Big Data

- The growth of user-driven content has fueled a rapid increase in the volume and type of data that is
  - generated
  - manipulated
  - analyzed
  - archived

- In addition, varied newer sets of sources, including sensors, Global Positioning Systems (GPS), automated trackers and monitoring systems, are generating a lot of data.

- These larger volumes of data sets, often termed big data, are imposing newer challenges and opportunities around storage, analysis, and archival.

Semi-structured and Sparse Data

- In parallel to the fast data growth, data is also becoming increasingly semi-structured and sparse.
- This means the traditional data management techniques around upfront schema definition and relational references is also being questioned.
- The quest to solve the problems related to large-volume and semi-structured data has led to the emergence of a class of newer types of database products.
- This new class of database products consists of column-oriented data stores, key/value pair databases, and document databases.
- Collectively, these are identified as NoSQL.

NoSQL Products

- The products that fall under the NoSQL umbrella are quite varied, each with their unique sets of features and value propositions.
- Given this, it often becomes difficult to decide which product to use for the case at hand.
- This class prepares you to understand the entire NoSQL landscape.
- It provides the essential concepts that act as the building blocks for many of the NoSQL products.
- Instead of covering a single product exhaustively, it provides a fair coverage of a number of different NoSQL products.
- The emphasis is often on breadth and underlying concepts rather than a full coverage of every API.
- Because a number of NoSQL products are covered, a good bit of comparative analysis is also included.

What this class covers

- This class starts with the essentials of NoSQL and graduates to advanced concepts around performance tuning and architectural guidelines.
- It focuses all along on the fundamental concepts that relate to NoSQL and explains those in the context of a number of different NoSQL products.
- We will have illustrations and examples that relate to MongoDB, CouchDB, HBase, Hypertable, Cassandra, Redis, and Berkeley DB.
- A few other NoSQL products, besides these, are also referenced.

Manipulating Big Data

- An important part of NoSQL is the way large data sets are manipulated.
- We will covers all the essentials of MapReduce-based scalable processing.
- We will have a few examples using Hadoop.
- Higher-level abstractions like Hive and Pig are also illustrated.

This class is divided into four parts:

- Part I: Getting Started
- Part II: Learning the NoSQL Basics
- Part III: Gaining Profi ciency with NoSQL
- Part IV: Mastering NoSQL

Definition and Introduction

- NoSQL is literally a combination of two words: No and SQL.
- The implication is that NoSQL is a technology or product that counters SQL.
- The creators and early adopters of the buzzword NoSQL probably wanted to say No RDBMS or No relational but were infatuated by the nicer sounding NoSQL and stuck to it.
- In due course, some have proposed NonRel as an alternative to NoSQL.
- A few others have tried to salvage the original term by proposing that NoSQL is actually an acronym that expands to “Not Only SQL.” Whatever the literal meaning, NoSQL is used today as an umbrella term for all databases and data stores that don’t follow the popular and well established RDBMS principles and often relate to large data sets accessed and manipulated on a Web scale.
- This means NoSQL is not a single product or even a single technology.
- It represents a class of products and a collection of diverse, and sometimes related, concepts about data storage and manipulation.

Context and History

- Before we start with details on the NoSQL types and the concepts involved, it's important to set the context in which NoSQL emerged.
- Non-relational databases are not new. In fact, the first non-relational stores go back in time to when the first set of computing machines were invented.
- Non-relational databases thrived through the advent of mainframes and have existed in specialized and specific domains — for example, hierarchical directories for storing authentication and authorization credentials — through the years.
- However, the non-relational stores that have appeared in the world of NoSQL are a new incarnation, which were born in the world of massively scalable Internet applications.
- These non-relational NoSQL stores, for the most part, were conceived in the world of distributed and parallel computing.

RDBMS Problems

- Starting out with Inktomi, which could be thought of as the first true search engine, and culminating with Google, it is clear that the widely adopted relational database management system (RDBMS) has its own set of problems when applied to massive amounts of data.

- The problems relate to
  - efficient processing
  - effective parallelization
  - Scalability
  - costs.

- We will discuss about each of these problems and the possible solutions to the problems through the rest of this semester.

Challenges of RDBMS

- The challenges of RDBMS for massive Web-scale data processing aren't specific to a product but pertain to the entire class of such databases.
- RDBMS assumes a well-defined structure in data.
- It assumes that the data is dense and is largely uniform.
- RDBMS builds on a prerequisite that the properties of the data can be defined up front and that its interrelationships are well established and systematically referenced.
- It also assumes that indexes can be consistently defined on data sets and that such indexes can be uniformly leveraged for faster querying.

RDBMS Assumptions

- Unfortunately, RDBMS starts to show signs of giving way as soon as these assumptions don't hold true.
- RDBMS can certainly deal with some irregularities and lack of structure but in the context of massive sparse data sets with loosely defined structures, RDBMS appears a forced fit.
- With massive data sets the typical storage mechanisms and access methods also get stretched.
- Denormalizing tables, dropping constraints, and relaxing transactional guarantee can help an RDBMS scale, but after these modifications an RDBMS starts resembling a NoSQL product.

- Flexibility comes at a price.
- NoSQL alleviates the problems that RDBMS imposes and makes it easy to work with large sparse data, but in turn takes away the power of transactional integrity and fl exible indexing and querying.
- Ironically, one of the features most missed in NoSQL is SQL, and product vendors in the space are making all sorts of attempts to bridge this gap.

Googlw

- Google has, over the past few years, built out a massively scalable infrastructure for its search engine and other applications, including Google Maps, Google Earth, GMail, Google Finance, and Google Apps.
- Google's approach was to solve the problem at every level of the application stack.
- The goal was to build a scalable infrastructure for parallel processing of large amounts of data.
- Google therefore created a full mechanism that included a distributed file system, a column-family-oriented data store, a distributed coordination system, and a MapReduce-based parallel algorithm execution environment.
- Graciously enough, Google published and presented a series of papers explaining some of the key pieces of its infrastructure.

The most important of these publications are as follows:

- Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. “The Google File System”; pub. 19th ACM Symposium on Operating Systems Principles, Lake George, NY, October 2003.
  - URL: http://labs.google.com/papers/gfs.html
- Jeffrey Dean and Sanjay Ghemawat. “MapReduce: Simplifi ed Data Processing on Large Clusters”; pub. OSDI’04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December 2004.
  - URL: http://labs.google.com/papers/mapreduce.html
- Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. “Bigtable: A Distributed Storage System for Structured Data”; pub. OSDI’06: Seventh Symposium on Operating System Design and Implementation, Seattle, WA, November 2006.
  - URL: http://labs.google.com/papers/bigtable.html
- Mike Burrows. “The Chubby Lock Service for Loosely-Coupled Distributed Systems”; pub. OSDI’06: Seventh Symposium on Operating System Design and Implementation, Seattle, WA, November 2006.
  - URL: http://labs.google.com/papers/chubby.html

Google Papers

- The release of Google's papers to the public spurred a lot of interest among open-source developers.
- The creators of the open-source search engine, Lucene, were the first to develop an open-source version that replicated some of the features of Google's infrastructure.
- Subsequently, the core Lucene developers joined Yahoo, where with the help of a host of other contributors, they created a parallel universe that mimicked all the pieces of the Google distributed computing stack.

Hadoop

- This open-source alternative is Hadoop, its sub-projects, and its related projects.
- You can find more information, code, and documentation on Hadoop at
  - http://hadoop.apache.org.
- Without getting into the exact timeline of Hadoop's development, somewhere toward the first of its releases emerged the idea of NoSQL.
- The history of who coined the term NoSQL and when is irrelevant, but it's important to note that the emergence of Hadoop laid the groundwork for the rapid growth of NoSQL.
- Also, it's important to consider that Google's success helped propel a healthy adoption of the new-age distributed computing concepts, the Hadoop project, and NoSQL.

Amazon Dynamo

- A year after the Google papers had catalyzed interest in parallel scalable processing and non-relational distributed data stores, Amazon decided to share some of its own success story.

- In 2007, Amazon presented its ideas of a distributed highly available and eventually consistent data store named Dynamo.

- You can read more about Amazon Dynamo in a research paper, the details of which are as follows:
  - Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swami Sivasubramanian, Peter Vosshall, and Werner Vogels,
  - "Dynamo: Amazon's Highly Available Key/value Store," in the Proceedings of the 21st ACM Symposium on Operating Systems Principles, Stevenson, WA, October 2007.

- Werner Vogels, the Amazon CTO, explained the key ideas behind Amazon Dynamo in a blog post accessible online at
  - www.allthingsdistributed.com/2007/10/amazons_dynamo.html

Other Products

- With endorsement of NoSQL from two leading web giants — Google and Amazon — several new products emerged in this space.
- A lot of developers started toying with the idea of using these methods in their applications and many enterprises, from startups to large corporations, became amenable to learning more about the technology and possibly using these methods.
- In less than 5 years, NoSQL and related concepts for managing big data have become widespread and use cases have emerged from many well-known companies, including Facebook, Netflix, Yahoo, EBay, Hulu, IBM, and many more.
- Many of these companies have also contributed by open sourcing their extensions and newer products to the world.

Just how much data qualifies as big data?

- This is a question that is bound to solicit different responses, depending on who you ask.
- The answers are also likely to vary depending on when the question is asked.
- Currently, any data set over a few terabytes is classified as big data.
- This is typically the size where the data set is large enough to start spanning multiple storage units.
- It's also the size at which traditional RDBMS techniques start showing the first signs of stress.

DATA SIZE MATH

A byte is a unit of digital information that consists of 8 bits. In the International System of Units (SI) scheme every 1,000 (10^3) multiple of a byte is given a distinct name, which is as follows:

- Kilobyte (kB) — 10^3
- Megabyte (MB) — 10^6
- Gigabyte (GB) — 10^9
- Terabyte (TB) — 10^{12}
- Petabyte (PB) — 10^{15}
- Exabyte (EB) — 10^{18}
- Zettabyte (ZB) — 10^{21}
- Yottabyte (YB) — 10^{24}

In traditional binary interpretation, multiples were supposed to be of 2^{10} (or 1,024) and not 10^3 (or 1,000). To avoid confusion, a parallel naming scheme exists for powers of 2, which is as follows:

- Kibibyte (KiB) — 2^{10}
- Mebibyte (MiB) — 2^{20}
- Gibibyte (GiB) — 2^{30}
- Tebibyte (TiB) — 2^{40}
- Pebibyte (PiB) — 2^{50}
- Exbibyte (EiB) — 2^{60}
- Zebibyte (ZiB) — 2^{70}
- Yobibyte (YiB) — 2^{80}

Data Consumption

- Even a couple of years back, a terabyte of personal data may have seemed quite large.
- However, now local hard drives and backup drives are commonly available at this size.
- In the next couple of years, it wouldn't be surprising if your default hard drive were over a few terabytes in capacity.
- We are living in an age of rampant data growth.
- Our digital camera outputs, blogs, daily social networking updates, tweets, electronic documents, scanned content, music files, and videos are growing at a rapid pace.
- We are consuming a lot of data and producing it too.
- It's difficult to assess the true size of digitized data or the size of the Internet but a few studies, estimates, and data points reveal that it's immensely large and in the range of a zettabyte and more.
- In an ongoing study titled, "The Digital Universe Decade – Are you ready?"
  http://emc.com/collateral/demos/microsites/idc-digital-universe/iview.htm
  IDC, on behalf of EMC, presents a view into the current state of digital data and its growth.
- The report claims that the total size of digital data created and replicated will grow to 35 zettabytes by 2020.
- The report also claims that the amount of data produced and available now is outgrowing the amount of available storage.

A few other data points worth considering are as follows:

- A 2009 paper in ACM titled, “MapReduce: simplified data processing on large clusters”
  - http://portal.acm.org/citation.cfm?id=1327452.1327492&coll=GUIDE&dl=&idx=J79&part=magazine&WantType=Magazines&title=Communications%20of%20the%20ACM revealed that Google processes 24 petabytes of data per day.

- A 2009 post from Facebook on its photo storage system, “Needle in a haystack: efficient storage of billions of photos”
  - http//facebook.com/note.php?note_id=76191543919

  mentioned the total size of photos in Facebook to be 1.5 pedabytes.
- The same post mentioned that around 60 billion images were stored on Facebook.

- The Internet archive FAQs at
  - http://archive.org/about/faqs.php say that 2 petabytes of data are stored in the Internet archive. It also says that the data is growing at the rate of 20 terabytes per month.

- The movie Avatar took up 1 petabyte of storage space for the rendering of 3D CGI effects.
  - “Believe it or not: Avatar takes 1 petabyte of storage space, equivalent to a 32-year-long MP3”
  - http://thenextweb.com/2010/01/01/avatar-takes-1-petabyte-storagespace-equivalent-32-year-long-mp3/.

Challenges

- As the size of data grows and sources of data creation become increasingly diverse, the following growing challenges will get further amplified:
- Efficiently storing and accessing large amounts of data is difficult.
- The additional demands of fault tolerance and backups makes things even more complicated.
- Manipulating large data sets involves running immensely parallel processes.
- Gracefully recovering from any failures during such a run and providing results in a reasonably short period of time is complex.
- Managing the continuously evolving schema and metadata for semi-structured and un-structured data, generated by diverse sources, is a convoluted problem.
- Therefore, the ways and means of storing and retrieving large amounts of data need newer approaches beyond our current methods.
- NoSQL and related big-data solutions are a first step forward in that direction.

Disk Storage And Data Read And Write Speed

- While the data size is growing and so are the storage capacities, the disk access speeds to write data to disk and read data from it is not keeping pace.
- Typical above-average current-generation 1 TB disks claim to access data at the rate of 300 Mbps, rotating at the speed of 7200 RPM.
- At these peak speeds, it takes about an hour (at best 55 minutes) to access 1 TB of data.
- With increased size, the time taken only increases.
- Besides, the claim of 300 Mbps at 7200 RPM speed is itself misleading.
- Traditional rotational media involves circular storage disks to optimize surface area.
- In a circle, 7200 RPM implies different amounts of data access depending on the circumference of the concentric circle being accessed.
- As the disk is filled, the circumference becomes smaller, leading to less area of the media sector being covered in each rotation.
- This means a peak speed of 300 Mbps degrades substantially by the time the disk is over 65 percent full.
- Solid-state drives (SSDs) are an alternative to rotational media.
- An SSD uses microchips, in contrast to electromechanical spinning disks.
- It retains data in volatile random-access memory.
- SSDs promise faster speeds and improved “input/output operations per second (IOPS)” performance as compared to rotational media.
- By late 2009 and early 2010, companies like Micron announced SSDs that could provide access speeds of over a Gbps www.dailytech.com/UPDATED+Micron+Announces+Worlds+First+Native+6Gbps+SATA+Solid+State+Drive/article17007.htm
- However, SSDs are fraught with bugs and issues as things stand and come at a much higher cost than their rotational media counterparts.
- Given that the disk access speeds cap the rate at which you can read and write data, it only make sense to spread the data out across multiple storage units rather than store them in a single large store.

Vertical Scalability

- Scalability is the ability of a system to increase throughput with addition of resources to address load increases.
- Scalability can be achieved either by provisioning a large and powerful resource to meet the additional demands or it can be achieved by relying on a cluster of ordinary machines to work as a unit.
- The involvement of large, powerful machines is typically classified as vertical scalability.
- Provisioning super computers with many CPU cores and large amounts of directly attached storage is a typical vertical scaling solution.
- Such vertical scaling options are typically expensive and proprietary.

Horizontal Scalability

- The alternative to vertical scalability is horizontal scalability.
- Horizontal scalability involves a cluster of commodity systems where the cluster scales as load increases.
- Horizontal scalability typically involves adding additional nodes to serve additional load.
- The advent of big data and the need for large-scale parallel processing to manipulate this data has led to the widespread adoption of horizontally scalable infrastructures.
- Some of these horizontally scaled infrastructures at Google, Amazon, Facebook, eBay, and Yahoo! involve a very large number of servers.
- Some of these infrastructures have thousands and even hundreds of thousands of servers.

MapReduce

- Processing data spread across a cluster of horizontally scaled machines is complex.
- The MapReduce model possibly provides one of the best possible methods to process large-scale data on a horizontal cluster of machines.
- MapReduce is a parallel programming model that allows distributed processing on large data sets on a cluster of computers.
- The MapReduce framework is patented by Google
  - http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PALL&p=1&u=/netahtml/PTO/srchnum.htm&r=1&f=G&l=50&s1=7,650,331.PN.&OS=PN/7,650,331&RS=PN/7,650,331
  - but the ideas are freely shared and adopted in a number of open-source implementations.

The Map Function

- MapReduce derives its ideas and inspiration from concepts in the world of functional programming.
- Map and reduce are commonly used functions in the world of functional programming.
- In functional programming, a map function applies an operation or a function to each element in a list.
  - For example, a multiply-by-two function on a list
    - [1, 2, 3, 4] would generate another list as follows: [2, 4, 6, 8].
- When such functions are applied, the original list is not altered.
- Functional programming believes in keeping data immutable and avoids sharing data among multiple processes or threads.
- This means the map function that was just illustrated, trivial as it may be, could be run via 2 or more multiple threads on the list and these threads would not step on each other, because the list itself is not altered.

The Reduce Function

- Like the map function, functional programming has a concept of a reduce function.
- Actually, a reduce function in functional programming is more commonly known as a fold function.
- A reduce or a fold function is also sometimes called an accumulate, compress, or inject function.
- A reduce or fold function applies a function on all elements of a data structure, such as a list, and produces a single result or output.
- So applying a reduce function-like summation on the list generated out of the map function, that is, [2, 4, 6, 8], would generate an output equal to 20.

Map and Reduce Functions

- So map and reduce functions could be used in conjunction to process lists of data, where a function is first applied to each member of a list and then an aggregate function is applied to the transformed and generated list.

Map and Reduce on Large Data Sets

- This same simple idea of map and reduce has been extended to work on large data sets.
- The idea is slightly modified to work on collections of tuples or key/value pairs.
- The map function applies a function on every key/value pair in the collection and generates a new collection.
- Then the reduce function works on the new generated collection and applies an aggregate function to compute a final output.

A Trivial Example To Explain The Flow

- Say you have a collection of key/value pairs as follows:
  [{ “94303”: “Tom”}, {“94303”: “Jane”}, {“94301”: “Arun”}, {“94302”: “Chen”}]
- This is a collection of key/value pairs where the key is the zip code and the value is the name of a person who resides within that zip code.
- A simple map function on this collection could get the names of all those who reside in a particular zip code.
- The output of such a map function is as follows:
  [{“94303”:[“Tom”, “Jane”]}, {“94301”:[“Arun”]}, {“94302”:[“Chen”]}]
- Now a reduce function could work on this output to simply count the number of people who belong to particular zip code.
- The final output then would be as follows:
  [{“94303”: 2}, {“94301”: 1}, {“94302”: 1}]

NoSQL Product Categories

- Now it is time to list some of the most well-known NoSQL products and categorize them in terms of their features and attributes.

- 1. Sorted Ordered Column-Oriented Stores
- 2. Key-Value Stores
- 3. Document Databases
- 4. Graph Databases

1. SORTED ORDERED COLUMN-ORIENTED STORES

- Google's Bigtable adopts a model where data in stored in a column-oriented way.
- This contrasts with the row-oriented format in RDBMS.
- The column-oriented storage allows data to be stored effectively.
- It avoids consuming space when storing nulls by simply not storing a column when a value doesn't exist for that column.
- Each unit of data can be thought of as a set of key/value pairs, where the unit itself is identified with the help of a primary identifier, often referred to as the primary key.
- Bigtable and its clones tend to call this primary key the row-key.
- Also, as the title of this subsection suggests, units are stored in an ordered-sorted manner.
- The units of data are sorted and ordered on the basis of the row-key.

BigTable Example

- To explain sorted ordered column-oriented stores, an example serves better than a lot of text, so let us go through an example.
- Consider a simple table of values that keeps information about a set of people.
- Such a table could have columns like first_name, last_name, occupation, zip_code, and gender.

- A person's information in this table could be as follows:
  first_name: John
  last_name: Doe
  zip_code: 10001
  gender: male

- Another set of data in the same table could be as follows:
  first_name: Jane
  zip_code: 94303

BigTable

- The row-key of the first data point could be 1 and the second could be 2.
- Then data would be stored in a sorted ordered column-oriented store in a way that the data point with row-key 1 will be stored before a data point with row-key 2 and also that the two data points will be adjacent to each other.
- Next, only the valid key/value pairs would be stored for each data point.
- So, a possible column-family for the example could be name with columns first_name and last_name being its members.
- Another column-family could be location with zip_code as its member.
- A third column-family could be profile.
- The gender column could be a member of the profile column-family.
- In column-oriented stores similar to Bigtable, data is stored on a column-family basis.
- Column-families are typically defined at configuration or startup time.
- Columns themselves need no a-priori defi nition or declaration.
- Also, columns are capable of storing any data types as far as the data can be persisted to an array of bytes.

So the underlying logical storage for this simple example consists of three storage buckets: `name`, `location`, and `profile`. Within each bucket, only key/value pairs with valid values are stored. Therefore, the `name` column-family bucket stores the following values:

```
For row-key: 1
first_name: John
last_name: Doe
For row-key: 2
first_name: Jane
```

The `location` column-family stores the following:

```
For row-key: 1
zip_code: 10001
For row-key: 2
zip_code: 94303
```

The `profile` column-family has values only for the data point with row-key 1 so it stores only the following:

```
For row-key: 1
gender: male
```

In real storage terms, the column-families are not physically isolated for a given row. All data pertaining to a row-key is stored together. The column-family acts as a key for the columns it contains and the row-key acts as the key for the whole data set.

Data in Bigtable and its clones is stored in a contiguous sequenced manner. As data grows to fill up one node, it is spilt into multiple nodes. The data is sorted and ordered not only on each node but also across nodes providing one large continuously sequenced set. The data is persisted in a fault-tolerant manner where three copies of each data set are maintained. Most Bigtable clones leverage a distributed filesystem to persist data to disk. Distributed filesystems allow data to be stored among a cluster of machines.

The sorted ordered structure makes data seek by row-key extremely efficient. Data access is less random and ad-hoc and lookup is as simple as finding the node in the sequence that holds the data. Data is inserted at the end of the list. Updates are in-place but often imply adding a newer version of data to the specific cell rather than in-place overwrites. This means a few versions of each cell are maintained at all times. The versioning property is usually configurable.

HBase is a popular, open-source, sorted ordered column-family store that is modeled on the ideas proposed by Google's Bigtable. Details about storing data in HBase and accessing it are covered in many chapters of this book.

Data stored in HBase can be manipulated using the MapReduce infrastructure. Hadoop's MapReduce tools can easily use HBase as the source and/or sink of data.

BigTable Clones

- The best way to learn about and leverage the ideas proposed by Google's infrastructure is to start with the Hadoop (http//hadoop.apache.org) family of products.

- The NoSQL Bigtable store called HBase is part of the Hadoop family.

HBase

- Official Online Resources — `http://hbase.apache.org`.
- History — Created at Powerset (now part of Microsoft) in 2007. Donated to the Apache foundation before Powerset was acquired by Microsoft.
- Technologies and Language — Implemented in Java.
- Access Methods — A JRuby shell allows command-line access to the store. Thrift, Avro, REST, and protobuf clients exist. A few language bindings are also available. A Java API is available with the distribution.

Protobuf, short for Protocol Buffers, is Google's data interchange format. More information is available online at `http://code.google.com/p/protobuf/`.

- Query Language — No native querying language. Hive (`http://hive.apache.org`) provides a SQL-like interface for HBase.
- Open-Source License — Apache License version 2.
- Who Uses It — Facebook, StumbleUpon, Hulu, Ning, Mahalo, Yahoo!, and others.

Hypertable

- Official Online Resources — `www.hypertable.org`.
- History — Created at Zvents in 2007. Now an independent open-source project.
- Technologies and Language — Implemented in C++, uses Google RE2 regular expression library. RE2 provides a fast and efficient implementation. Hypertable promises performance boost over HBase, potentially serving to reduce time and cost when dealing with large amounts of data.
- Access Methods — A command-line shell is available. In addition, a Thrift interface is supported. Language bindings have been created based on the Thrift interface. A creative developer has even created a JDBC-compliant interface for Hypertable.
- Query Language — HQL (Hypertable Query Language) is a SQL-like abstraction for querying Hypertable data. Hypertable also has an adapter for Hive.
- Open-Source License — GNU GPL version 2.
- Who Uses It — Zvents, Baidu (China's biggest search engine), Rediff (India's biggest portal).

Cloudata

- Official Online Resources — `www.cloudata.org/`.
- History — Created by a Korean developer named YK Kwon (`www.readwriteweb.com/hack/2011/02/open-source-bigtable-cloudata.php`). Not much is publicly known about its origins.
- Technologies and Language — Implemented in Java.
- Access Methods — A command-line access is available. Thrift, REST, and Java API are available.
- Query Language — CQL (Cloudata Query Language) defines a SQL-like query language.
- Open-Source License — Apache License version 2.
- Who Uses It — Not known.

2. KEY/VALUE STORES

- A HashMap or an associative array is the simplest data structure that can hold a set of key/value pairs.
- Such data structures are extremely popular because they provide a very efficient, big O(1) average algorithm running time for accessing data.
- The key of a key/value pair is a unique value in the set and can be easily looked up to access the data.
- Key/value pairs are of varied types: some keep the data in memory and some provide the capability to persist the data to disk.
- Key/value pairs can be distributed and held in a cluster of nodes.
- A simple, yet powerful, key/value store is Oracle's Berkeley DB.
- Berkeley DB is a pure storage engine where both key and value are an array of bytes.
- The core storage engine of Berkeley DB doesn't attach meaning to the key or the value.
- It takes byte array pairs in and returns the same back to the calling client.
- Berkeley DB allows data to be cached in memory and fl ushed to disk as it grows.
- There is also a notion of indexing the keys for faster lookup and access.
- Berkeley DB has existed since the mid-1990s.
- It was created to replace AT&T's NDBM as a part of migrating from BSD 4.3 to 4.4.
- In 1996, Sleepycat Software was formed to maintain and provide support for Berkeley DB.

Cache

- Another type of key/value store in common use is a cache.
- A cache provides an in-memory snapshot of the most-used data in an application.
- The purpose of cache is to reduce disk I/O.
- Cache systems could be rudimentary map structures or robust systems with a cache expiration policy.
- Caching is a popular strategy employed at all levels of a computer software stack to boost performance.
- Operating systems, databases, middleware components, and applications use caching.

BerkeleyDB and Memcached API

- As the NoSQL movement has gathered momentum, a number of key/value pair data stores have emerged.
- Some of these newer stores build on the Memcached API, some use Berkeley DB as the underlying storage, and a few others provide alternative solutions built from scratch.
- Many of these key/value pairs have APIs that allow get-and-set mechanisms to get and set values.
- A few, like Redis (http://redis.io/), provide richer abstractions and powerful APIs.
- Redis could be considered as a data structure server because it provides data structures like string (character sequences), lists, and sets, apart from maps.
- Also, Redis provides a very rich set of operations to access data from these different types of data structures.

Membase (Proposed to be merged into Couchbase, gaining features from CouchDB after the creation of Couchbase, Inc.)

- Official Online Resources — `www.membase.org/`.
- History — Project started in 2009 by NorthScale, Inc. (later renamed as Membase). Zygna and NHN have been contributors since the beginning. Membase builds on Memcached and supports Memcached's text and binary protocol. Membase adds a lot of additional features on top of Memcached. It adds disk persistence, data replication, live cluster reconfiguration, and data rebalancing. A number of core Membase creators are also Memcached contributors.
- Technologies and Language — Implemented in Erlang, C, and C++.
- Access Methods — Memcached-compliant API with some extensions. Can be a drop-in replacement for Memcached.
- Open-Source License — Apache License version 2.
- Who Uses It — Zynga, NHN, and others.

Kyoto Cabinet

- Official Online Resources — `http://fallabs.com/kyotocabinet/`.
- History — Kyoto Cabinet is a successor of Tokyo Cabinet (`http://fallabs.com/tokyocabinet/`). The database is a simple data file containing records; each is a pair of a key and a value. Every key and value are serial bytes with variable length.
- Technologies and Language — Implemented in C++.
- Access Methods — Provides APIs for C, C++, Java, C#, Python, Ruby, Perl, Erlang, OCaml, and Lua. The protocol simplicity means there are many, many clients.
- Open-Source License — GNU GPL and GNU LGPL.
- Who Uses It — Mixi, Inc. sponsored much of its original work before the author left Mixi to join Google. Blog posts and mailing lists suggest that there are many users but no public list is available.

Redis

- Official Online Resources — `http://redis.io/`.
- History — Project started in 2009 by Salvatore Sanfilippo. Salvatore created it for his startup LLOOGG (`http://lloogg.com/`). Though still an independent project, Redis primary author is employed by VMware, who sponsor its development.
- Technologies and Language — Implemented in C.
- Access Methods — Rich set of methods and operations. Can access via Redis command-line interface and a set of well-maintained client libraries for languages like Java, Python, Ruby, C, C++, Lua, Haskell, AS3, and more.
- Open-Source License — BSD.
- Who Uses It — Craigslist.

Amazon Dynamo

- The three key/value pairs listed here are nimble, fast implementations that provide storage for realtime data, temporary frequently used data, or even full-scale persistence.
- The key/value pairs listed so far provide a strong consistency model for the data it stores.
- However, a few other key/value pairs emphasize availability over consistency in distributed deployments.
- Many of these are inspired by Amazon's Dynamo, which is also a key/value pair.
- Amazon's Dynamo promises exceptional availability and scalability, and forms the backbone for Amazon's distributed fault tolerant and highly available system.
- Apache Cassandra, Basho Riak, and Voldemort are opensource implementations of the ideas proposed by Amazon Dynamo.
- Amazon Dynamo brings a lot of key high-availability ideas to the forefront.
- The most important of the ideas is that of eventual consistency.
- Eventual consistency implies that there could be small intervals of inconsistency between replicated nodes as data gets updated among peer-to-peer nodes.
- Eventual consistency does not mean inconsistency.
- It just implies a weaker form of consistency than the typical ACID type consistency found in RDBMS.

Cassandra

- Official Online Resources — `http://cassandra.apache.org/`.
- History — Developed at Facebook and open sourced in 2008, Apache Cassandra was donated to the Apache foundation.
- Technologies and Language — Implemented in Java.
- Access Methods — A command-line access to the store. Thrift interface and an internal Java API exist. Clients for multiple languages including Java, Python, Grails, PHP, .NET. and Ruby are available. Hadoop integration is also supported.
- Query Language — A query language specification is in the making.
- Open-Source License — Apache License version 2.
- Who Uses It — Facebook, Digg, Reddit, Twitter, and others.

Voldemort

- Official Online Resources — `http://project-voldemort.com/`.
- History — Created by the data and analytics team at LinkedIn in 2008.
- Technologies and Language — Implemented in Java. Provides for pluggable storage using either Berkeley DB or MySQL.
- Access Methods — Integrates with Thrift, Avro, and protobuf (`http://code.google.com/p/protobuf/`) interfaces. Can be used in conjunction with Hadoop.
- Open-Source License — Apache License version 2.
- Who Uses It — LinkedIn.

Riak

- Official Online Resources — `http://wiki.basho.com/`.
- History — Created at Basho, a company formed in 2008.
- Technologies and Language — Implemented in Erlang. Also, uses a bit of C and JavaScript.
- Access Methods — Interfaces for JSON (over HTTP) and protobuf clients exist. Libraries for Erlang, Java, Ruby, Python, PHP, and JavaScript exist.
- Open-Source License — Apache License version 2.
- Who Uses It — Comcast and Mochi Media.

All three — Cassandra, Riak and Voldemort — provide open-source Amazon Dynamo capabilities. Cassandra and Riak demonstrate dual nature as far their behavior and properties go. Cassandra has properties of both Google Bigtable and Amazon Dynamo. Riak acts both as a key/value store and a document database.

3. DOCUMENT DATABASES

- Document databases are not document management systems.
- More often than not, developers starting out with NoSQL confuse document databases with document and content management systems.
- The word document in document databases connotes loosely structured sets of key/ value pairs in documents, typically JSON (JavaScript Object Notation), and not documents or spreadsheets (though these could be stored too).
- Document databases treat a document as a whole and avoid splitting a document into its constituent name/value pairs.
- At a collection level, this allows for putting together a diverse set of documents into a single collection.
- Document databases allow indexing of documents on the basis of not only its primary identifier but also its properties.
- A few different open-source document databases are available today but the most prominent among the available options are MongoDB and CouchDB.

MongoDB

- Official Online Resources — `www.mongodb.org`.
- History — Created at 10gen.
- Technologies and Language — Implemented in C++.
- Access Methods — A JavaScript command-line interface. Drivers exist for a number of languages including C, C#, C++, Erlang. Haskell, Java, JavaScript, Perl, PHP, Python, Ruby, and Scala.
- Query Language — SQL-like query language.
- Open-Source License — GNU Affero GPL (`http://gnu.org/licenses/agpl-3.0.html`).
- Who Uses It — FourSquare, Shutterfly, Intuit, Github, and more.

CouchDB

- Official Online Resources — `http://couchdb.apache.org` and `www.couchbase.com`. Most of the authors are part of Couchbase, Inc.
- History — Work started in 2005 and it was incubated into Apache in 2008.
- Technologies and Language — Implemented in Erlang with some C and a JavaScript execution environment.
- Access Methods — Upholds REST above every other mechanism. Use standard web tools and clients to access the database, the same way as you access web resources.
- Open-Source License — Apache License version 2.
- Who Uses It — Apple, BBC, Canonical, Cern, and more at `http://wiki.apache.org/couchdb/CouchDB_in_the_wild`.

4. GRAPH DATABASES

- So far we have listed most of the mainstream open-source NoSQL products.
- A few other products like Graph databases and XML data stores could also qualify as NoSQL databases.
- This class does not cover Graph and XML databases.
- However, we list the two Graph databases that may be of interest and something you may want to explore beyond this class: Neo4j and FlockDB:
- Neo4J is an ACID-compliant graph database. It facilitates rapid traversal of graphs.

Neo4j

- Official Online Resources — `http://neo4j.org`.
- History — Created at Neo Technologies in 2003. (Yes, this database has been around before the term NoSQL was known popularly.)
- Technologies and Language — Implemented in Java.
- Access Methods — A command-line access to the store is provided. REST interface also available. Client libraries for Java, Python, Ruby, Clojure, Scala, and PHP exist.
- Query Language — Supports SPARQL protocol and RDF Query Language.
- Open-Source License — AGPL.
- Who Uses It — Box.net.

FlockDB

- Official Online Resources — `https://github.com/twitter/flockdb`
- History — Created at Twitter and open sourced in 2010. Designed to store the adjacency lists for followers on Twitter.
- Technologies and Language — Implemented in Scala.
- Access Methods — A Thrift and Ruby client.
- Open-Source License — Apache License version 2.
- Who Uses It — Twitter.