JCIP Chap 11 notes

CMSC 433, Fall 2010

Some definitions

Latency

- **The time delay between the moment a action is initiated, and the moment the first of its effects occurs.**
- Examples:
  - Time it takes a web browser to begin displaying the page.
  - Time it takes Orbitz to say it is initiating your search.

Service Time

- **Total Time required to service a request.** Defined as the elapsed time from when the request is initiated to the time when the full response has been produced.
- Examples:
  - Full time required to get the results from Orbitz.
  - On mobile devices, Google maps will first load a low-res version of the map, then load the full-res version. This improves

Capacity

- **Total workload that can be handled without violating predetermined performance criteria.**
- Example
  - Assuming all requests must be responded to within 5 ms, Orbitz has a capacity of 1000 requests/sec

Throughput

- **Number of units of work that can be handled per unit of time.**
- Example:
  - Number of Orbitz searches that are completed per second

Capacity / Throughput Relation

- Throughput is requests / sec, regardless of any other performance criteria (e.g., time per request, CPU utilization, memory consumption, etc.).
- Capacity is throughput under certain other restrictions

Bottleneck

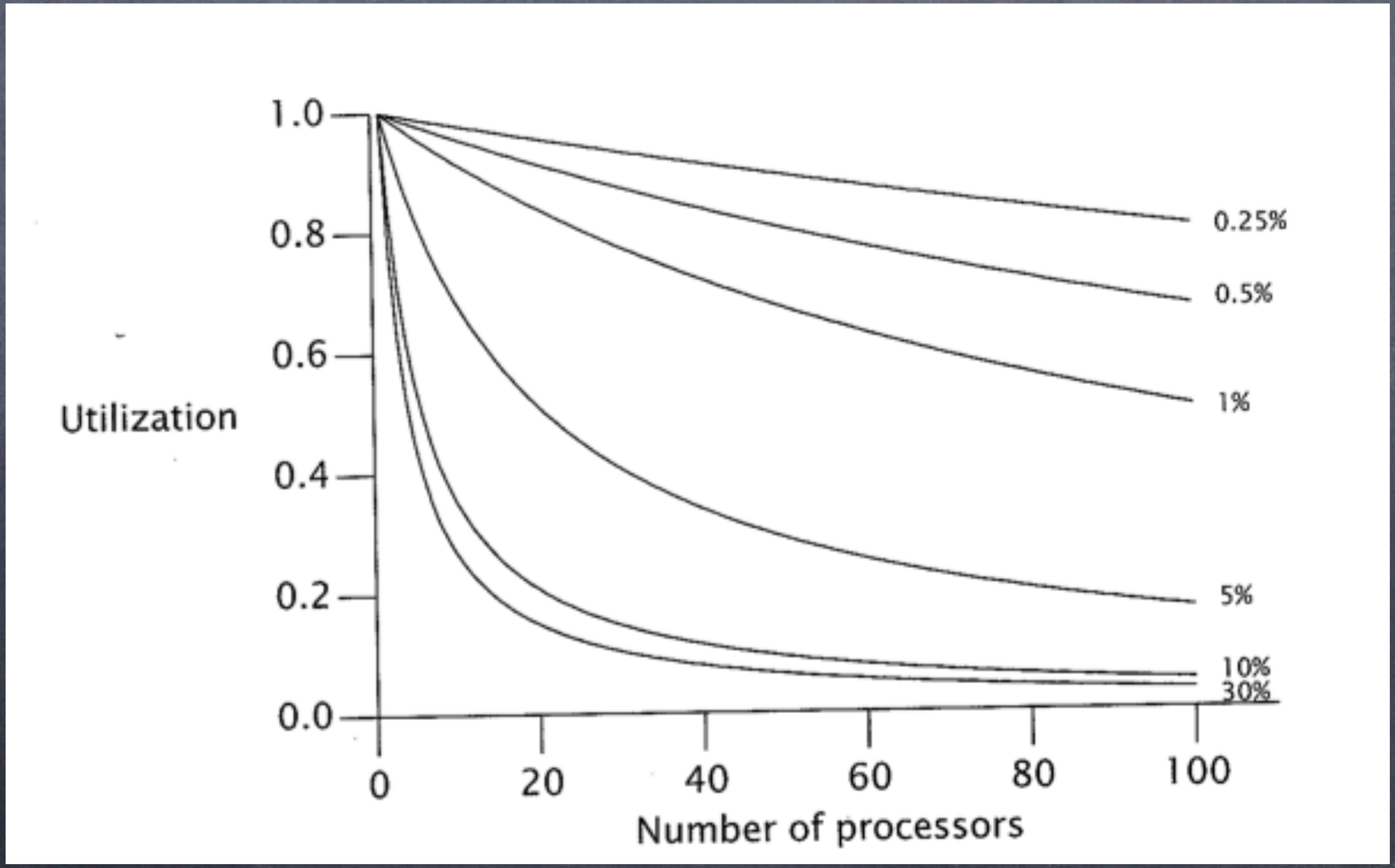
- A point in an application is a bottleneck, if adding additional resources at this point (and keeping other resources fixed) will cause increased performance (service time, throughput, or capacity).
- Example
  - If the CPU usage on the Orbitz search server is never more than 50%, then CPU

Scalability

- **The scalability of a system is its ability to improve throughput and capacity when given additional resources (CPU time, I/O bandwidth, network bandwidth).**
- If an application is perfectly scalable, throughput will double if resources are doubled (up to Amdahl's law restrictions)
  - May need to increase work to do

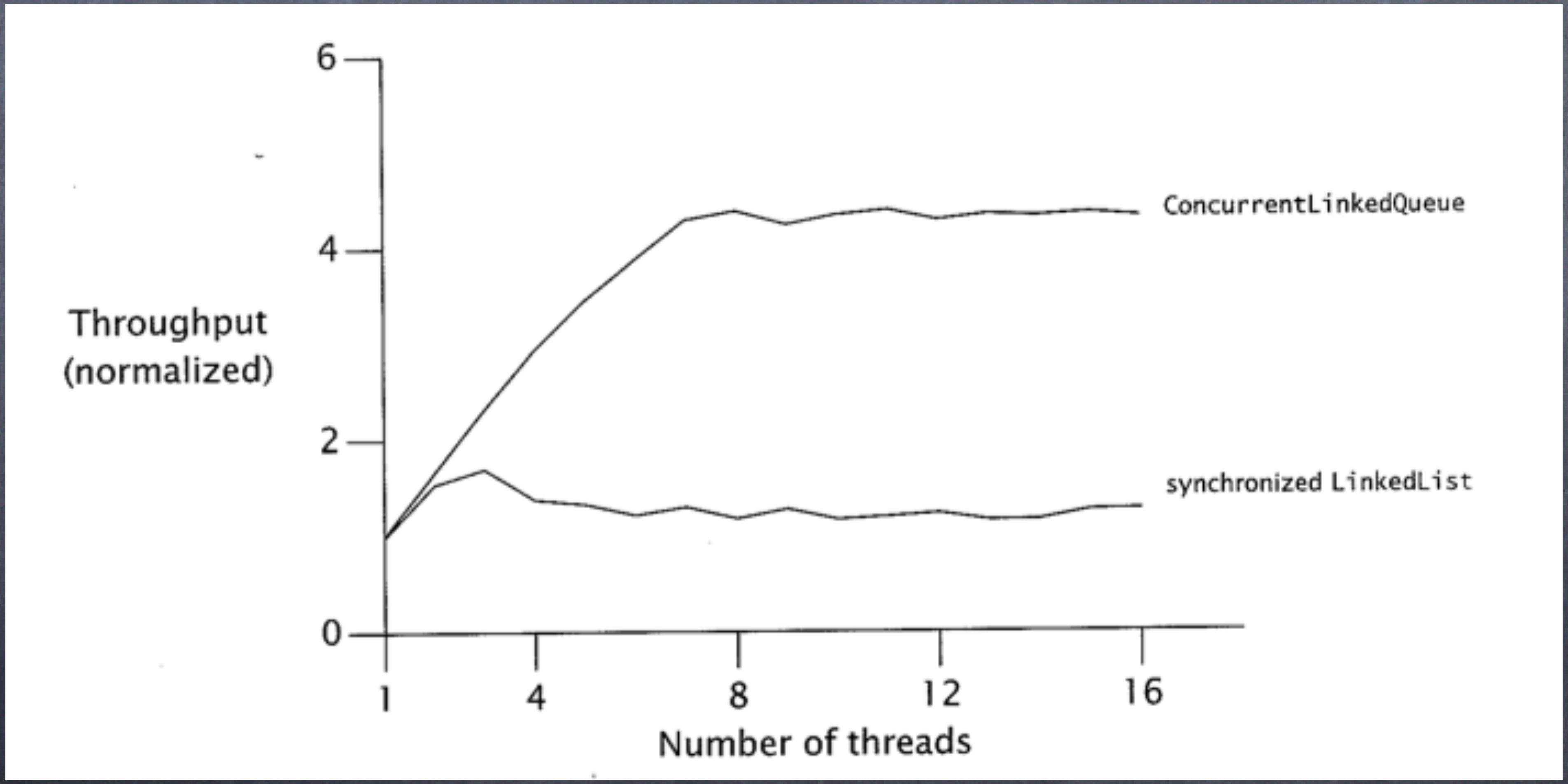
Amdahl's Law

- Speedup ≤ 1 / (F + (1-F) / N)
  - where Speedup = ST time / MT time
- Utilization = Speedup / N



Reducing Overhead

- Avoid serialization due to lock contention



Reducing Lock Contention

- Reduce duration that locks are held
  - narrower synchronized blocks
- Reduce the frequency of lock requests
  - Use lock splitting or striping
- Replace exclusive locks with coordination mechanisms that permit greater concurrency

Lock splitting

- Replace a single lock in a program with two or more locks
  - Example: separate locks for protecting events in BasicLog, vs. protecting Listenable for event notification

Lock striping

- Like lock splitting, but based on a dynamically determined policy
  - ConcurrentHashMap uses 16 locks, each for 1/16 of the hashtable
  - Key idea: prior to splitting/striping, program contends more for the lock than the data it protects

Avoiding "hot fields"

- Consider recomputing within separate threads, rather than contending for shared fields via locks
  - The extra computation cost in each thread will scale according to Amdahl's law, but serializing on a lock will not
  - E.g., avoid "object pooling," i.e., custom allocators: synchronization bottleneck