PART 0. INTRODUCTION

2. DATA PROCESSING PARADIGMS FOR BIG DATA

Sangmi Lee Pallickara
Computer Science, Colorado State University
http://www.cs.colostate.edu/~cs535

FAQs

- Slides are available on the course web
- Wait list
- Term project topics

This material is built based on


Lambda Architecture

Why we are looking at Lambda Architecture

- To perform large-scale analytics over voluminous data, we need a high-level architecture that provides,
  - Robustness
  - Fault-tolerant: Both against hardware failures and human mistakes
  - Support for a wide range of workloads and use cases
    - Low-latency reads and updates
    - Batch analytics jobs
    - Scalability
    - Scale-out capabilities with minimal maintenance

Typical problems for scaling traditional databases

- Suppose that the application should track the number of page views for any URL a customer wishes to track
  - The customer’s web page pings the application’s web server with its URL every time a pageview is received
  - Application tells you top 100 URLs by number of pageviews

<table>
<thead>
<tr>
<th>Id (integer)</th>
<th>User_id (integer)</th>
<th>url (varchar255)</th>
<th>Pageviews bigint</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Direct access
- Direct access from Web server to the backend DB cannot handle the large amount of frequent write requests
- Timeout errors

Scaling with a queue
- Batch many increments in a single request
- What if your data amount increases even more?
  - Your worker cannot keep up with the writes
- What if you add more workers?
  - Again, the Database will be overloaded

Scaling by sharding the database
- Horizontal partitioning or sharding of database
  - Uses multiple database servers and spreads the table across all the servers
  - Chooses the shard for each key by taking the hash of the key modded by the number of shards
- What if your current number of shards cannot handle your data?
  - Your mapping script should cope with new set of shards
  - Application and data should be re-organized

Other issues
- Fault-tolerance issues
  - What if one of the database machines is down?
  - A portion of the data is unavailable
- Corruption issues
  - What if your worker code accidentally generated a bug and stored the wrong number for some of the data portions

How will Big Data techniques help?
- The databases and computation systems used in Big Data applications are aware of their distributed nature
- Sharding and replications will be considered as a fundamental component in the design of Big Data systems
- Data is dealt as immutable
  - Users will mutate data continuously
  - The raw pageview information is not modified
- Applications will be designed in different ways

Lambda Architecture
- Big Data data processing architecture as a series of layers
  - Batch layer
  - Serving layer
  - Speed layer

http://www.cs.colostate.edu/~cs535
Batch layer
- Batch layer
  - Precomputes results using distributed processing system
    - The component that performs the batch view processing
      \[ \text{batch view} = \text{function(accumulated data)} \]
  - Stores an immutable, constantly growing master dataset
  - Computes arbitrary functions on that dataset
    - Batch-processing systems
      - e.g. Hadoop, Spark, TensorFlow

Generating batch views

Serving layer
- The batch layer emits batch view as the result of its functions
  - These views should be loaded somewhere and queried
  - Specialized distributed database that loads in a batch view and makes it possible to do random reads on it
  - Batch update and random reads should be supported
    - e.g. BigQuery, ElephantDB, Dynamo, MongoDB, Cassandra

Speed layer
- Is there any data not represented in the batch view?
  - Data arrives while the precomputation is running
    - With fully real-time data system
  - Speed layer looks only at recent data
    - Whereas the batch layer looks at all the data at once
      - \[ \text{realtime view} = \text{function(earliest view, new data)} \]

Relevance of Data
- Once the data arrives at the serving layer, the corresponding results in the real-time views are no longer needed
  - You can discard pieces of the real-time views

http://www.cs.colostate.edu/~cs535
**Lambda architecture**

- **Process Stream**: Sends input to both the Batch layer and the Speed layer.
- **Batch layer**: Performs batch processing and updates batch views.
- **Speed layer**: Updates views in real-time.
- **Serving layer**: Combines views and serves them.
- **New data stream**: Continuously feeds new data to the process stream.

**Example with Lambda architecture**

- Web analytics application tracking the number of pageviews over a range of days.
  - The speed layer keeps its own separate view of [url, day].
  - Updates its views by incrementing the count in the view whenever it receives new data.
  - The batch layer recomputes its views by counting the pageviews.
  - To resolve the query, you query **both the batch and realtime views**.
  - With satisfying ranges.
  - Sum up the results.

**Extended examples and use cases**

- Full recomputation vs. partial recomputation
  - e.g. using Bloom filters
- Recomputational algorithms vs. incremental algorithms
  - Additive algorithms vs. approximation algorithms
    - e.g. HyperLogLog for count-distinct problem

**Composing algorithms**

- The batch/speed layer will split your data.
  - The exact algorithm on the batch layer.
  - An approximate algorithm on the speed layer.
  - The batch layer repeatedly overrides the speed layer.
  - The approximation gets corrected.
  - Eventual accuracy.

**Example of Cardinality estimation**

- Cardinality estimation
  - Count-distinct problem: finding number of distinct elements
  - Counting **exact unique counts** in the batch layer.
  - A **HyperLogLog** as an approximation in the speed layer.
  - Batch layer corrects what’s computed in the speed layer.
  - Eventual accuracy.

**Recent trends in technology** (1/3)

- Physical limits of how fast a single CPU can go.
  - Parallelize computation to scale to more data.
  - **Scale-out** solution.
  - Elastic clouds
    - Infrastructure as a Service (IaaS)
    - Rent hardware on demand rather than owning your hardware.
    - Increase and decrease the size of your cluster nearly instantaneously.
    - Simplifies system administration.

---

http://www.cs.colostate.edu/~cs535
Recent trends in technology (2/3)

- Open source ecosystem for Big Data
  - Batch computation systems
    - Hadoop
    - Spark
  - Serialization frameworks
    - Serializes an object into a byte array from any language
    - Deserialize that byte array into an object in any language
    - Thrift, Protocol Buffers, and Avro

Recent trends in technology (3/3)

- Open source ecosystem for Big Data - cont.
  - Random-access NoSQL databases
    - Sacrifice the full expressiveness of SQL
    - Specializes in certain kinds of operations
    - Cassandra, Hbase, MongoDB, etc.
  - Messaging/queuing systems
    - Sends and consumes messages between processes in a fault-tolerant manner
    - Apache Kafka
  - Real-time computation system
    - High throughput, low latency, stream-processing systems
    - Apache Storm

Mapping recent technologies

Mapping course components

Kappa architecture

BSQ

- Lambda architecture vs. Kappa architecture
  - What is the difference between Lambda architecture and Kappa architecture?
  - Use case of Lambda architecture?
  - User case of Kappa architecture?
PART 0. INTRODUCTION

3. DATA MODEL FOR BIG DATA

: APACHE THRIFT

Sangmi Lee Pallickara
Computer Science, Colorado State University
http://www.cs.colostate.edu/~cs535

This material is built on


Why we are looking at Thrift

- Applications and services involve multiple, distributed components
- Possibly developed in different languages
- Communicate very intensively with each other
- The wire formats used for data interchange may evolve over time

- Goal
- Support this interoperability and evolution of wire formats
- But without compromising on performance (i.e., speed)

Apache Thrift

- A framework for creating interoperable and scalable services
- Data serialization framework
- Originally developed at Facebook
- Now an Apache project
- Users can create their services via a simple Interface definition language (IDL)
- Consumable and serviceable by numerous languages
- Codes for clients and servers are automatically generated
- Binary communication protocol
- Compact size

Thrift Architecture


You can change the protocol and transport without regenerating code

http://www.cs.colostate.edu/~cs535