Thread Performance Considerations

Which Model is Better?

1. When comparing two threading models, consider what metrics to apply
    - Total execution time
    - Average time to complete an order
2. Boss-Worker Model
    - 6 workers, 120ms per order
        - Total execution time = 120ms * 3 batches = 360ms
        - Average execuation time = ((5*120)+(5*240)+360) / 11 = 196ms
3. Pipeline Model
    - 20 ms per pipeline stage
        - Total execution time = 120 + (10 * 20) = 320ms
        - Average execution time = (120+140+...+320) / 11 = 220ms

Are Threads Useful?

1. Parallelization -> Speed up
2. Specialization -> Hot cache
3. Efficiency -> Lower memory requirement and cheaper synchronization
4. Threads hide latency of IO operations (single CPU)
5. But what is useful?
    - For matrix multiply -> Execution time
    - For a web service application -> Client requests/time, response time
        - Could be average, min, max, 95%
    - For hardware -> Higher utilization (CPU)
    - Evaluate the answer based on relevant metrics

Metrics for Operating Systems/Toy Shops

1. Throughput
    - How many toys per hour?
    - Process completion rate
2. Response Time
    - Average time to react to a new order
    - Average time to respond to input (mouse click)
3. Utilization
    - Percent of workbenches in use over time
    - Percentage of CPU

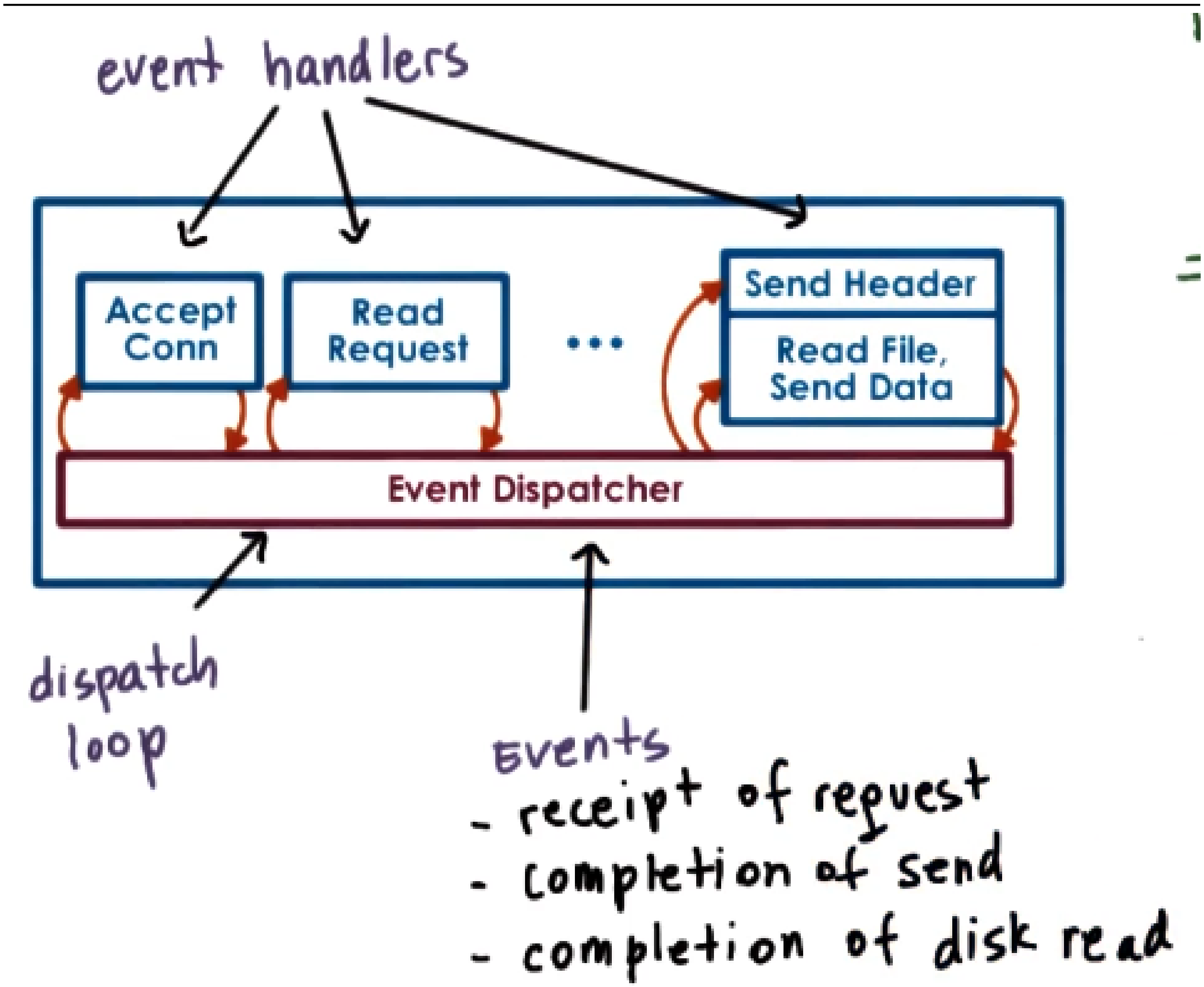
Performance Metrics

1. Metrics - A measurement standard
    - Measurable and/or quantifiable property... (Execution time)
    - of the system we're interested in... (Software implementation of a problem)
    - that can be used to evaluate system behavior (improvement vs other implementations)
2. Execution time, throughput, request rate, CPU utilization, wait time, platform efficiency, performance/dollar, performance/Watt, percentage of SLA violations, client-perceived performance, aggregate performance, average resource usage
3. Obtain metrics by experiments with real software deployment, machines, workload
    - Not always possible, so use 'toy' experiments representative of realistic settings
    - Testbed - Simulation occurring using realistic settings
4. Usefulness of threads depends on the metrics and workload we care about
    - Different number of toy orders -> different implementation of toy shop

- Different type of graph -> Different shortest path algorithm
- Different file patterns -> Different file system
- It always depends, but this is never a valid answer

Multiprocess vs Multithreaded

1. Use the example of a web server (concurrently processing requests)
2. Steps in a simple web server
   - Client/browser sends a request
   - Web server accepts the request
   - Server accepts connection
   - Server reads the request
   - Server parses the request
   - Server finds the file
   - Server computes the header
   - Server sends the header
   - Server reads the file and send the data
   - Server closes the connection
3. Multiprocessing approach
   - Pro: Simple implementation
   - Con: Allocate memory for each
   - Con: Costly context switching
   - Con: Hard/costly to maintain shared state
   - Con: Tricky to set up ports
4. Multithreaded approach
   - Pro: Shared address space
   - Pro: Shared state
   - Pro: Cheap context switch
   - Con: Implementation is more difficult
   - Con: Explicitly handle synchronization
   - Con: Requires underlying support for threads
5. Event-Driven Model
   - Single address space, process, thread of control
   - Event dispatcher waits for an event to occur and invokes a handler
   - Dispatcher == State machine
   - Calling a handler == jumping to the code
     - Runs to completion, if it needs to block, initiate blocking operation and pass control to dispatch loop



Event-Driven Model

Concurrency in the Event-Driven Model

1. MP and MT: 1 request per execution context
2. Event-driven: Many requests interleaved in an execution context
   - Single thread switches among processing of different requests
   - Dispatcher moves requests between handlers as needed
3. What is the benefit?
   - Can hide latency by context switching
   - If there's no need to context switch, cycles are spent being productive
   - Process request until wait necessary then switch to another request
   - Multiple CPUs -> Multiple event-driven processes
4. How is it implemented?
   - Sockets -> Network, Files -> Disk
   - File descriptors are used for both
   - Event == Input on file descriptor (FD)
   - Use select(), poll(), epoll() to pick a file descriptor
5. Benefits
   - Single address space, single flow of control
   - Smaller memory requirement

- No context switching
- No synchronization

6. Helper Threads and Processes
    - A blocking request/handler will block the entire process
    - Use asynchronous I/O operations
        - Process/thread makes system call
        - OS obtains all relevent info from stack, and either learns where to return results, or tells caller where to get results later
        - Process/thread can continue
        - Requires support from kernel (threads) and/or device (DMA)
        - Fits nicely with event-driven model
    - What if asynchronous calls are not available?
        - Helpers are designated for blocking I/O operations only
        - Pipe/socket based communcation with event dispatcher (select/poll)
        - Helper blocks, but main event loop (and process) will not
    - AMPED - Asymmetric Multi-Process Event-Driven Model
    - AMTED - Asymmetric Multi-Threaded Event-Driven Model
    - Pro: Resolves portability limitations of basic event-driven model
    - Pro: Smaller footprint than regular worker thread
    - Con: Applicability of certain classes of applications
    - Con: Event routing on multi-CPU systems

Flash Web Server

1. Event-driven webserver (AMPED) with asymmetric helper processes
2. Helpers used for disk reads
3. Pipes used for communication with dispatcher
4. Helper reads file in memory (via mmap)
5. Dispatcher checks (via mincore) if pages are in memory to decide 'local' handler or helper
    - Results in large savings
6. Additional optimizations
    - Application-level caching for both data and computation
    - Alignment for DMA
    - Use of DMA with scatter-gather -> vector I/O operations

Apache Web Server

1. Core -> Basic server skeleton
2. Modules -> Per functionality (security, content management, HTTP requests)
3. Flow of control is similar to event-driven model, but Apache is a combination of multiprocess and multithread
    - Each process == Boss/worker with dynamic thread pool
    - Number of processes can be dynamically adjusted

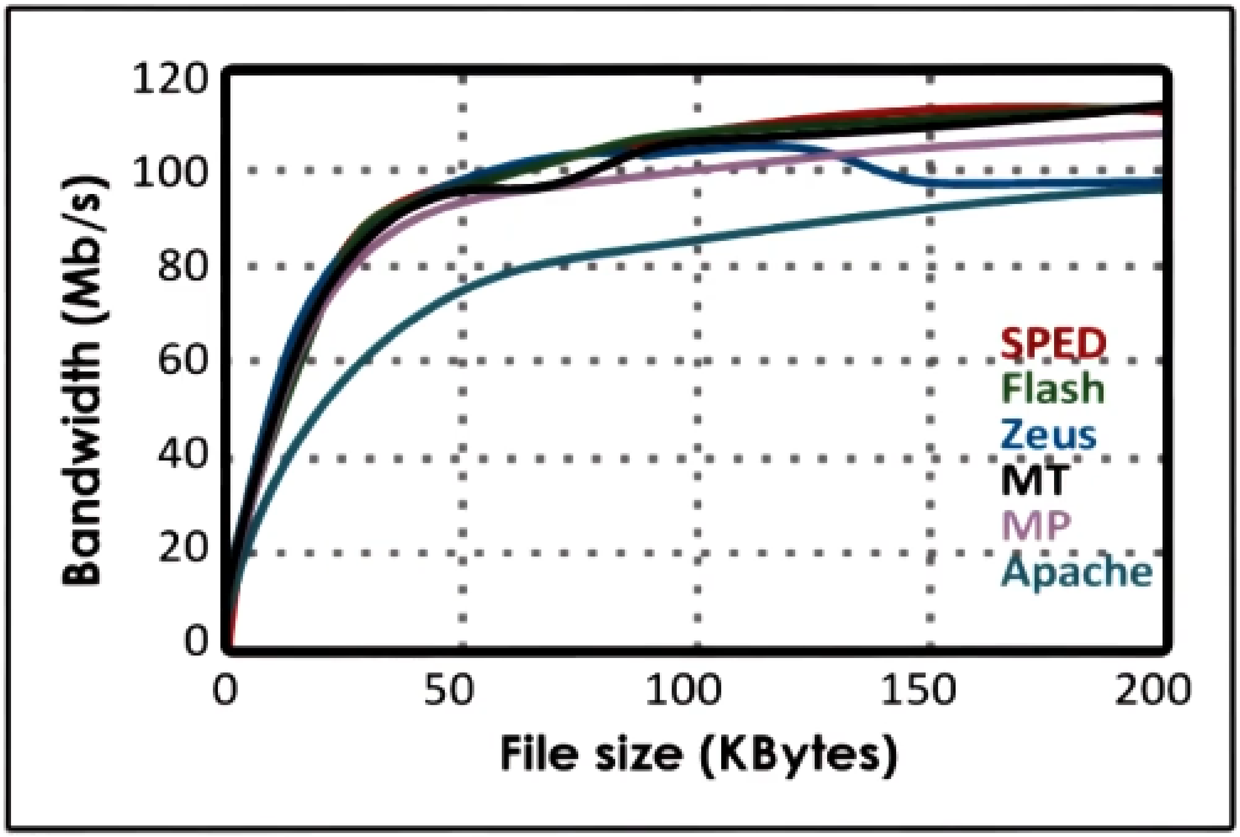
Experimental Methodology

1. What systems are you comparing? Define comparison points
    - Multiprocess (each process single thread)
    - Multithreaded (boss-worker)
    - Single process event-driven (SPED)
    - Zeus (SPED with 2 processes)
    - Apache (v1.3.1, multiprocess)
    - Compare against Flash (AMPED model)
        - Same optimizations except for Apache

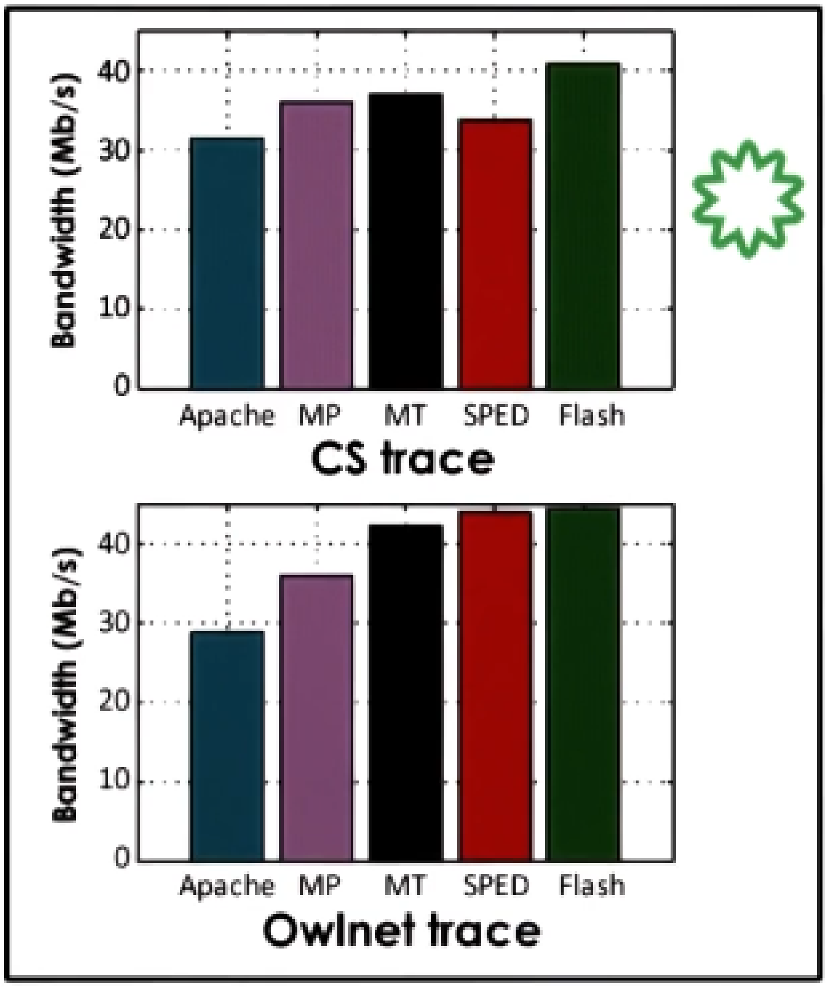
2. What workloads will be used? Define inputs
    - Realistic request workload
        - Distribution of web page accesses over time
        - Trace-based (gathered from real web servers) - Reproducible
3. How will you measure performance? Define metrics
    - Bandwidth = Total bytes transferred from files / total time
    - Connection rate = total client connections / total time
    - Evaluated as a function of file size
        - Larger file size -> ammortize per connection cost -> higher bandwidth
        - Also requires more work per connection -> lower connection rate

Experimental Results (Best Case)

1. Synthetic load: Number of requests (N) for same file (best case)
2. Measure bandwidth
    - Bandwidth = N * bytes(file) / time
    - File size: 0-200 kB - varies work per request
3. All implementations exhibit similar results
    - SPED has best performance
    - Flash AMPED has extra check for memory presence
    - Zeus has anomaly (due to DMA optimization)
    - MT/MP is slower (due to context switching)
    - Apache lacks optimizations
4. Owlnet Trace
    - Trends similar to best case
    - Small trace, mostly fits in cache
    - Sometimes blocking I/O is required (SPED blocks, Flash doesn't)
5. CS Trace
    - Larger trace, mostly requires I/O
    - SPED worst -> lack of asynchronous I/O
    - MT better than MP (memory footprint, fast synchronization)
    - Flash best (smaller memory footprint -> more memory for caching, fewer requests -> blocking IO, no synchronization needed)
6. Optimizations were important, Apache would have performed better with the same optimizations applied
7. Summary of Performance Results
    - When data is in cache:
        - SPED » Flash (unnecessary test for memory presence)
        - SPED, Flash » MT, MP (context switching overhead)
    - When disk-bound workload:
        - Flash » SPED (blocks because no asynchronous I/O)
        - Flash » MT/MP (more memory efficient, less context switching)



Best Case Experiment



Owlnet vs CS Trace

Designing Experiments

1. Relevant experiments -> Statements about a solution that others believe in and care about
2. Example: Web server experiment
    - Clients: Response time
    - Operators: Throughput
    - Possible Goals
        - Improved response time and throughput -> ideal
        - Improved response time -> acceptable
        - Improved response time, decreased throughput -> May be useful?
        - Maintain response time when request rate increases
        - Goals drive metrics and experiment design
    - Picking metrics
        - Standard metrics appeal to a broader audience
        - Consider operators, users
    - Picking configuration space
        - System resources: hardware (CPU, memory), software (# of threads)
        - Workload: Request rate, concurrent requests, file size, access pattern
        - Choose a subset of configuration parameters

    - Pick ranges for each variable factor
    - Pick relevant workload (include best/worst case scenarios)
    - Pick useful combinations of factors (some make same point)
  - Must compare apples to apples
  - Compare system to state-of-the-art or most common practice
    - Or ideal best/worst case scenario
3. After designing the experiments...
  - Run test cases n times
  - Compute metrics
  - Represent results
  - Draw conclusions