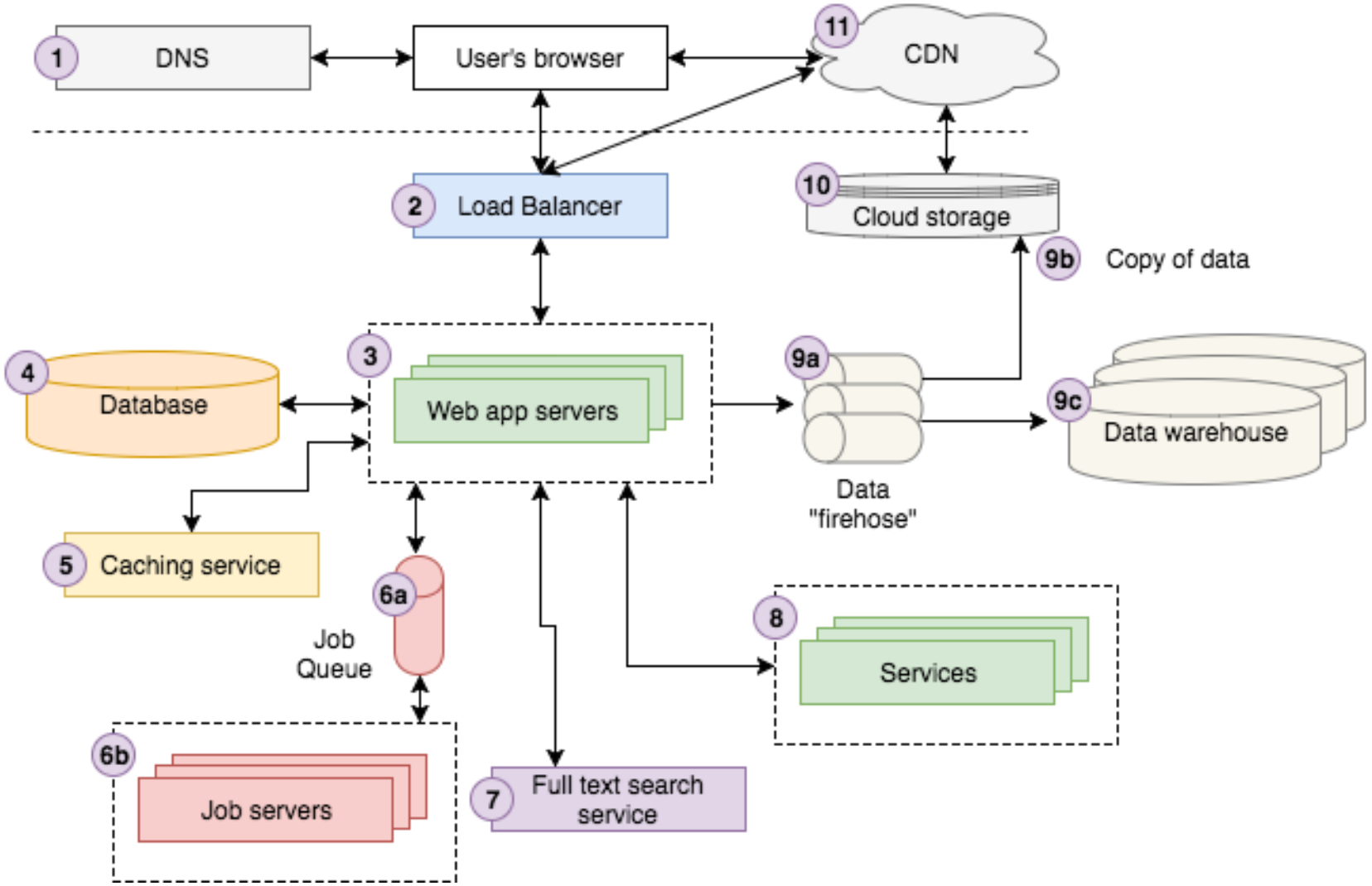
Web Server Architecture

Modern web application architecture



Components

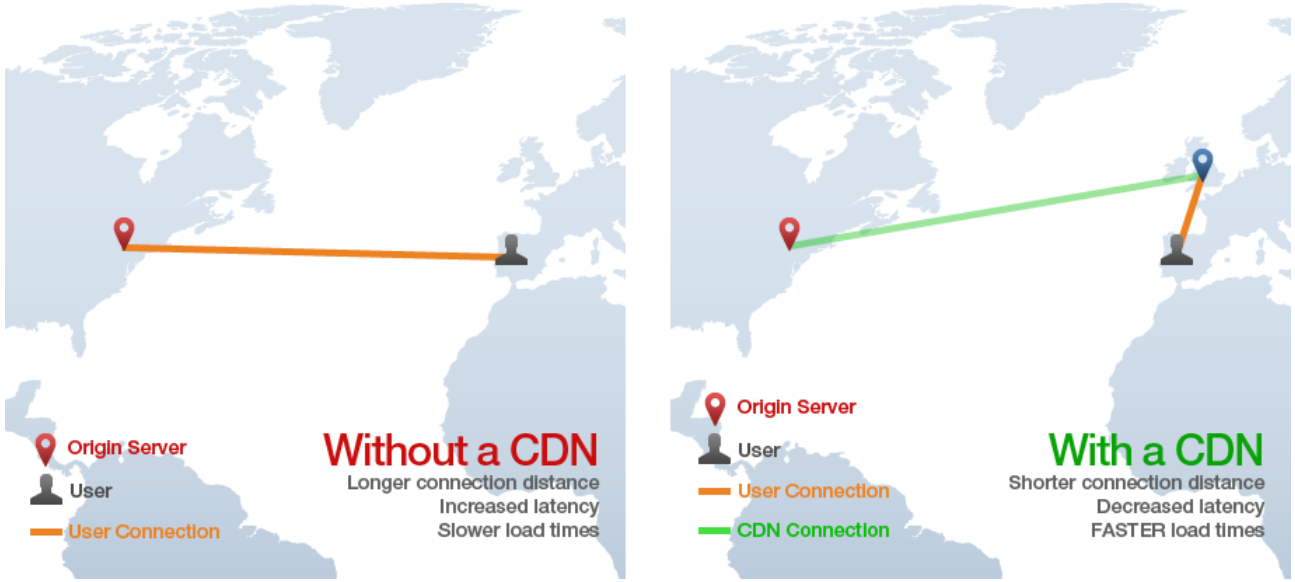
- **DNS**: DNS provides a key/value lookup from a domain name (e.g., google.com) to an IP address (e.g., 85.129.83.120)
- **Load Balancer**: They route incoming requests to one of many application servers that are typically clones / mirror images of each other and send the response from the app server back to the client.
- **Web Application Servers:** They execute the core business logic that handles a user's request and sends back HTML to the user's browser
- **Caching Service:** A caching service provides a simple key/value data store that makes it possible to save and lookup information in close to O(1) time

- **Job Queue & Servers:** Most web applications need to do some work asynchronously behind the scenes that's not directly associated with responding to a user's request.
- **Full-text Search Service:** the app returns the most "relevant" results. The technology powering this functionality is typically referred to as "full-text search", which leverages an inverted index

Documents (Photo titles)	
id	**title**
1	Man running in the mountains
2	Mountains with snow
3	Man running marathon

Inverted Index	
keyword	**photo_ids**
man	1, 3
running	1, 3
mountains	1, 2
snow	2
marathon	3

- **CDN** stands for “Content Delivery Network” and the technology provides a way of serving assets such as static HTML, CSS, Javascript, and images over the web much faster than serving them from a single origin server. It works by distributing the content across many “edge” servers around the world so that users end up downloading assets from the “edge” servers instead of the origin server.



How to reduce load?

Content Delivery Networks (CDN)

You can push static content (images/css/javascript/video) to external delivery networks which take the request load off of your servers

Add MoreWebservers

Technically this is true,but isn't generally going to be the best option.

Tune ExistingWebservers

If your server has enough resources you can experiment with increasing the maxclients that your webserver will allow.

Source: Steve French Digg.com

Webserver Load

Optimize Code and Code Use

- Cache Code
- Cache Data Outputs
- Profile Code

Scale the System Architecture

Source: Steve French Digg.com

DataAccess Load

Cache Data

Memcache - http://www.danga.com/memcached/

Test:

Fetch 100000 users from mysql and memcached

Caching Strategy

Cache OnAccess

Optimistic Cache Priming

Optimize Queries

Explain Queries

Add Indices

Sometimes many"simple" queries are better than a single complex one.

```
select * from users where userid in (1000,3002,53,100000,999,....);

   VS

for($i=1,$i <= 500;$i++)
   select * from users where userid = $i
```

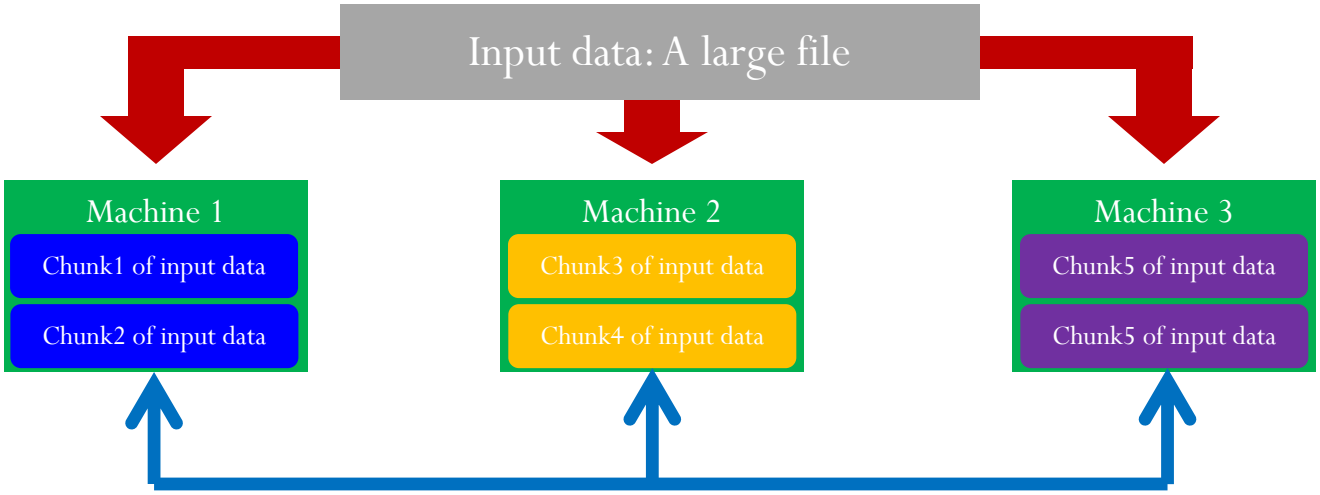
Source: Steve French Digg.com

Scaling Traditional Databases

- Traditional RDBMSs can be either scaled:
  - Vertically (or Up)
    - Can be achieved by hardware upgrades (e.g., faster CPU, more memory, or large r disk)
    - Limited by the amount of CPU, RAM and disk that can be configured on a single machine
  - Horizontally (or Out)
    - Can be achieved by adding more machines
    - Requires database *sharding* and probably *replication*
    - Limited by the Read-to-Write ratio and communication overhead

Why Sharding Data?

- Data is typically *sharded* (or *striped*) to allow for concurrent/ parallel accesses

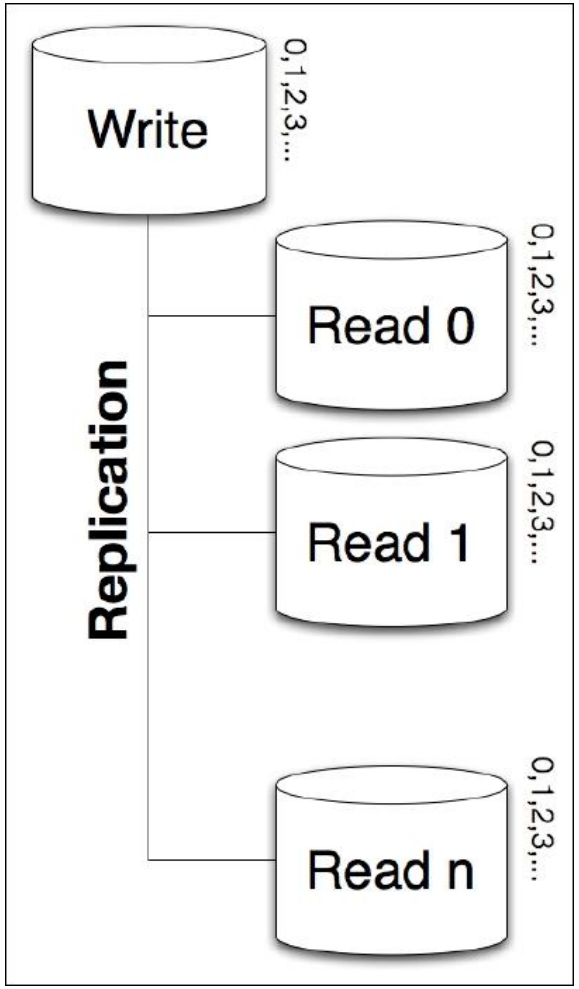


E.g., Chunks 1, 3 and 5 can be accessed in parallel

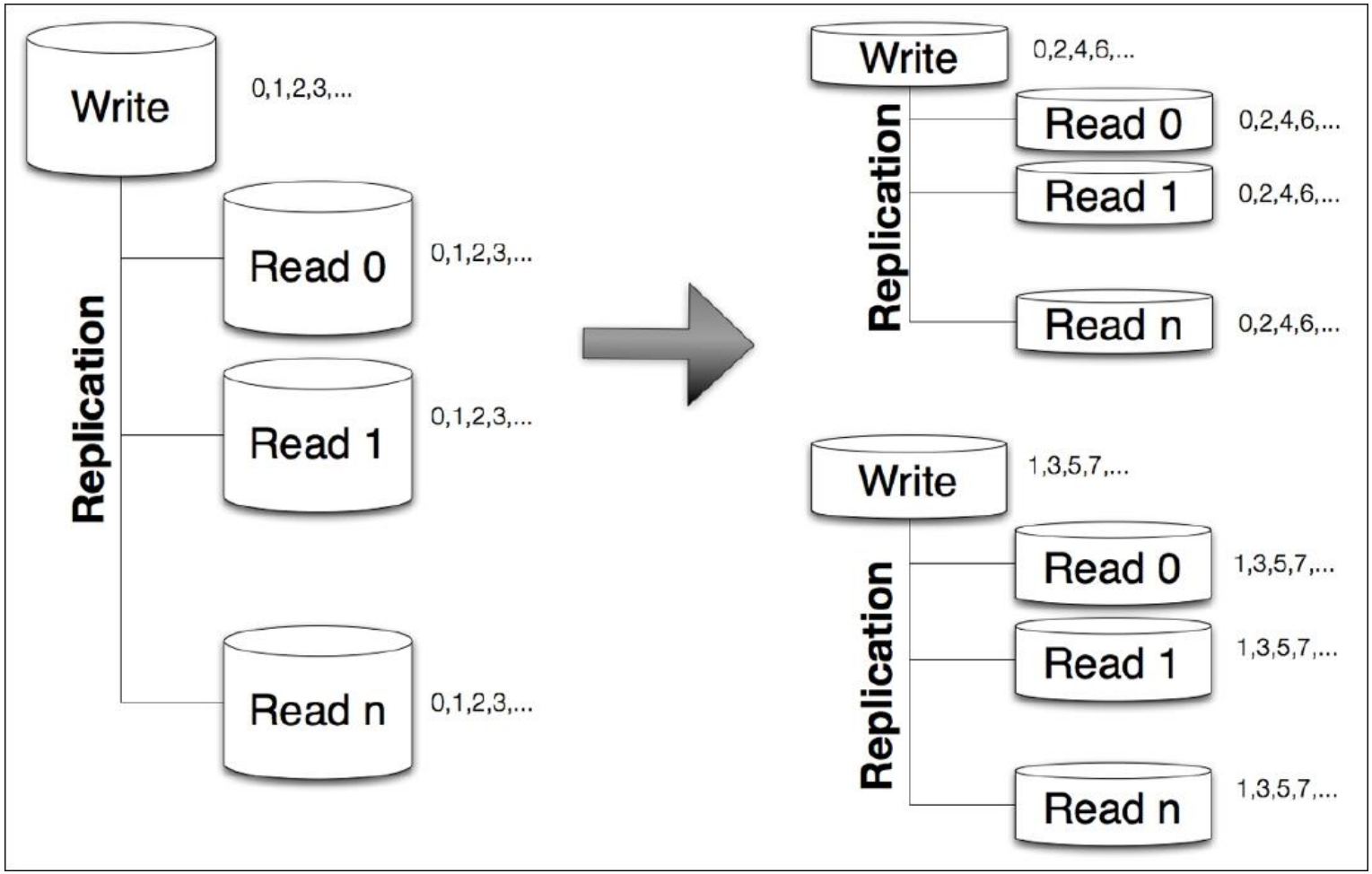
Why Replicating Data?

- Replicating data across servers helps in:
  - Avoiding performance bottlenecks
  - Avoiding single point of failures
  - And, hence, enhancing scalability and availability

Scaling Out Read Slaves



Horizontal Partitions (Sharding)



Vertical Partitions

```
create table users(
    id bigint not null auto_increment,
    username char(20),
    password char(50),
    secretquestion char(100),
    secretanswer char(100),
    favoritecolor char(25),
    website text,
    active enum('Y','N')
);
```

```
create table users(
    id bigint not null autoincrement,
    username char(20),
    password char(50),
    active enum('Y','N')
);

create table userdetails(
    userid bigint not null,
    secretquestion char(100),
    secretanswer char(100),
    favoritecolor char(25),
    website text
);
```

Data Denormalization

Benefits

Data is replicated to tables that need it.
Less JOIN operations

Drawbacks

Updating values takes additional operations,so this technique is best used on rarely-changing data.

```
create table users(
    id bigint not null autoincrement,
    username varchar(20),
    password varchar(50),
    secretquestion varchar(100),
    secretanswer varchar(100),
    active enum('Y','N')
);
create table stories(
    id bigint not null autoincrement,
    title varchar(100),
    description varchar(255),
    username varchar(20)
);
create table comments(
    userid bigint not null,
    comment_body varchar(255),
    username varchar(20)
);
```

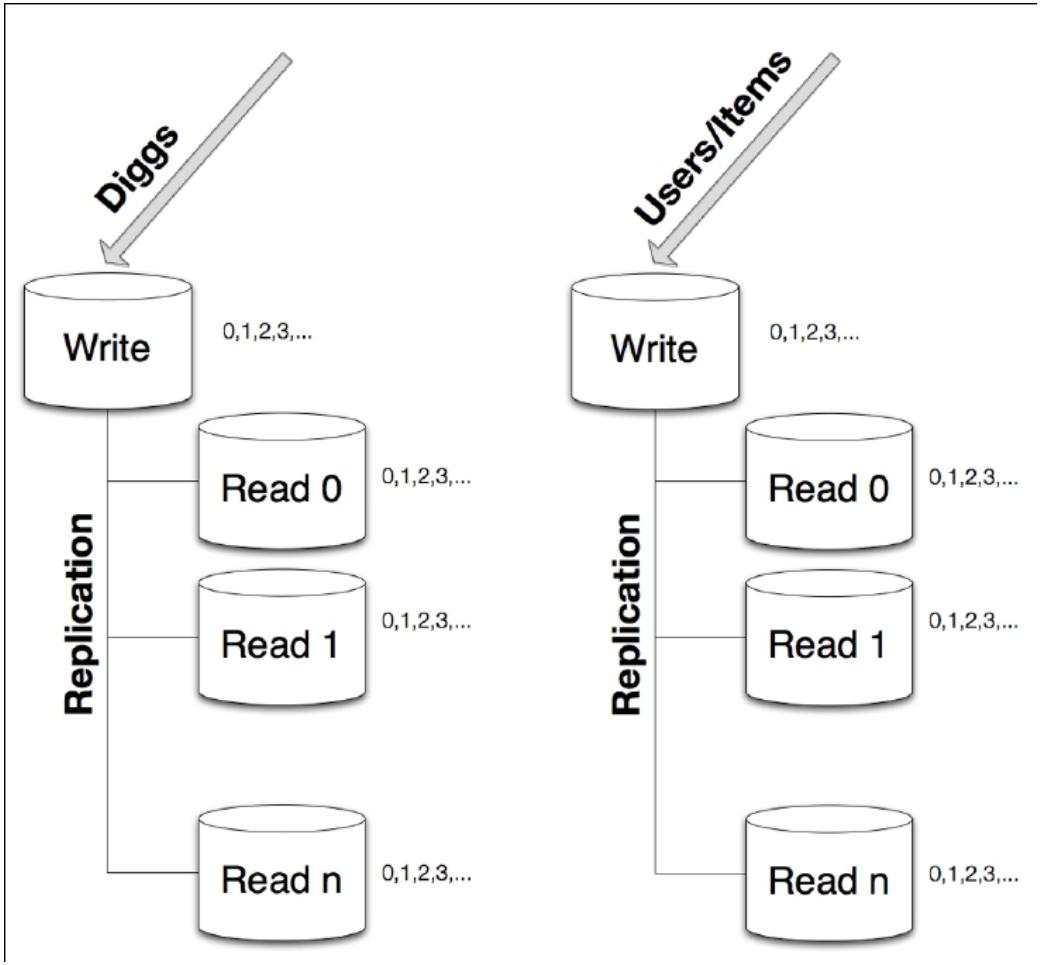
Scale the System Architecture -Write Traffic

Query-Specific Server Pools

Scale OutWritable Servers

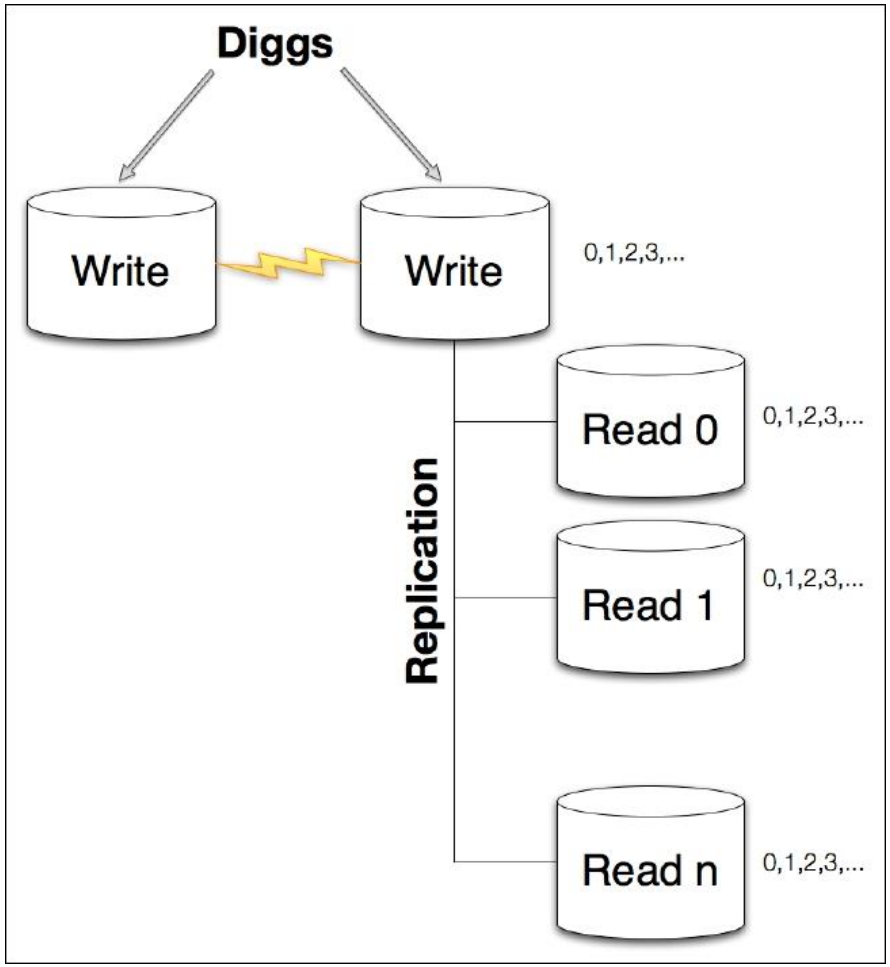
Horizontal Partitioning

Query-Specific Server Pools



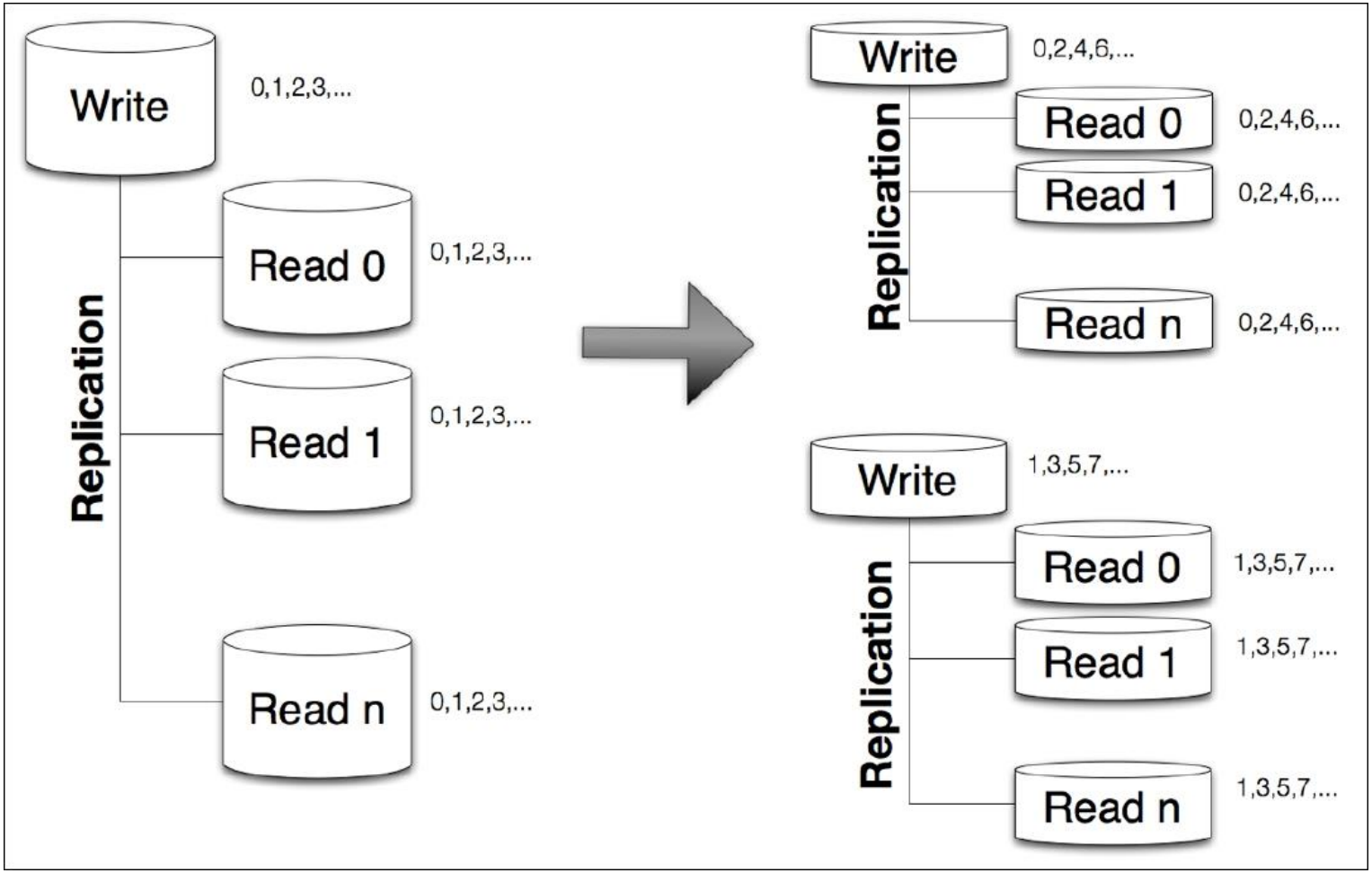
Source: Steve French Digg.com

Scale OutWritable Servers



Source: Steve French Digg.com

Horizontal Partitions (Sharding)



Source: Steve French Digg.com

Database Tuning

Key/Value Stores

Benefits:

- Faster
- Suited to horizontal scaling
- Redundancy
- Read/WriteTradeoffs

Drawbacks:

- Ad-hoc queries are almost impossible.

SQL Injection

- Username = ' or 1=1 --
  - The original statement looked like:
    'select * from users where username = '" + username + "' and pa
    ssword = '" + password + "'‘
    The result =
    select * from users where username = '' or 1=1 --' and password
    = ''

SQL Injection

- is a technique that exploits a security vulnerability occurring in the database layer of an application. The vulnerability is pre sent when user input is either incorrectly filtered for string li teral escape characters embedded in SQL statements or user input is not strongly typed and thereby unexpectedly execute d.

Prepared Statement

- Executing a statement takes time for database server
  - Parses SQL statement and looks for syntax errors
  - Determines optimal way to execute statement
    - Particularly for statements involving loading multiple tables
- Most database statements are similar in form
  - Example: Adding books to database
    - Thousands of statements executed
    - All statements of form:
    - **"SELECT * FROM books WHERE productCode = ____"**

Prepared Statement

```
  check = connection.prepareStatement("SELECT * FROM books WHERE productCode = ?");
  check.setString(1, productCode);
  books = check.executeQuery();
  if (books.next()) {
    RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/AddError.jsp");
    dispatcher.forward(request, response);
    return;
    }
  }
catch (SQLException e) { System.out.println("BAD QUERY"); }
```

Stored Procedure

- A stored procedure is a precompiled executable object that c ontains one or more SQL statements.
  - stored procedures are precompiled objects they execute faster a t the database server.
  - For the consecutive run it will run from the compiled stage and hence boost performance.

DB Encryption

- DB Encryption can be divided into Data-in-transit and Data-at-rest
- Encryption is useful as a last layer of defense (defense in depth). Should never be used as an alternative solution
- Encryption should be used only when needed
- Key Management is Key

Auditing

- Oracle-supplied auditing using AUDIT command. Results go to AUD$
- Trigger-based DML auditing
- Either way, DBA must monitor auditing table.

When to audit

- When should we audit Oracle users ?
  - Basic set of auditing measures all the time
  - Capture user access, use of system privileges, changes to the db schema (DDL)

    If company handles sensitive data (financial market, military, etc .) OR

    If there are suspicious activities concerning the DB or a user, sp ecific actions should be done.

How to optimize Queries?

- Less work → Faster query

- What is work for a query?
  - I/O — How many bytes did you read?
  - Shuffle — How many bytes did you pass to the next stage?
    - Grouping — How many bytes do you pass to each group?
  - Materialization — How many bytes did you write?
  - CPU work — User-defined functions (UDFs), functions

Don’t project unnecessary columns

- On how many columns are you operating?
- Excess columns incur wasted I/O and materialization
- Don’t SELECT * unless you need every field

Filter early and often using WHERE clauses

- On how many rows (or partitions) are you operating?
- Excess rows incur "waste" similar to excess columns

Do the biggest joins first (if you have to)

- Joins — In what order are you merging data?
  - Guideline — Biggest, Smallest, Decreasing Size Thereafter
  - Avoid self-join if you can, since it squares the nu mber of rows processed
- Consider your JOIN order, try to filter the sets pr e-JOIN

Low Cardinality GROUP BYs are faster

- Grouping — How much data are we grouping per-key for aggre gation?
- Guideline — Low-cardinality keys/groups → fast, high-cardinali ty → slower
- However, higher key cardinality (more groups) leads to more shu ffling; key skew can lead to increased tail latency
- Note: Get a count of your groups when trying to understand per formance

Built-in functions are faster than JavaScript UDFs

- Functions — What work are we doing on the data?
- Guideline — Some operators are faster than others; all are fa ster than JavaScript® UDFs
- Example — Exact COUNT(DISTINCT) is very costly, but APPROX_COUNT_DISTINCT is very fast
- **Use SQL analytic functions** - The Oracle analytic functio ns can do multiple aggregations (e.g. rollup by cube) with a si ngle pass through the tables, making them very fast for repor ting SQL

ORDER on the outermost query

- Sorting—How many values do you need to sort?
- Filtering first reduces the number of values you need to sort
- Ordering first forces you to sort the world

Other Minor SQL Tuning

- **Avoid the LIKE predicate** = Always replace a "like" with an equality, when appropriate.
- **Never mix data types** - If a WHERE clause column predicate is numeric, do not to use quotes. For char index columns, always use quotes. There are mixed data type predic ates: where cust_nbr = "123"
- **Use those aliases** - Always use table aliases when referencing columns
- Sometimes you may have more than one subqueries in your main query. Try to minimize the number of subquery block in your query.

Databases Security – General Strategies

- Principle of Least Privilege!
- Stay up-to-date on patches
- Remove/disable unneeded default accounts
- Firewalling/Access Control
- Running Database processes under dedicated non-privilege d account.
- Password Security
- Disable unneeded components

Principle of Least Privilege

- If X service doesn't need access to all tables in Y database... then don't give it access to all tables.
  - Example: A web application that reads a list of people from a database and lists them on a website. The database also contains sensitive information about those people.
- Do not give accounts privileges that aren't needed
  - Unneeded privileges to accounts allow more opportunity for privilege escalation attacks.

Stored Procedures, Triggers

- Stored Procedures and Triggers can lead to privilege escala tion and compromise. Be sure to be thinking about securit y implications when allowing the creation of, and creating these.