassandra.apache.org). These enabled many start-up companies to make use of big data on a solid foundation. For example, Netflix, Twitter, Constant Contact, Digg, and CloudKick were all Cassandra users.

BigTable employed a simple data model and deliberately avoided providing complicated features such as general transaction management. Again echoing the developers, "The most important lesson we learned is the value of simple designs." BigTable did not implement a full relational database model, but went beyond bare key-value pairs to provide a data store based on row keys, column keys (and column families), with timestamps to support versioning. Client applications interacted directly with BigTable via a lean application programming interface (API), while BigTable itself relied on other building blocks such as the Google File System [Ghemawat et al. 2003]. BigTable partitioned a table into ranges of rows or "tablets," which were the fundamental units for distributing data. BigTable then relied on a single master server and many tablet servers (perhaps thousands) to distribute and manipulate very large tables, with the bulk of all communications going directly through the bank of tablet servers. The early performance benchmarks were impressive, but the cost and sophistication required to create BigTable and the other building blocks necessary for the first wave of Web scale data all but eliminated small business entrants. Fortunately, the next wave of big data entrepreneurs had access to open source and commercial implementations of BigTable-like toolkits!

NoSQL Tools and MongoDB

By 2012, database systems that targeted big data applications were appearing at a rapid rate. Most of these involved spreading both data and processing across many machines, so that much larger amounts of computing power could be effectively harnessed. Nevertheless, bringing together large numbers of distributed machines for highly targeted tasks still involved challenges in communication, coordination, and even fault tolerance [Gelernter and Carriero 1992]. Intermediate results often needed to be communicated between machines, certain processing steps could be dependent on each other (requiring a

specific sequencing), and any machine might fail at the worst moment. All these challenges were made more difficult in the context of running non-stop (24-by-7) big data applications.

MongoDB (the "mongo" derives from "hu***mongo***us") was an example of an open source database that was specifically designed to handle big data. Its first release was in 2009 and its "production-ready" version first appeared in 2011. Already, however, it boasted a number of well-known users, including *MTV*, *craigslist* and *FourSquare*.

Elliot Horowitz, the CTO of *10gen*—the original developer of the product—described the philosophy behind MongoDB design on mongodb.org website:

> MongoDB wasn't designed in a lab. We built MongoDB from our own experiences building large scale, high availability, robust systems. We didn't start from scratch, we really tried to figure out what was broken, and tackle that. So the way I think about MongoDB is that if you take MySql, and change the data model from relational to document based, you get a lot of great features: embedded docs for speed, manageability, agile development with schema-less databases, easier horizontal scalability because joins aren't as important. There are lots of things that work great in relational databases: indexes, dynamic queries and updates to name a few, and we haven't changed much there. For example, the way you design your indexes in MongoDB should be exactly the way you do it in MySql or Oracle, you just have the option of indexing an embedded field.

The product was developed using a document model. Whereas atomicity, breaking data elements into atoms that could not be further decomposed, was central to the relational model (related to the "A" in the previously described ACID properties), MongoDB was built around a document model. A document, in turn, could be atomic but was much more likely to be a collection of atoms such as an array or list. It could also be a complex object, such as a data type defined as a class. Doing this dramatically reduced the amount of communication and processing that required for joining tables in the relational model. These savings proved to be particularly critical when tables and databases were distributed across servers.

The MongoDB website identified the key advantages of the product being the following:

- **Document-oriented**
  - Documents (objects) map nicely to programming language data types
  - Embedded documents and arrays reduce need for joins
  - Dynamically-typed (schemaless) for easy schema evolution
  - No joins and no multi-document transactions for high performance and easy scalability
- **High performance**
  - No joins and embedding makes reads and writes fast
  - Indexes including indexing of keys from embedded documents and arrays
  - Optional streaming writes (no acknowledgements)
- **High availability**
  - Replicated servers with automatic master failover
- **Easy scalability**
  - Automatic sharding (auto-partitioning of data across servers)
  - Reads and writes are distributed over shards
  - No joins or multi-document transactions make distributed queries easy and fast
  - Eventually-consistent reads can be distributed over replicated servers
- **Rich query language**

Exhibit 2 illustrates the use of shards to break databases across multiple systems.

Web Analytics

Lasa's company, SiteWit, was a participant in the broader marketplace known as web analytics. As companies became more and more dependent upon the web for communications and customer support, the need to understand what customers were looking at and, even more importantly, what website characteristics influenced customer decisions became critical. Web analytics, broadly defined, studied web traffic in an attempt to understand website effectiveness.

According to a 2011 report by the Gartner Group [Gassman 2011], the web analytics marketplace could be divided into three broad segments, only one of which seems likely to offer much revenue generation potential:

- ***Low end***, where basic traffic is measured but little sophisticated analysis is performed. The free standard version of Google Analytics, and various open source tools serve the needs of this market segment.
- ***Middle***, where companies have attempted to interpret basic measures in terms of their business value. Typically, free tools are used to gather these metrics. In some cases, however, organizations in this category acquire additional applications, or may move to the high end tier once such value has been demonstrated.
- ***High end***, where businesses make a systematic study of website value and are willing to invest in such value once it can be measured. Gassman provides examples that include:
    - Automated processes to optimize online campaigns and behavior on the website.
    - Ability to target landing page content to suit the context of visitors and to customize content to visitors' behavior throughout their visits.
    - Ability to mash Web analytics data with other data, including transaction, master customer and third-party data and social media metrics.

According to the Gartner study, four companies dominated the high end customer segment. These companies were:

1. *Adobe:* Reporting about 6000 customers accounting for almost $500 million in revenue. The company had, through internal development and acquisition, acquired a full suite of products including tools for social analytics, one of the most rapidly growing areas of interest.
2. *Google:* By far the largest competitor, with a reported installed base of over 200,000 customers—most of whom used the free standard version of Google Analytics. In 2011 it introduced a premium service with vastly expanded capabilities. According to Gassman, a subscription to the premium service typically cost $150,000/year.
3. *IBM:* Entered the web analytics market in 2010 by acquiring two mid-size players with a combined customer base estimated to be somewhere around 3000 clients.
4. *Webtrends:* A private company reported to have around 3500 customers. It was also growing through acquisition of smaller companies.

All told, it seemed likely that the existing global web analytics market was well in excess of $1 billion. It was also evident that all four of the largest participants were growing through aggressive acquisition of much smaller firms.

SiteWit

SiteWit, headquartered in Tampa, Florida, was on the leading edge of the market for online predictive analytics and paid search optimization software. It provided an online marketing optimization and predictive analytics platform that allows online marketers to optimize their Google AdWords and Bing adCenter campaigns, with Facebook soon to follow. Pay-per-click campaign management was available

within the SiteWit.com software-as-a-service (SaaS) platform, along with predictive analytics that segmented and scored website traffic. The company offered a "freemium" model, with all website monitoring, traffic reports, and predictive analytics available at no cost. Website traffic monitoring relied on a comprehensive revenue attribution model that uses first click, last click, and multi-click attribution to better understand how multiple visitor sessions affect purchasing and other e-commerce actions. Active campaign management was offered at a flat fee, rather using the traditional advertising model that based charges on a percentage of ad spend. Excerpts from the SiteWit website are included in Exhibit 3.

Management Team

SiteWit was established in 2009 as a result of conversations between Ricardo Lasa and Donald Berndt, who became the company's founders. By the time of the case, the company's management team had grown to seven employees, some of whom were part time.

Ricardo Lasa

Ricardo Lasa, SiteWit's CEO, was originally from Madrid, Spain. He grew up around businesses as his father Jose Luis Lasa built a large and successful real estate development firm. Lasa came to the United States to finish his undergraduate degree in MIS at the University of South Florida (USF), and went on to complete both a Masters degree in MIS and an MBA. He stays active in the technical community, serving on the Advisory Board of the Information Systems Department at USF, as well as in organizations such as the Tampa Bay Technology Forum (tbtf.org) and Tampa Bay WaVE (tampabaywave.org). He has been the CEO and founder of several other technology startups, including Web Piston (webpiston.com), a do-it-yourself website builder [Gill and Lasa, 2010] and Rivergy, Inc. a leading web developer in the Tampa Bay area. Ricardo Lasa gained critical experience in understanding the software-as-a-service (SaaS) business model through Web Piston, selling thousands of websites via online sales. Web Piston relied heavily on online advertising, running campaigns around the world. It was this experience that led him to co-found SiteWit Corporation, using the early version of the service to optimize his own Web Piston campaigns. Ricardo Lasa is a CEO with a lot of technical depth and he helped develop many of the core SiteWit components along with a small group of programmers that have worked together for a long time (building both Rivergy and Web Piston).

Donald Berndt

Donald Berndt, SiteWit's Chief Scientist, was an Associate Professor at the University of South Florida in addition to working at SiteWit, where his research focused on data mining, business intelligence, bioterrorism surveillance, and healthcare data warehousing and management. His academic credentials included a doctorate from the Stern School of Business at New York University, an MS from SUNY Stony Brook, and a BS from the University of Rhode Island. Prior to joining USF, he was an instructor and lecturer at SUNY Stony Brook and New York University. He was also a research programmer for Yale University.

Berndt co-founded SiteWit as a result of sitting on the board of conversations he had with Lasa while sitting on the advisory board of WebPiston, another company founded by Lasa. Additional information on Lasa, Berndt and the remainder of the SiteWit management team taken from the SiteWit website is presented in Exhibit 4.

The System Architecture

SiteWit was designed and developed for cloud computing from the outset. The company ran on Amazon Web Services (AWS), although other vendors had also been used during development. Cloud computing offered a flexible and cost effective infrastructure for the data intensive Web analytic tasks that underpinned SiteWit's functionality. The high-level architecture, as shown in Exhibit 5, was specifically designed with high availability and scalability in mind.

Availability

The cloud computing environment offered some significant advantages with respect to cost and on-demand resources, but the virtualized servers also brought challenges related to somewhat unpredictable I/O latencies and discrete failures. To meet availability goals, SiteWit layered more traditional database recovery and availability strategies on top of the cloud-based components. In particular, the core relational database servers were mirrored with failover capabilities. These databases were also used to refresh the development environment with real data. Finally, the lowest level web log data was continually archived to a separate database instance. All other services were provided using easily replicated commodity servers for redundancy and performance gains through coarse-grained parallelism. The cloud computing infrastructure made it very easy to provision new servers to meet demands.

Scalability

Given the data intensive nature of the SiteWit feature set, one of the most important aspects of the architecture was scalability. Careful consideration was given to the location of computationally demanding tasks, leaving some within the core database servers and locating others on commodity application servers. SiteWit used several groups of such servers for data collection, session processing (on application servers), and reporting. Dedicated Web servers that recorded the low-level page hit data handled data collection. Most importantly, the very intensive processes used to group sessions into threads for visitor histories, compute the many session attributes for predictive modeling, and handle cost and revenue attribution all took place on a collection of dedicated application servers that could easily be expanded to meet escalating demands. SiteWit maintained three attribution models: first click, last click, and multi-click (even across funnel) attribution. An extensive process status and queuing system was used to distribute tasks across this server group.

Another demanding task was creating the aggregated summary data used for reporting. Again, a collection of reporting servers could be used, incrementally pulling low-level data and producing the various aggregations necessary for presentation via SiteWit Web servers. The core database servers coordinated the activities of these satellite server groups and handled specialized tasks such as training predictive models for visitor scoring and segmentation.

NoSQL at SiteWit

Motivated by the growing challenge of handling huge flows of analytical data with its existing RDBMS architecture, the technical team at SiteWit had already looked at some alternative NoSQL databases, even running some preliminary tests. They had already started using MongoDB in a limited way, to serve documents within its overall architecture for test purposes. In addition, two corporate partners had gained some experience with specific systems. One partner had already made the leap, building their system using Citrusleaf (a NoSQL platform). Their products had extremely high performance demands and their experience was very positive. The other partner had completed some experiments with NoSQL systems, such as MongoDB. In fact, SiteWit engineers had joined their staff at a recent MongoDB conference.

Chris Lord, CTO and Matt Munday, Chief Software Architect (CSA) both attended the MongoDB conference and were evaluating other NoSQL technologies as well. While there was a lot of positive hype surrounding many of the platforms, all available technologies made tradeoffs and had significant limitations. Matt Munday (always the skeptic) had done some digging around looking for some outside opinions on NoSQL databases and MongoDB in particular. In early 2012, he posted a fairly in-depth review on the internal network (excerpted in Exhibit 6).

The Decision

Lasa knew his company was approaching a waypoint that would require a course correction. Things were going well. The company had already passed many critical points that could sink a startup. The core team had developed a complex product, which was already selling in the marketplace. Along the way, a beta version had allowed his company to raise money from angel investors and then from a small Series A funding round. With money in the bank and several products in the market, the company was adding a bit more development depth and focusing on growing the sales team. So, why was he again facing more sleepless nights?

The Challenge

The challenge was coming from a critical technical issue: scaling. At the core, SiteWit was an analytics company. A very large of amount of detailed Web analytics data was collected and processed as part of delivering online advertising services, such as keyword bidding, campaign optimization, and predictive analytics for re-marketing. The prospect of adding many more clients meant facing dramatic growth in the sheer volume of data being processed. While they had already faced several milestones and had re-engineered key processes to meet performance goals, explosive growth would certainly bring new challenges. Chief among these challenges would be to scale the core database technologies. So, to continue the nautical theme, Lasa and his technical team were facing the need to choose a course. One course involved sticking with well-known relational database technologies with some tacks (in steady winds) along the way to adjust to growing demands. His core team was well versed in the Microsoft technology stack and had worked together for more than a decade on software-as-a-service (SaaS) applications. The other course was akin to a jibe in heavy winds, a high-risk and dramatic change in direction that involved re-implementing core components in newer, highly distributed NoSQL databases. So, the decision could be summarized as follows: SQL or NoSQL that is the question.

The decision would affect every product and service offered by the company—since every one of them ultimately derived from the company's ability to capture every click on a website and, subsequently, to process that information at various levels of abstraction which also needed to be stored. If SiteWit's ability to manage this data was degraded, it would lose its effectiveness in its key sources of customer value: automatically adjusting bids for search terms, developing suggestions for optimizing advertising campaigns, and creating predictive models for scoring or segmenting website visitors. Lasa often repeated, especially within earshot of the technical staff, that the "biggest risk facing the company is the engine." He had little doubt that the biggest risks in failing to deliver quality services, as well as scaling for future growth, were related to the core data collection and processing infrastructure. Among these risks, a few specific threats particularly stood out.

1. The key to providing high quality campaign optimization services and predictive models was having the fine-grained data necessary for analysis. Whenever the data collection and processing systems fail, most other services also need to be paused (directly affecting the customer experience).
2. Even when the processing services were running, most of the customer experience was driven by the availability of insightful reports that were challenging to compute. Slowdowns in the data infrastructure meant delays in delivering reports and a poor customer experience.
3. An important competitive advantage for SiteWit was the highly automated implementation of even complex tasks, such as predictive modeling. This enabled the delivery of sophisticated services at very affordable prices. Any issues that needed to be resolved by highly (or even moderately) skilled labor cut deeply into profits.
4. The key to long-term success for SiteWit was a very large customer base with low prices and low costs. That meant that the core data intensive tasks needed to grow much larger in scale. Internet

giants, including Google, Amazon, or Facebook, had already made the transition to big data. There was no other way that they could achieve acceptable performance.

In the back of his mind, Lasa also knew that the nature of his competition was changing. Thus far, he had succeeded by maintaining a technological edge. If he lost that edge, how could he make a business case for SiteWit against giants such as Adobe, Google and IBM?

The Choices

So, with these factors in mind, Lasa and his team faced an interesting set of choices. Broadly speaking these fell into three categories.

1. **Do nothing.** SiteWit Corporation was a lean startup with limited resources. As a startup, simply surviving the initial growth stage and establishing product-market fit with early adopters had been a challenge. Do they really need to add new technologies and more uncertainty at this stage? Perhaps the most prudent course was to focus on refining the products and gaining valuable early customers before worrying about scaling. Did it make sense to "bet the business" with a largely unknown and, to a great extent, unproven technology? After, SiteWit did not have the resources of a Google to spend whatever it might take to make its solution work when difficulties were encountered.
2. **Proceed cautiously with NoSQL technology through limited experiments.** Even though SiteWit was an early stage company, it boasted a culture of research since its products rested on a foundation of big data, analytics, and machine learning. In addition, a data-driven approach was taken in the development process as part of the lean startup philosophy, including "innovation accounting" and the learning cycle [Ries 2011]. It might be reasonable to pick some component that could be implemented using NoSQL technology to gain experience and validate the technology (and better understand the specific benefits within the SiteWit context). In fact, it might also be possible to build a parallel implementation of a component that would enable a very realistic benchmarking comparison. While this seemed like a prudent approach, the web analytics market was changing so rapidly that prudence could easily mean being left behind.
3. **Develop a new product, alone or through a partnership, that makes use of NoSQL technologies.** A couple of potential SiteWit partners were experimenting with—or even already resting firmly on—NoSQL technologies. In some cases, the technology was a bit different than what would be used within SiteWit. Nevertheless, the technologies were certainly close enough to shed light on potential benefits. One strategy might be to identify a partnership opportunity that would make use of a NoSQL database, learning both from the partner and experience of developing a real system (with shared value). This option would certainly proceed faster than the second option, but would also involve some loss of control.
4. **Take a leap of faith.** Again, SiteWit was an early stage company facing plenty of risk factors. Adding a few more for an important competitive advantage could be a reasonable tradeoff given the size of the market and the potential premium of being considered the industry leader. Since development resources were limited, it would pay to hire the best engineers on the critical NoSQL database project that was the heartbeat of all product offerings. Splitting the attention of key technical staff would likely be a recipe for disaster, with the possibility of poorly implementing both SQL and NoSQL databases. In addition, there were several analytics-oriented startups that have successfully implemented NoSQL platforms and had grown quickly with the confidence to scale. As the Nike slogan goes: just do it.

References

[Chang et al. 2006] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber, "Bigtable: A Distributed Storage System for Structured Data," *Seventh Symposium on Operating System Design and Implementation* (OSDI), 2006.

[Codd 1970] E. F. Codd, "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM*, 13 (6): 377–387, 1970. DOI: 10.1145/362384.362685

[Codd 1982] E. F. Codd, "Relational Database: A Practical Foundation for Productivity," *Communications of the ACM*, 25 (2): 109–117, February 1982.

[Gassman 2011] B. Gassman, "Web Analytics Market Update, 2012," *The Gartner Group*. 17 November 2011, ID:G00224599

[Gelernter and Carriero 1992] D. Gelernter and N. Carriero, "Coordination Languages and Their Significance," *Communications of the ACM*, 35 (2): 97–107, February 1992. DOI: 10.1145/129630.129635

[Ghemawat et al. 2003] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, "The Google File System," *ACM Symposium on Operating Systems Principles* (SOSP), October 19-22, 2003.

[Gill and Lasa 2010] T. Grandon Gill and Ricardo Lasa, "Web Piston: Choosing a New Strategy," *ICIS 2010 Proceedings*, Paper 132, 2010. http://aisel.aisnet.org/icis2010_submissions/132

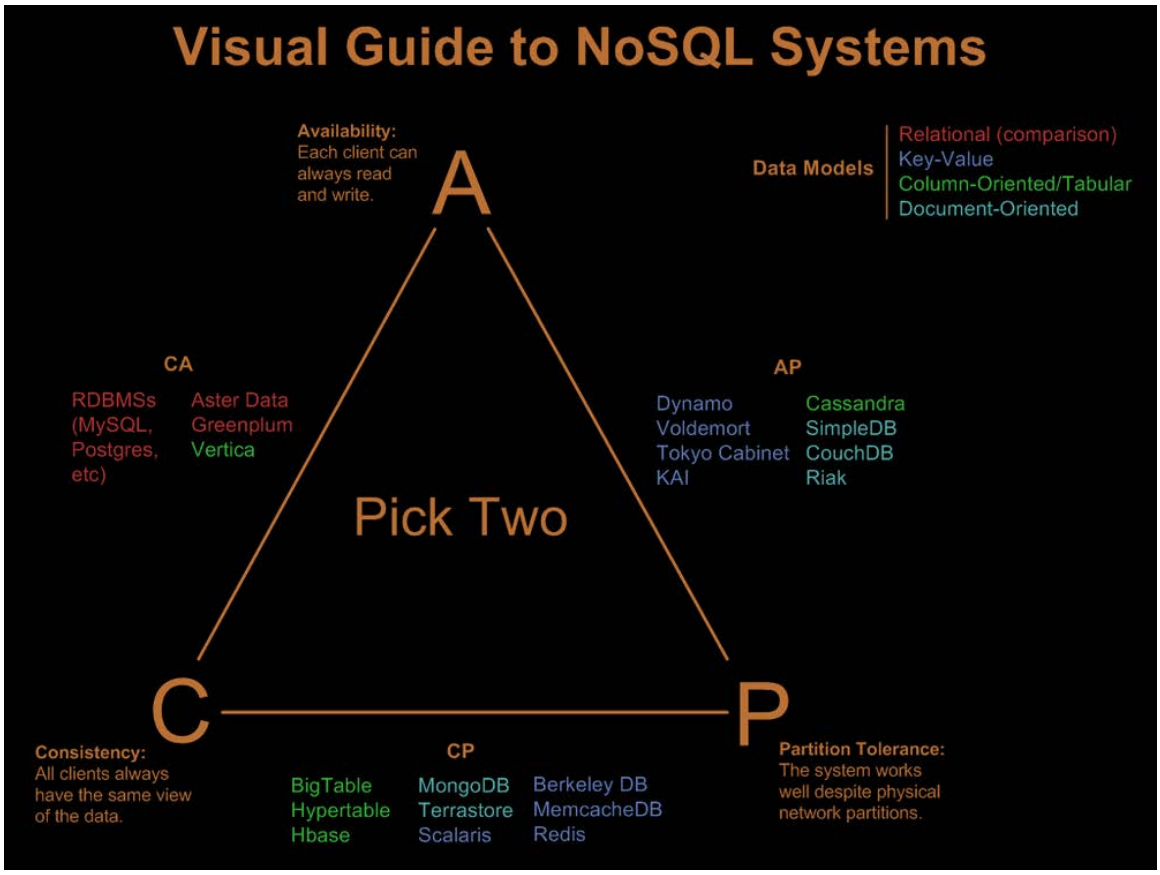
[Hurst 2010] Nathan Hurst, http://blog.nahurst.com/visual-guide-to-nosql-systems, March 15, 2010.

[Prichett 2008] Dan Prichett, "BASE: An ACID Alternative," ACM Queue, May/June 2008.

[Ries 2011] Eric Ries, *The Lean Startup*, Crown Publishing Group, Random House, Inc.

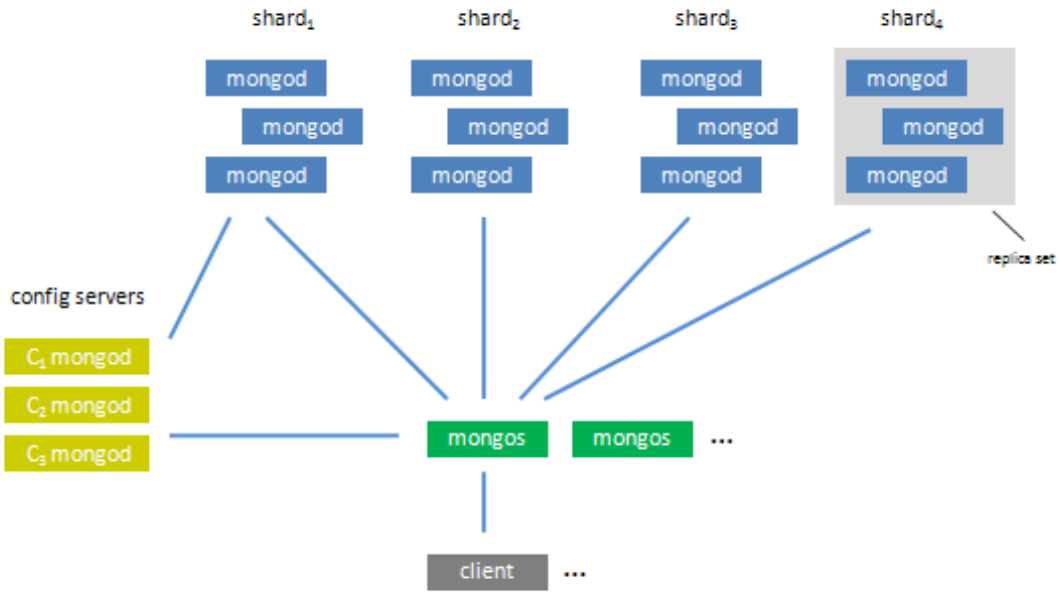
[Seth and Gilbert 2002] Nancy Lynch and Seth Gilbert, "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services," *ACM SIGACT News*, Volume 33, Issue 2, 2002.

Exhibit 1: Visual Guide to NoSQL Systems



Source: [Hurst 2010]

Exhibit 2: A Distributed Set of MongoDB Servers or Shards



Source: mongodb.org

Exhibit 3: Excerpts from SiteWit Website

Easily Build and Manage Online Advertising Campaigns for your Business

Get More Clients Through Internet Marketing

People all over are searching for businesses just like yours. If you're small business isn't engaged in online marketing, you're missing out on their business! We'll put your name out there and start driving those potential customers to your site by advertising your small to medium business on Google.

Transparent Pricing

Internet marketing doesn't have to cost you an arm and a leg. We'll charge your small business a small, flat fee for our Internet marketing services. You'll also know your fees upfront and where your advertising dollars are going. As a small business, we understand you're budget is limited. That's why we make every dollar work for you!

Complete Online Marketing Management

We take care of all of your small businesses' online marketing needs! All you have to do is go through our quick sign up process and we'll create, manage, and optimize your online marketing campaigns. As a growing small business, it is important to utilize online marketing because it really is the new way to advertise.

No Previous PPC Experience Required

We built SMB specifically for small to medium businesses who have no prior experience with PPC. We take care of the technical stuff while you watch as your customer base grows. Internet Marketing is the wave of the future and the best part is we make it easy and affordable to start!

Flexible Advertising Budgets

Start advertising your business on Google for as low as $50 per month. As your business grows so can your advertising budget. This in turn will bring in even more customers to your site. Our low prices will allow you start marketing on the Internet with ease.

Always Working For You

SiteWit optimizes and monitors your online paid search campaigns 24x7. Your marketing budget will be spent in the best way possible. You can't go wrong when you have the experts at Sitewit taking care of everything.

Source: www.sitewit.com

Exhibit 4: SiteWit Management Team

Team Leaders

SiteWit was founded in 2009 by Ricardo Lasa and Dr. Donald Berndt with the mission of finding a way to objectively measure the quality of internet traffic. Ricardo was at the time Chief Executive Officer of Web Piston Website Builder and Don was sitting on Web Piston's advisory board.

After an advisory board meeting Ricardo and Don started talking about ways to improve Web Piston's paid search campaigns. Don's background in data mining and Ricardo's domain knowledge were a perfect match to explore how data mining could be used to purchase only the best available traffic. That meeting was the spark that got SiteWit started.

Since then, the founders have put together a team of the best and brightest software engineers, data mining experts, and business people to put together a next generation PPC bid management and behavioral analytics suite.

The result is SiteWit. The core of the team has been working together for over 10 years and we greatly enjoy working with each other, as crazy as that might seem!

Our Team

Ricardo Lasa – Co-Founder, Chief Executive Officer

Ricardo is a hard working serial entrepreneur that is highly involved in the technology and business community in Tampa Bay. Prior to co-founding SiteWit, Ricardo started Web Piston Website Builder, now one of the leading platforms for small businesses to build their website and market their products and services. Through running Web Piston Ricardo realized there had to be a better way to buy paid search and started to work on SiteWit with Don in 2006. Web Piston is a thriving company built on sound business principles, profitable and growing even in these dire times of the "Great Recession." Ricardo is focusing all his efforts now in growing SiteWit and helping Search Engine Marketing firms provide better online marketing for their clients through a simple and cost efficient mechanism.

Ricardo holds a bachelors degree in Management information systems, a Master of Business Administration with emphasis in Entrepreneurship, Marketing, and International Management, and a Master of Science in Management Information Systems with emphasis on Database Architecture and User Interface Design, all from the University of South Florida Business School. Ricardo also attended the Harvard Business School Launching New Ventures program in 2009 and is looking forward to joining the HBS Owner/President Management program in 2010. Ricardo sits on the board of directors of the Tampa Bay Technology Forum, the advisory board of the Management Information Systems from the University of South Florida Business School, and the board of directors of Filasa, a real estate development firm headquartered in Madrid, Spain.

Don Berndt ph.D – Co-Founder, Chief Scientist

Don is a professor in the College of Business at the University of South Florida. His research interests are centered on business intelligence technologies such as data warehousing, data mining, and text mining. A particular emphasis of his work is the application of these technologies in healthcare, including data and text mining for electronic medical records. More recent work has focused on information markets as another mechanism for prediction and collective intelligence (working together with Ricardo on Agorx). He has published more than 75 research papers on these topics and worked with a number of entrepreneurial technology firms, as well as other organizations throughout his career.

His first exposure to entrepreneurism was a position as a LISP programmer at Cognitive Systems, an artificial intelligence start-up associated with Yale University. That was followed by work in scientific computing and healthcare analytics at other companies. Working with Ricardo on SiteWit has provided an opportunity to apply data warehousing and data mining technologies to website traffic analysis and the development of learning algorithms for managing paid search. Don received his Ph.D. in Management Information Systems from the Stern School of Business at New York University and M.S. in Computer Science from Stony Brook University.

Jesse Baynard – Lead User Experience

Jesse holds a Masters degree in Information Systems from the University of South Florida. Since graduating in 2000, Jesse's primary focus has been in turning complex business problems into simple and effective technology solutions. These solutions have spanned across numerous vertical markets including healthcare, back office management, and e-commerce. Now as a member of the SiteWit management team, Jesse is dedicated to bringing clarity and expertise to the problem of search engine marketing inefficiencies. Intrigued by the power of mining behavioral analytics in predicting behavior, Jesse believes that this technology is the best automated approach for a problem that is far too complicated for click-through or simple ROI ratios alone. In his spare time he is involved in the Tampa Bay area tech community, an avid photographer and parent of two children.

Matt Munday – Lead Software Developer

Matt is SiteWit's lead developer with many years of experience in designing, building and implementing web and backend applications for e-commerce and web development that have been tremendous business successes. Matt is an avid programmer who spends a lot of his free time creating applications for Google's Android Os, designing and creating templates and plugins for WordPress and a template for Zenphoto, and working with open source projects such as Snark. Matt has deep level expertise in Java, C++, C#, VB.NET, PHP & TSQL.

Chris Lord – Lead Software Developer

Chris is a detail oriented software developer with extensive experience in e-commerce development. Before working on Site Wit Chris developed a major shopping cart platform used by thousands of business owners in several countries. He developed the billing, customer management, and customer communication systems for Web Piston Website Builder and many customized e-commerce solutions for businesses in vastly different markets. Chris also oversees the network and hosting infrastructure of SiteWit and its cloud implementation. He has received certifications from both CIW and Microsoft.

Evan Bushelman – Lead Customer Advocate

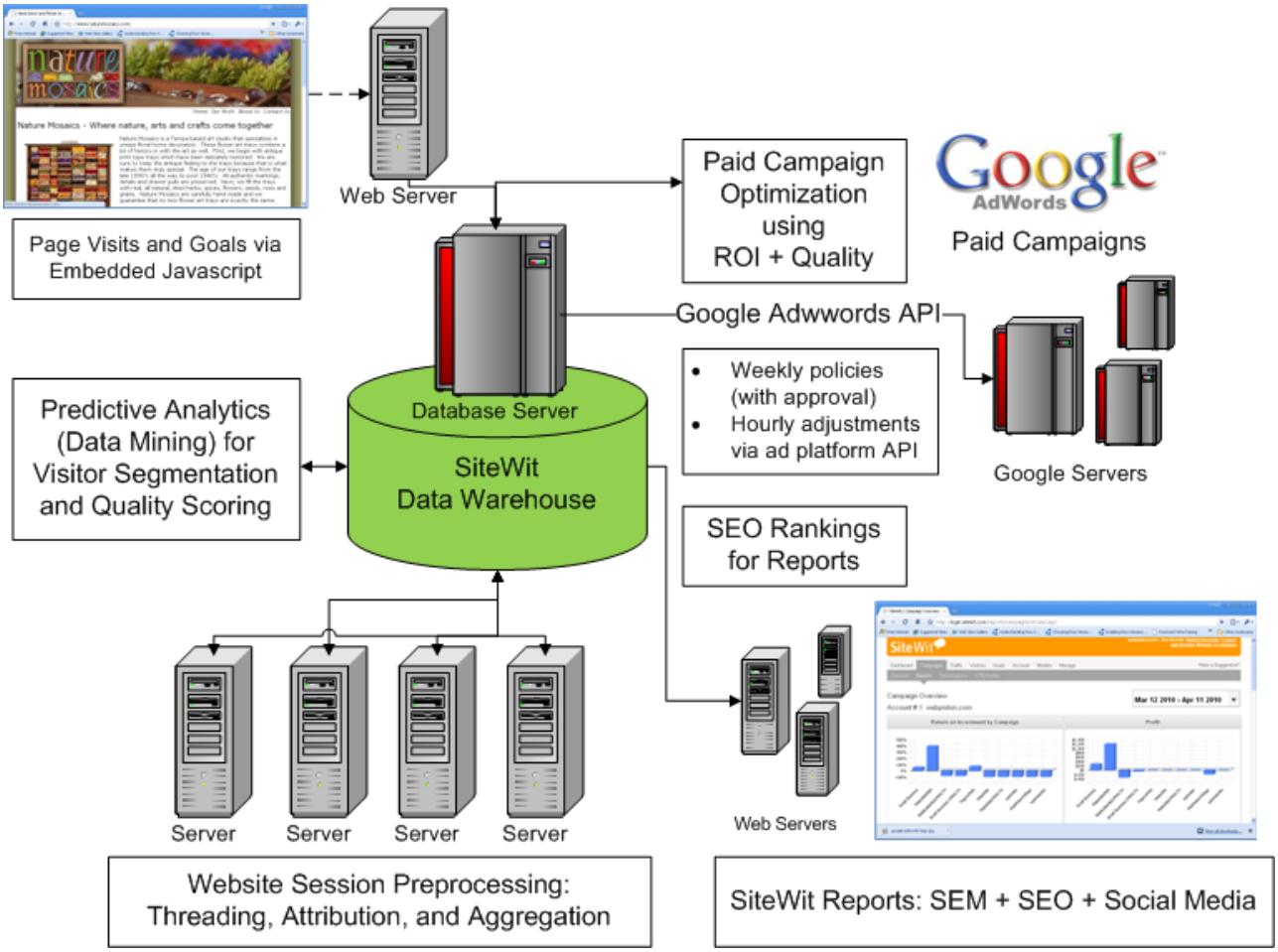
Evan is responsible for building and managing relationships with our customer base. We first met Evan during the summer of '09 when he interned for Web Piston. He instantly left his mark, which is why he was invited to join the SiteWit team once he graduated. Evan holds a Bachelor's degree in Finance from the University of South Florida. He is a Google AdWords Certified Partner and available to answer all of your search marketing inquiries.

Andy Montoya – Customer Advocate

Andy is a graduate from the University of South Florida having acquired a Bachelor's degree in Business Administration with a concentration in Marketing. He was an active member of the American Marketing Association USF Chapter and learned the essentials of building strong business relationships. Andy has worked in the customer service field for over five years and understands the value of effective communication. He listens with an empathetic ear and makes sure that all customer concerns are dealt with quickly and efficiently. Since working at SiteWit, Andy has been a problem-solver for customers and has helped in the implementation of various marketing initiatives.

Source: www.sitewite.com/team-leaders/

Exhibit 5: The SiteWit System Architecture



Source: Prepared by case writers

Exhibit 6: Excerpts from Matt Munday's Review of MongoDB

The post began by noting some key strengths of the product:

> To be fair, it must be acknowledged that MongoDB is popular, and that there are valid reasons for its popularity.
>
> * It is remarkably easy to get running.
>
> * Schema-free models that map to JSON-like structures have great appeal to developers (they fit our brains), and a developer is almost always the individual who makes the platform decisions when a project is in its infancy.
>
> * Maturity and robustness, track record, tested real-world use cases, etc, are typically more important to sysadmin types or operations specialists, who often inherit the platform long after the initial decisions are made.
>
> * Its single-system, low concurrency read performance benchmarks are impressive, and for the inexperienced evaluator, this is often The Most Important Thing.

The post then went on to warn about some serious problems.

> But if you're intending to really run a large scale system on Mongo, one that a business might depend on, simply put:
>
> **1. MongoDB issues writes in unsafe ways *by default* in order to win benchmarks**
>
> **2. MongoDB can lose data in many startling ways**
>
> **3. MongoDB requires a global write lock to issue any write**
> Under a write-heavy load, this will kill you.
>
> **4. MongoDB's sharding doesn't work that well under load**
>
> **5. mongos is unreliable**
>
> **7. Things were shipped that should have never been shipped**
>
> **8. Replication was lackluster on busy servers**
>
> Please take this warning seriously.

Source: SiteWit internal network posting