CONCURRENCY ISSUES

Stephen Schaub

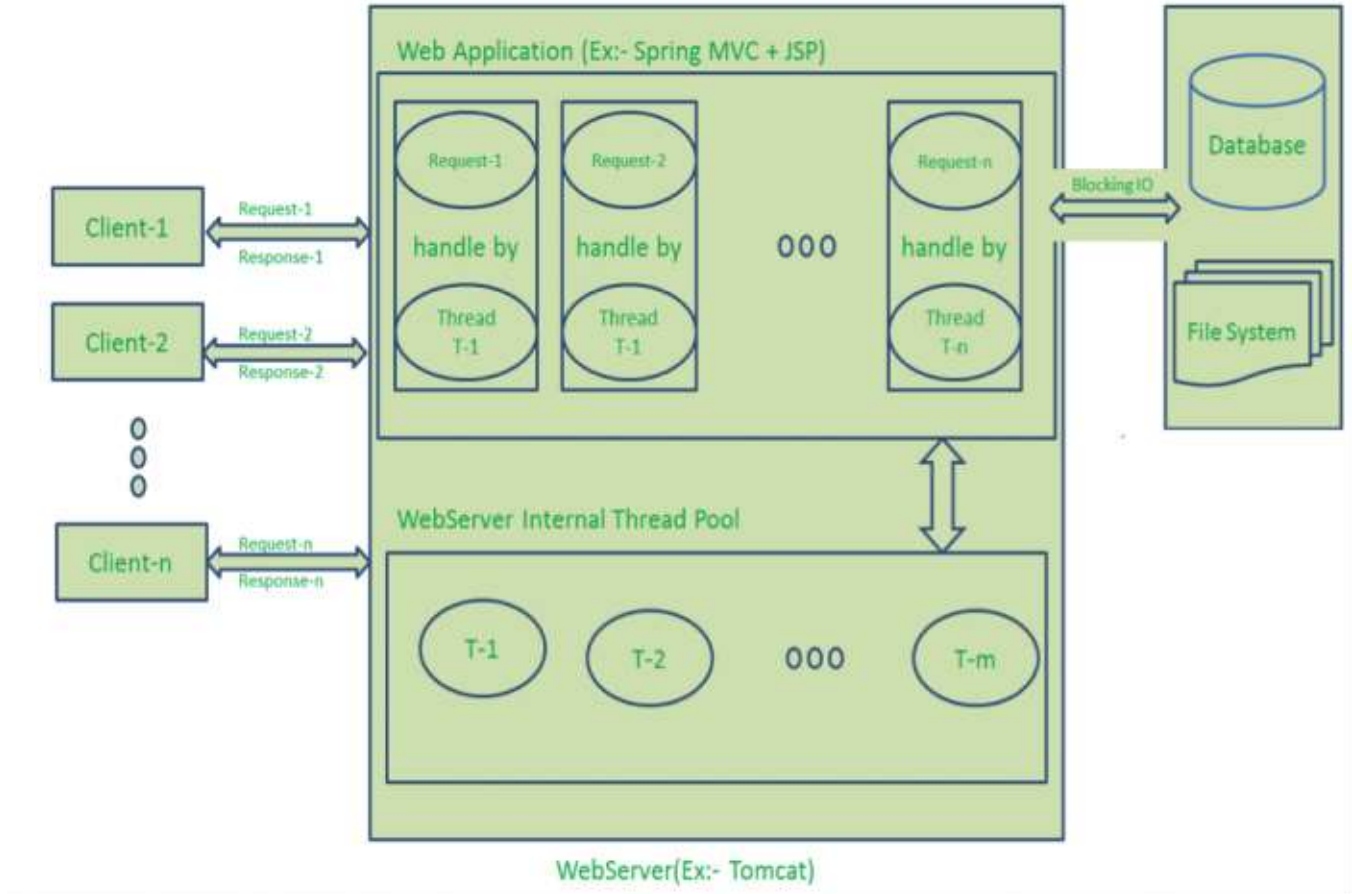
Topics

- Server Concurrency Models
- Database Concurrency

Server Concurrency Models

- Single-threaded Request-Response
- Multithreaded Request-Response
- Single-threaded Event Loop

Multithreaded Request-Response



http://www.journaldev.com/7462/node-js-architecture-single-threaded-event-loop

Single-Threaded Event Loop Model

- Components:
  - Event queue
  - Main event loop
- Main event loop:
  - While (running):
    - Dequeue event from queue
      - (sleep if queue empty)
    - Invoke callback to handle event
- Observations:
  - Only one callback in your Node app runs at a time (no race conditions!)
  - While a callback is running, nothing else can happen

Background I/O

- How does I/O occur in the background without threads?
- Solution: OS-level non-blocking I/O
  - OS system calls provide both blocking and nonblocking I/O mechanisms
  - Node leverages the nonblocking I/O mechanisms to allow the main thread to initiate an I/O request without waiting for it to complete
  - Node's main event loop checks to see if I/O has completed and adds events to the event queue

The Full Story

- Node uses the cross-platform library **libuv** to implement its event loop
- libuv uses OS-specific mechanisms to perform non-blocking I/O
- All supported OS's can do non-blocking network I/O
- Not all do non-blocking file I/O
- libuv uses a thread pool to implement non-blocking file I/O

Database Concurrency

Database Concurrency

- What happens when two requests issue database queries?
- Depends on the database driver
  - Sqlite: Each query is executed on a separate background thread
  - MySql: Depends on connection management strategy

Database Connection Management Strategies

- All requests serviced by single connection
- Each request serviced by a separate connection
- Connection pooling

Single Shared Connection

- See nodejs/mysql_demos/nonpooled.js
- A MySQL connection can be used to execute only 1 query at a time
- Concurrent attempts to use the connection are serialized
  - Performance implications?
- Issue: Stale connection
  - Database server may terminate connection if it is unused for a period of time

One Connection Per Request

- See lab3/webapp example
- For each incoming request:
  - Open a new database connection
  - Use connection to handle request
  - Close the connection
- Issues?

Connection Pooling

- See nodejs/mysql_demos/pooled.js
- Driver maintains a pool of lazily created connections
- pool.getConnection() returns an available connection from pool
  - If pool capacity is reached, additional calls to getConnection() are queued until a connection becomes available
- Client code must call release() to return connection to pool
  - Failure to call release() results in resource leak

Database Transactions

- Recall: A database transaction is a series of SQL commands that begin with START TRANSACTION and end with COMMIT:

```
START TRANSACTION;
SELECT * FROM FOO;
UPDATE FOO SET BLAH = BLAH + 1 WHERE OOMPH = 15;
COMMIT;
```

- When do you need a transaction?

When Transactions are Needed

- Handling a request requires two or more SQL commands
- If interference by concurrent execution of other SQL commands could result in incorrect results, you need a transaction
- Example:
  - E-commerce application allows users to place orders for items
  - Back order is not supported
  - Placing an order involves, among other things:
    - Verify adequate quantity available
    - Decrement quantity available