CompSci 516
Data Intensive Computing Systems

Lecture 21 (optional)

NoSQL systems

Instructor: Sudeepa Roy

optional reading

Key-Value Stores

Basic Idea

optional reading

- The simplest data stores
- Use a data model similar to the popular memcached distributed in-memory cache
  - with a single key-value index for all the data
- Unlike memcached, these systems generally provide a persistence mechanism and additional functionality
  - e.g. replication, versioning, locking, transactions, sorting, and/or other features
  - The client interface provides inserts, deletes, and index lookups
- Like memcached, none of these systems offer secondary indices or keys

Project Voldemort

optional reading

- An advanced key-value store, written in Java
- Open source, with substantial contributions from LinkedIn
- Voldemort details:
  - provides multi-version concurrency control (MVCC) for updates
  - Updates replicas asynchronously, so it does not guarantee consistent data
  - However, it can guarantee an up-to-date view if you read a majority of replicas.
  - supports optimistic locking for consistent multi-record updates: if updates conflict with any other process, they can be backed out
  - supports automatic sharding of data (consistent hashing)
  - can store data in RAM, but also supports a BerkeleyDB and Random Access File storage engine
  - supports lists and records in addition to simple scalar values

Consistent Hashing

optional reading

- Consistent hashing is used to distribute data around a ring of nodes
- data hashed to node K is replicated on node K+1 ... K+n
  - where n is the desired number of extra copies (often n=1)
  - once data partitioning is set up, its operation is transparent
  - Nodes can be added or removed from a database cluster, and the system adapts automatically
- Voldemort automatically detects and recovers failed nodes

optional reading

Memcached – Membrain - Membase

- The memcached open-source distributed in-memory indexing system has been enhanced
  - to include features analogous to the other key-value stores: persistence, replication, high availability, dynamic growth, backup, …
  - Without persistence or replication, memcached does not really qualify as a "data store"
  - However, Membrain and Membase certainly do, and these systems are also compatible with existing memcached applications

Now CouchDB + Membase = CouchBase – see CouchBase.com

optional reading

Other key-value store systems

- Riak
- Redis
- Scalaris
- Tokyo Cabinet

- Optional:
  - see the Cattell's paper and the references therein for the above for details

optional reading

Summary: Key-Value Store

- All the key-value stores support insert, delete, and lookup operations
- All of these systems provide scalability through key distribution over nodes
- Voldemort, Riak, Tokyo Cabinet, and enhanced memcached systems can store data in RAM or on disk, with storage add-ons
  - The others store data in RAM, and provide disk as backup, or rely on replication and recovery so that a backup is not needed
- Scalaris and enhanced memcached systems use synchronous replication
  - the rest use asynchronous
- Scalaris and Tokyo Cabinet implement transactions
  - the others do not.
- Voldemort and Riak use multi-version concurrency control
  - the others use locks

optional reading

Document Stores

optional reading

Basic Idea

- Document stores support more complex data than the key-value stores
- "document store" may be confusing
  - these systems could store "documents" in the traditional sense (articles, Microsoft Word files, etc.)
  - but a document in these systems can be any kind of "pointerless object"
- Unlike the key-value stores, these systems generally support
  - secondary indexes
  - multiple types of documents (objects) per database, and
  - nested documents or lists
- Like other NoSQL systems, the document stores do not provide ACID transactional properties

optional reading

SimpleDB - 1

- Part of Amazon's proprietary cloud computing service (since ~2007)
  - along with their Elastic Compute Cloud (EC2) and their Simple Storage Service (S3) on which it is based
- Simple model
  - has Select, Delete, GetAttributes, and PutAttributes operations on documents
  - no nested documents (simpler)

optional reading

SimpleDB - 2

- Like most of the new systems,
  - supports eventual consistency, not transactional consistency
  - does asynchronous replication

- Unlike key-value datastores, and like the other document stores
  - supports more than one grouping in one database:
  - documents are put into domains, which support multiple indexes
  - select operations are on one domain, and specify a conjunction of constraints on attributes in the form:
  - "select <attributes> from <domain> where <list of attribute value constraints> "
  - Different domains may be stored on different Amazon nodes.

optional reading

SimpleDB - 3

- Does not automatically partition data over servers
  - Some horizontal scaling can be achieve by reading any of the replicas, if you don't care about having the latest version
  - Writes do not scale - they must go asynchronously to all copies of a domain
  - If customers want better scaling, they must do so manually by "sharding" themselves

optional reading

SimpleDB - 4

- SimpleDB is a "pay as you go" proprietary solution from Amazon
  - Some built-in constraints (as of 2011), some of which are quite limiting: a 10 GB maximum domain size, a limit of 100 active domains, a 5 second limit on queries, and so on
  - Amazon doesn't license SimpleDB source or binary code to run on your own servers
  - SimpleDB does have the advantage of Amazon support and documentation

optional reading

CouchDB: Data Model

- CouchDB has been an Apache project since early 2008
- A CouchDB "collection" of documents is similar to a SimpleDB domain
  - but the CouchDB data model is richer
- Collections comprise the only schema in CouchDB
  - secondary indexes must be explicitly created on fields in collections
- A document has field values
  - can be scalar (text, numeric, or boolean) or compound (a document or list)

optional reading

CouchDB: Queries

- Queries are done with what CouchDB calls "views"
- The indexes are B-trees
  - so the results of queries can be ordered or value ranges
- Queries can be distributed in parallel over multiple nodes using a map-reduce mechanism
- However, CouchDB's view mechanism puts more burden on programmers than a declarative query language

optional reading

CouchDB: Scalability

- Like SimpleDB, CouchDB achieves scalability through asynchronous replication, not through sharding
- Reads can go to any server if you don't care about having the latest values
  - updates must be propagated to all the servers
- However, a new project called CouchDB Lounge has been built to provide sharding on top of CouchDB
  - see: http://code.google.com/p/couchdb-lounge/
- Now: CouchDB + Membase = Couchbase
  - to have CouchDB's richer data model + speed/elastic scalability of Membase

optional reading

CouchDB: Consistency

- Like SimpleDB, CouchDB does not guarantee consistency
- Unlike SimpleDB, each client does see a self-consistent view of the database with repeatable reads
  - CouchDB implements multi-version concurrency control on individual documents
- CouchDB will notify an application if someone else has updated the document since it was fetched
  - The application can then try to combine the updates, or can just retry its update and overwrite
- CouchDB also provides durability on system crash
  - All updates (documents and indexes) are flushed to disk on commit, by writing to the end of a file
  - Together with the MVCC mechanism, CouchDB's durability thus provides ACID semantics at the document level

optional reading

MongoDB

- MongoDB is an open source document store written in C++
- Has some similarities to CouchDB
  - provides indexes on collections
  - lockless
  - provides a document query mechanism

optional reading

MongoDB vs. CouchDB

- MongoDB supports automatic sharding
  - distributing documents over servers
- Replication in MongoDB is mostly used for failover
  - not for (dirty read) scalability as in CouchDB
- MongoDB does not provide the global consistency of a traditional DBMS
  - but you can get local consistency on the up-to-date primary copy of a document
- MongoDB supports dynamic queries with automatic use of indices, like RDBMSs
  - In CouchDB, data is indexed and searched by writing map-reduce views
  - MongoDB also supports map-reduce – helps complex aggregations across docs
- CouchDB provides MVCC on documents, while MongoDB provides atomic operations on fields.

optional reading

MongoDB: Atomic Ops on Fields

- The update command supports "modifiers" that facilitate atomic changes to individual values
  - $set sets a value
  - $inc increments a value
  - $push appends a value to an array
  - $pushAll appends several values to an array
  - $pull removes a value from an array, and $pullAll removes several values from an array
- Since these updates normally occur "in place", they avoid the overhead of a return trip to the server
- There is an "update if current" convention for changing a document only if field values match a given previous value
- MongoDB supports a findAndModify command to perform an atomic update and immediately return the updated document
  - useful for implementing queues and other data structures requiring atomicity

optional reading

MongoDB: Index

- MongoDB indices are explicitly defined using an ensureIndex call
  - any existing indices are automatically used for query processing
- To find all products released last year (2015) or later costing under $100 you could write:
- db.products.find(
  {released: {$gte: new Date(2015, 1, 1,)}, price
  {'$lte': 100},})

optional reading

MongoDB: Data

- MongoDB stores data in a binary JSON-like format called BSON
  - BSON supports boolean, integer, float, date, string and binary types
  - MongoDB can also support large binary objects, eg. images and videos. These are stored in chunks that can be streamed back to the client for efficient delivery.

optional reading

MongoDB: Replication

- MongoDB supports master-slave replication with automatic failover and recovery
  - Replication (and recovery) is done at the level of shards
  - Replication is asynchronous for higher performance, so some updates may be lost on a crash

optional reading

Summary

- Another Document Store system : Terrastore
- The document stores are schema-less, except for
  - attributes (which are simply a name, and are not pre- specified)
  - collections (which are simply a grouping of documents), and
  - indexes defined on collections (explicitly defined, except in SimpleDB)
  - There are some differences in their data models, e.g. SimpleDB does not allow nested documents
- The document stores are very similar but use different terminology
  - e.g. a SimpleDB Domain = CouchDB Database = MongoDB Collection (= Terrastore Bucket)
  - SimpleDB calls documents "items"
  - an attribute is a field in CouchDB, or a key in MongoDB (or Terrastore)

optional reading

Summary

- Unlike the key-value stores, the document stores provide a mechanism to query collections based on multiple attribute value constraints
  - However, CouchDB does not support a non-procedural query language: it puts more work on the programmer and requires explicit utilization of indices
- The document stores generally do not provide explicit locks
  - have weaker concurrency and atomicity properties than traditional ACID-compliant databases
  - They differ in how much concurrency control they do provide.
- Documents can be distributed over nodes in all of the systems
  - but scalability differs
  - All of the systems can achieve scalability by reading (potentially) out-of-date replicas
  - MongoDB (and Terrastore) can obtain scalability without that compromise, and can scale writes as well, through automatic sharding and atomic operations on documents
  - CouchDB might be able to achieve this write-scalability with the help of the new CouchDB Lounge code

optional reading

Extensible Record Stores

optional reading

Basic Idea

- Seem to have been motivated by Google's success with BigTable
  - still the recent extensible record stores cannot come close to BigTable's scalability
- Basic data model is rows and columns
- Basic scalability model is splitting both rows and columns over multiple nodes
- Rows are split across nodes through sharding on the primary key
  - They typically split by range rather than a hash function
  - This means that queries on ranges of values do not have to go to every node
- Columns of a table are distributed over multiple nodes by using "column groups"
  - a way for the customer to indicate which columns are best stored together
- Both horizontal and vertical partitioning can be used simultaneously on the same table

optional reading

Column Groups and Rows

- must be pre-defined with the extensible record stores
  - But, not a big constraint, as new attributes can be defined at any time
- Rows are analogous to documents:
  - they can have a variable number of attributes (fields)
  - the attribute names must be unique
  - rows are grouped into collections (tables), and
  - an individual row's attributes can be of any type

optional reading

HBASE

- HBase is an Apache project written in Java (initially released in 2007)
- Patterned directly after BigTable
- HBase uses the Hadoop distributed file system in place of the Google file system. (Lecture 14)
  - It puts updates into memory and periodically writes them out to files on the disk
  - The updates go to the end of a data file, to avoid seeks
  - The files are periodically compacted
  - Updates also go to the end of a write ahead log, to perform recovery if a server crashes

optional reading

HBASE

- Row operations are atomic, with row-level locking and transactions
  - These use optimistic concurrency control, aborting if there is a conflict with other updates.
- There is multiple master support, to avoid a single point of failure
  - MapReduce support allows operations to be distributed efficiently.
- HBase's log-structured merge file indexes allow fast range queries and sorting
- The support for transactions is attractive, and unusual for a NoSQL system

optional reading

Cassandra

- Cassandra is written in Java, and used under Apache licensing. Originally open sourced by Facebook in 2008
  - It was designed by a Facebook engineer and a Dynamo engineer, and is described as a marriage of Dynamo and BigTable
  - Cassandra is used by Facebook as well as other companies, so the code is reasonably mature.
  - Client interfaces are created using Facebook's Thrift framework: http://incubator.apache.org/thrift/

- Cassandra is similar to the other extensible record stores in its data model and basic functionality
  - Has column groups, updates are cached in memory and then flushed to disk, and the disk representation is periodically compacted. It does partitioning and replication. Failure detection and recovery are fully automatic
- However, Cassandra has a weaker concurrency model than some other systems
  - there is no locking mechanism, and replicas are updated asynchronously

optional reading

Cassandra

- Has the concept of a "supercolumn" that provides another level of grouping within column groups
- Databases (called keyspaces) contain column families
  - A column family contains either supercolumns or columns (not a mix of both).
  - Supercolunns contain columns
  - As with the other systems, rows are variable length and are not constrained by a table schema
- Cassandra uses an ordered hash index
  - gives most of the benefit of both hash and B-tree indexes
  - However, sorting would still be slower than with B-trees.
- For applications where Cassandra's eventual- consistency model is not adequate, "quorum reads" of a majority of replicas provide a way to get the latest data
  - Cassandra writes are atomic within a column family
  - Some support for versioning and conflict resolution

optional reading

Summary

- Other systems:
  - HyperTable, Yahoo's PNUT
- The extensible record stores are mostly patterned after BigTable
  - They are all similar, but differ in concurrency mechanisms and other features
- Cassandra focuses on "weak" concurrency (via MVCC) and HBase and HyperTable on "strong" consistency (via locks and logging)

optional reading

Scalable Relational Systems

optional reading

Basic Idea

- Some RDBMSs are expected to provide scalability comparable with NoSQL data stores
- But, with two provisos:
  - Use small-scope operations: Operations that span many nodes, e.g. joins over many tables, will not scale well with sharding
  - Use small-scope transactions: Likewise, transactions that span many nodes are going to be very inefficient, with the communication and 2PC overhead
- NOSQL systems make these two impossible
  - scalable RDBMS allows them, but penalizes a customer for these operations
- Have higher-level SQL language and ACID properties
  - but pay a price when they span nodes

optional reading

Examples

- MySQL cluster
  - since 2004, uses shared-nothing
- VoltDB
- Clustrix
- ScaleDB
- ScaleBase
- NimbusDB

optional reading

VoltDB

- An open-source RDBMS designed for high performance (per node) as well as scalability
  - The scalability and availability features are competitive with MySQL Cluster and the NoSQL systems in this paper
- Tables are partitioned over multiple servers
  - clients can call any server
  - The distribution is transparent to SQL users, but the customer can choose the sharding attribute
- Alternatively, selected tables can be replicated over servers
  - e.g. for fast access to read-mostly data.
- Shards are replicated
  - so that data can be recovered in the event of a node crash
  - Database snapshots are also supported, continuous or scheduled

optional reading

VoltDB eliminates nearly all "waits" in SQL - 1

- The system is designed for a database that fits in (distributed) RAM on the servers
  - so that the system need never wait for the disk
- Indexes and record structures are designed for RAM rather than disk
  - the overhead of a disk cache/buffer is eliminated as well
- Performance will be very poor if virtual memory overflows RAM
  - but the gain with good RAM capacity planning is substantial
- SQL execution is single-threaded for each shard
  - using a shared-nothing architecture
  - so there is no overhead for multi-thread latching.

optional reading

VoltDB eliminates nearly all "waits" in SQL - 2

- All SQL calls are made through stored procedures
  - a group of SQL statements compiled into a single execution plan
  - each stored procedure is one transaction
  - i.e. If data is sharded to allow transactions to be executed on a single node, then no locks are required, and therefore no waits on locks
  - Transaction coordination is likewise avoided.
  - Stored procedures are compiled to produce code comparable to the access level calls of NoSQL systems

optional reading

Some other non-relational DBMS

- Graph Database Systems
  - e.g. Neo4j and OrientDB
  - provides efficient distributed storage and queries
- Object-oriented database systems
  - e.g. Versant
  - materialize a graph of objects as programming language objects
- Distributed object-oriented stores
  - similar to above
  - distribute object graphs in-memory on multiple servers
  - e.g. GemFire