CS 555: DISTRIBUTED SYSTEMS

[THREADS]

October 10, 2017

Shrideep Pallickara
Computer Science
Colorado State University

Frequently asked questions from the previous class survey

- Shuffle less/Shuffle better
  - Which action? Those with wide dependencies
- Examples of custom partitioners?
- Threads
  - Is it expensive to repeatedly create threads?
  - Memory Management Unit

Topics covered in this lecture

- Threads
  - Contrasting with processes
  - Threads in Distributed Systems
  - An example of performance improvements with Threads
  - Threading architectures for Servers
  - State

Threads execute their own piece of code independently of other threads, but …

- No attempt is made to achieve high-degree of concurrency transparency
  - Especially, not at the cost of performance
- Only maintains information to allow a CPU to be shared among several threads
- Thread context
  - CPU Context + Thread Management info
  - List of blocked threads

Information not strictly necessary to manage multiple threads is ignored

- Protecting data against inappropriate accesses by multiple threads in a process?
  - Developers must deal with this

October 10, 2017

CS555: Distributed Systems [Fall 2017]
Dept. Of Computer Science, Colorado State University
Why prefer multiple threads over multiple processes?
- Threads are cheaper to create and manage than processes
- Resource sharing can be achieved more efficiently between threads than processes
- Threads within a process share the address space of the process
- Switching between threads is cheaper than for processes
- BUT... threads within a process are not protected from one another

Other costs for processes
- When a new process is created to perform a task there are other costs
  - In a kernel supporting virtual memory the new process will incur page faults
  - Due to data and instructions being referenced for the first time
  - Hardware caches must acquire new cache entries for that particular process

Contrasting the costs for threads [1/2]
- With threads these overheads may also occur but they are likely to be smaller
- When thread accesses code & data that was accessed recently by other threads in the process?
  - Automatically take advantage of any hardware or main memory caching

Contrasting the costs for threads [2/2]
- Switching between threads is much faster than that between processes
- This is a cost that is incurred many times throughout the lifecycle of the thread or process

A process with multiple threads of control can perform more than 1 task at a time
Implications?

- **Performance** of a multithreaded application is seldom worse than a single threaded one
  - Actually leads to performance gains
- Development requires additional effort
  - No automatic protection against each other

Thread use in non-distributed settings

- Interactive multithreaded application
  - Parts of program may be blocked or slow
  - Remainder of program may still chug along
- A single threaded process can ONLY run on 1 processor
  - Regardless of how many are available
  - Underutilization of computational resources

Another drawback of processes is the overheads for IPC (Inter Process Communications)

- Interactive multithreaded application
  - Parts of program may be blocked or slow
  - Remainder of program may still chug along
- A single threaded process can ONLY run on 1 processor
  - Regardless of how many are available
  - Underutilization of computational resources

Applications can be constructed using separate threads

- Communications dealt entirely using shared data
  - Performance is much better
- Software engineering
  - Collection of several (generally independent) tasks
  - Word Processor
    - Input handling, spell check, layout, index generation …

Threads in distributed systems: Multithreaded clients

- Hide communication latencies
  - Initiate communications
  - Immediately do something else
- Web browsers
  - As soon as main HTML page is fetched
    - Display it
    - Activate threads to retrieve other data types
Several connections can be opened simultaneously

- To the same server
  - If the server is overloaded, things get even slower
- To replicated servers
  - Data transfer in parallel
  - Much faster rendering of content

Multithreaded Servers

- Simplifies server code
- Easier to develop servers that exploit parallelism
  - E.g., handling concurrent connections
    - Each connection managed by a different thread
    - Multiple connections handled by a pool of threads

Client and Server with Threads

- Server has queue of requests received from clients
- Server also has a pool of one or more threads
  - Each thread repeatedly removes requests & processes it
- Each thread applies the same methods to process the requests
  - Each request takes 2 ms of processing plus 8 ms of I/O (when server reads from disk i.e., no caching)

Maximum server throughput with 1 thread

- The turnaround time for handling any request is 2 + 8 = 10 ms
- The server can handle 100 requests per second
- Any new requests that arrive while the thread is handling a request
  - These will be queued

AN EXAMPLE OF PERFORMANCE IMPROVEMENTS WITH THREADS
**Server throughput with 2 threads**

- We assume that the threads are independently schedulable
- One thread can be scheduled while the other is blocked for I/O
- Thread T2 can process a second request when thread T1 is blocked, and vice versa
- This increases throughput but both threads may be blocked for I/O on the single disk drive
- If all I/O requests are serialized and take 8 ms each:
  - Maximum throughput is 1000/8 = 125 requests/second

**Server throughput with disk block caching**

- Server keeps data that it reads in buffers
- When a server thread tries to retrieve data:
  - It first examines the cache and avoids disk accesses if it finds data element there
  - If the hit rate is 75%:
    - The mean I/O time per-request reduces to \((0.75 \times 0 + 0.25 \times 8) = 2\) milliseconds
- Maximum theoretical throughput:
  - Becomes 500 requests per second

**But there are costs associated with caching**

- Average processor time for a request increases
  - This is because it takes time to search for cached data for every operation
  - Let us assume that this is now 2.5 milliseconds
  - The server can now handle 1000/2.5 requests per second i.e. 400

**Let's look at caching plus multiple threads**

- Each request takes about 2.5 (processing) + 2 (I/O)
  - Total time per request is now 4.5 mSecs when disk accesses are serialized
  - Each thread can do 1000/4.5 requests per second i.e. 222 requests/second
  - With two threads:
    - 444 requests/second
  - With three threads:
    - 500 requests (bound by the I/O time)

**Worker pool architecture**

- Server creates a fixed pool of worker threads to process requests
  - Pool is initialized when server starts up
  - Incoming requests are placed into a queue
  - Workers retrieve requests (work units) from the queue and process them
Managing priorities in the worker pool?
- Introduce multiple queues
- Worker threads scan queues in the order of descending priority

Disadvantages of the worker pool model
- Number of worker threads is fixed
  - So, threads in the pool may be too few to adequately cope with the rate of requests
  - Need to account for coordinated accesses to the shared queue

Thread-per-request architecture
- Worker thread is spawned for each incoming request
  - Worker thread destroys itself after processing request
- Advantages:
  - Threads do not contend for the shared work-queue
  - Throughput is potentially maximized
- Disadvantage
  - Overhead for thread creation and destruction operations

Thread-per-connection architecture
- Associates a thread per connection
- New worker thread created when a client makes a connection
  - Destroyed when client closes the connection
- Client may make many requests over the connection

Thread-per-object architecture
- Associate a thread with each remote object
- A separate thread receives requests and queues them
  - But there is a queue per-object

Thread-per-connection & Thread-per-object
- Advantages
  - Server benefits from lower thread management overheads compared to thread-per-request
- Disadvantages
  - Clients may be delayed when a worker thread has several outstanding requests, but another thread has no work to perform
Server Design Issues

- **Iterative Servers**
  - Handles request
  - Returns response to requesting client

- **Concurrent Servers**
  - Pass request to a separate thread/process
  - Multithreaded server
  - Await new incoming request

---

The endpoint issue

- Clients send their requests to an endpoint
  - Port to which a server listens to

- But how do clients know about a port?
  - Globally assign endpoints for well-known ports
    - Internet Assigned Numbers Authority (IANA)
    - FTP (TCP, 21), HTTP (TCP, 80)

---

Implementing each service with a separate server could waste resources

- Instead of having multiple servers awaiting client requests
  - Have a single super-server

- INETD daemon on Unix
  - Listens to several ports for Internet services
    - Pop3 (110), FTP (21), Telnet (23)

- When request comes in:
  1. Fork process to handle it
  2. Process exits once done

---

Designing Servers: Support interruption

- Terminate client session
  - Server will eventually detect connection loss (TCP)

- Send out-of-band data
  - Data to be processed before any other client data

- But how can we send this out-of-band data?
  1. Send to a different port
  2. Reuse same connection
    - TCP urgent data e.g. socket.sendUrgentData(int data)

---
Tracking State in Servers

- Stateless servers
- Stateful servers

Stateless servers

- No state information about clients
  - E.g. Web Servers
- Usually some state is maintained
  - Log of documents accessed by client
  - But if this is lost, there should be no disruption of service
- Soft state: track state for a limited time
  - When timer elapses, revert to default behavior

Stateful servers

- Maintain persistent information on clients
- Use this to improve performance
  - Real and perceived
- Special measures needed to recover from failures

Stateful servers: A file server example

- Allows client to maintain local copy of file
  - Even for updates to the file
  - Maintain (client,file) tuples to track file state
  - Identify who has most recent version of file
  - If server crashes it must recover the (client,file) entries

A hybrid approach: Have the client send its state to the server

- Cookies serve this purpose for Web pages
- Tells a site about the pages accessed by a user
  - Use this to decide how to manage client
  - Sent back to browser every time state info changes
- Cookies don’t stay where they are baked!

The contents of this slide-set are based on the following references