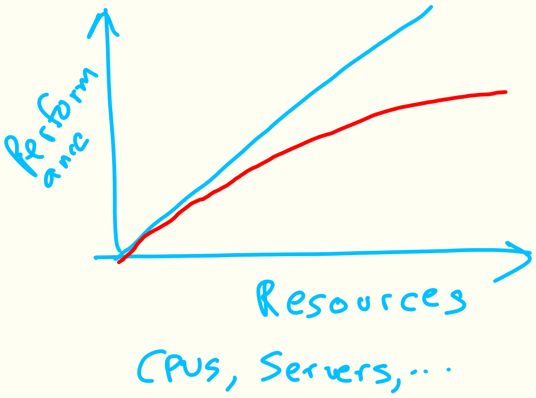
Elastic Scaling

Holy Grail: Linear Scaling

Linear Scaling
Performance increases linearly with resources

Reality
- Hard to achieve in practice
- Most scaling is sub-linear

Key Question
What is the performance as more resources are added?



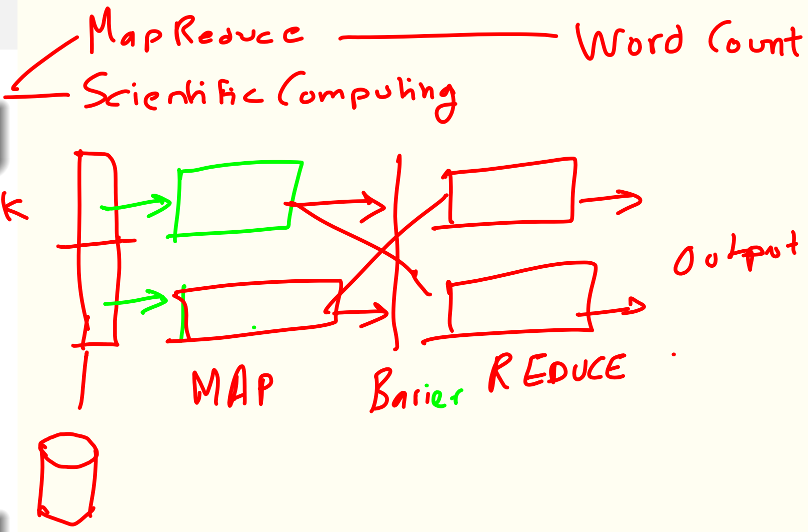
Amdahl's Law

For conventional parallel applications, what is running time on n servers?

- Ideal, linear scaling: $T(n) = \frac{T(1)}{n}$
- In practice, only a fraction of the program can be parallelized, the rest is sequential
- Let p be the parallel fraction.
- $T(n) \geq (1-p) + \frac{p}{n}$
- Parallel speedup of a program: $S(n) = \frac{T(1)}{T(n)}$

What can't be parallelized?

- Sequential file access
- Waiting for user input
- Communication synchronization



Amdahl's Law

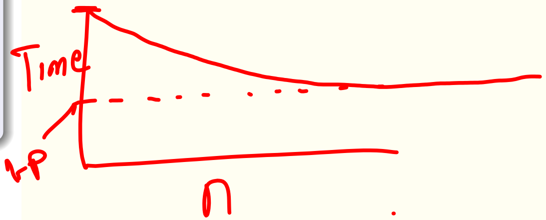
Insights

- Useful for "what-if" scenarios about performance
- Diminishing returns
- Cost = number of servers X running time
- Cost = $n * T(n) = n * [(1 - p) + (p/n)]$
- Amdahl's law gives minimum running time at "infinite" scaling

Pay-per-hour X # servers = n

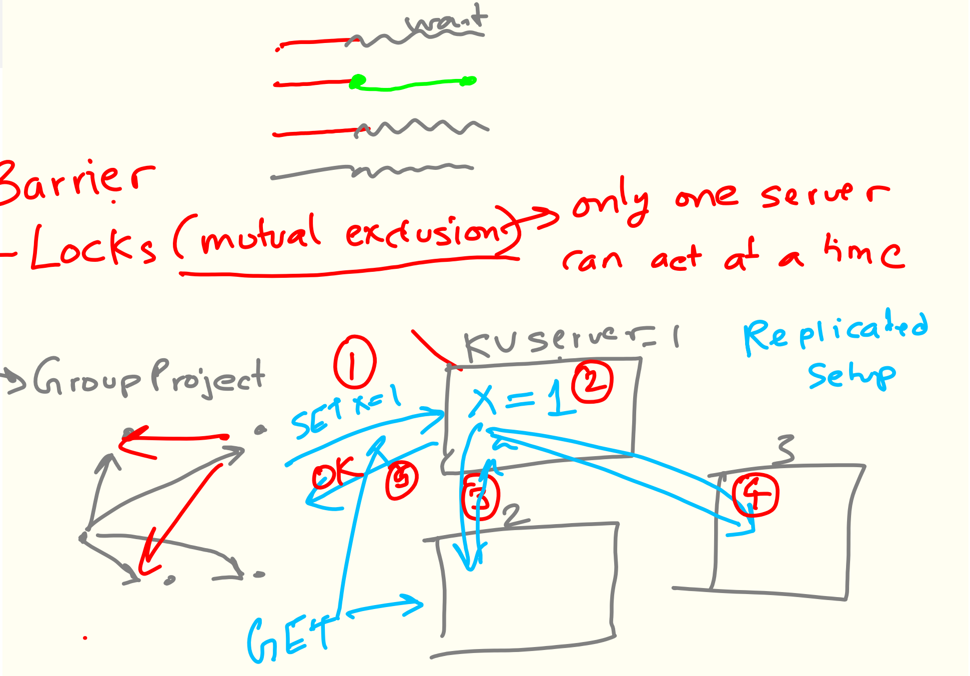
$T(n) = (1-p) + \frac{p}{n}$ ← # CPUs, # servers,

$n \to \infty$ $T(n) = (1-p)$



More On Scaling

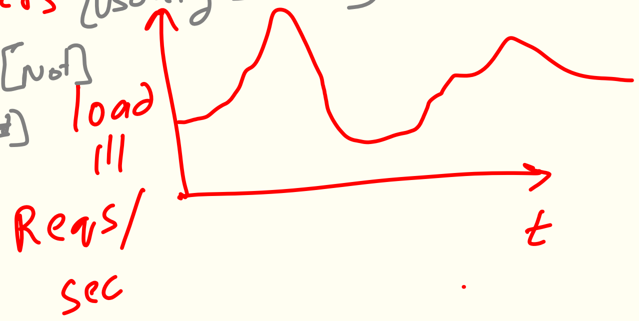
- In perfect scaling, throughput $X(n) = \lambda n$
- Contention for resources causes a slowdown by $\sigma(n-1)$
- $X(n) = \dfrac{\lambda n}{1+\sigma(n-1)}$
- Amdahl's law: Serialization is main form of contention
- Consistency or coherence penalty grows with *square* of number of nodes
- Broadcast-based strict consistency example: each SET request involves n^2 communication
- Coherence penalty also common in human systems (adding more programmers to a project makes it slower, etc.)
- "Universal scalability law": $X(n) = \dfrac{\lambda n}{1+\sigma(n)+\kappa n(n-1)}$



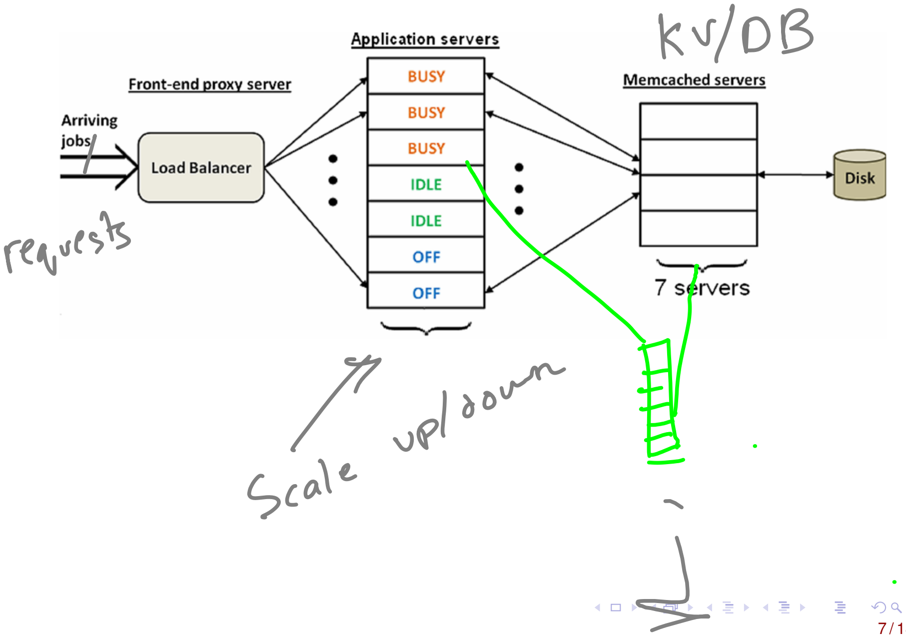
Horizontal Scaling

- Add more servers
- Often for *stateless* services that do not have consistency problems
- Enabled by cloud's utility computing model
- Servers are behind a "load balancer" that routes client requests.

- Latency-sensitive, interactive app
- KV [NOT stateless]
- Web servers [usually stateless]
- Databases [Not]
- Chat/IM. [usually not]



Application Architecture



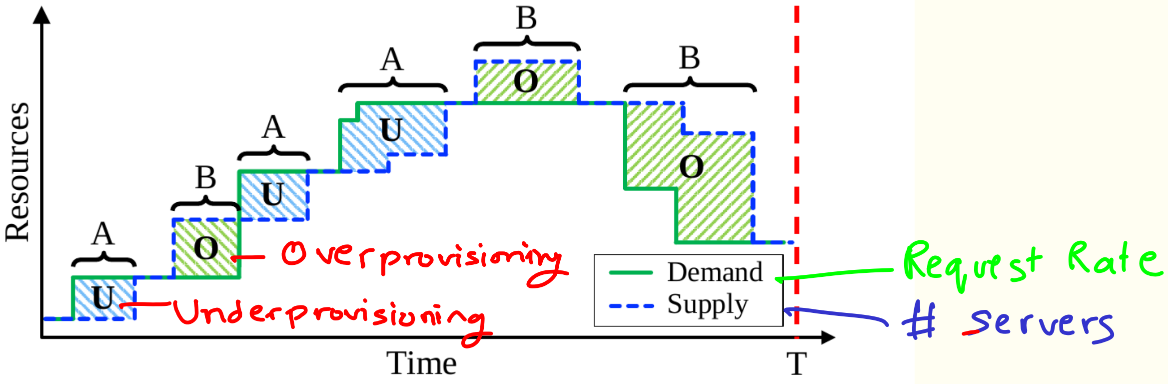
Scope of Scaling

- Vertical scaling: Make machines bigger → Sometimes possible in virtualiz cloud servers
- Single tier/ multi-tier
- Infrastructure: VMs or containers
- Purpose:
  - Performance
  - Cost
  - Energy
  - Availability of Service — handle workload spikes
- Centralized/decentralized

Horiz

Elastic scaling

- Servers change with workload
- Especially relevant in cloud
- Cost is function of resources used



When To Scale

Key:Match available resources to the workload

- Under-provisioning: Load on individual servers is high
- Leads to SLA violations for applications

- Over-provisioning: Excess amount of servers
- Servers cost money, so need to be careful with overprovisioning.

Degraded performance

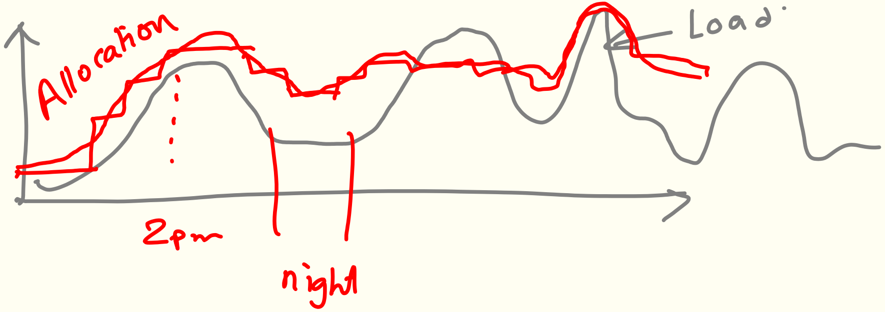
Service Level Agreement

— Avg Response time < 100 ms Target

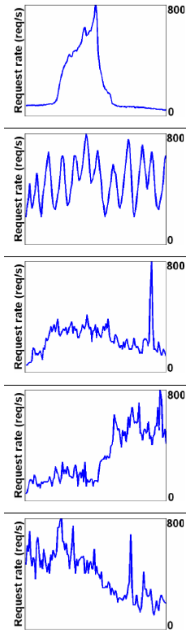
Else, pays $1000/second

Scaling Indicators/Triggers

- CPU utilization [if CPU on all servers >90%, then add.]
- Workload timeseries.
- Application SLA violations
- Scheduled (more during day, etc.)

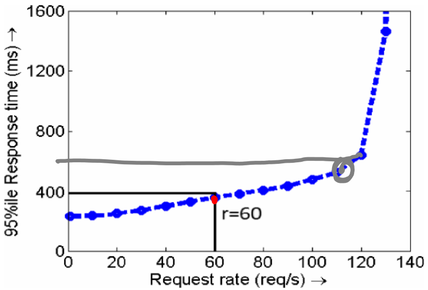


Diversity In Workload Patterns



How much to scale

- Add/remove servers until desired outcome is reached
- Want to "right size" the cluster to handle current workload
- Capacity planning: Can use queueing theory models
- M/M/1 system gives us response time distribution for single server
- M/M/c system for c servers

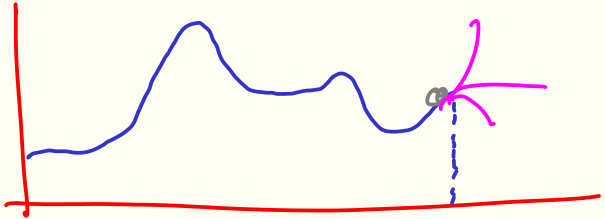


95%ile response time vs. request rate

Performance SLA

95% ile Response Time

Online: Only past workload is known



Elastic Scaling Approaches

Reactive Scaling

- Looks at current values of scaling metrics to determine scaling action.
- Challenge: Scaling operations are not instantaneous and take time (up to few minutes).

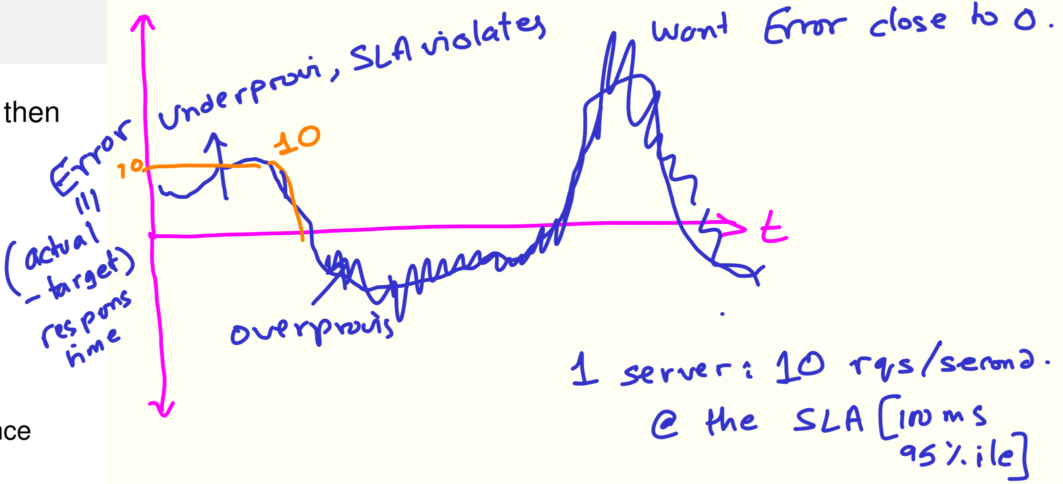
Proactive Scaling

- Predict future workload and scale accordingly

Reactive Scaling

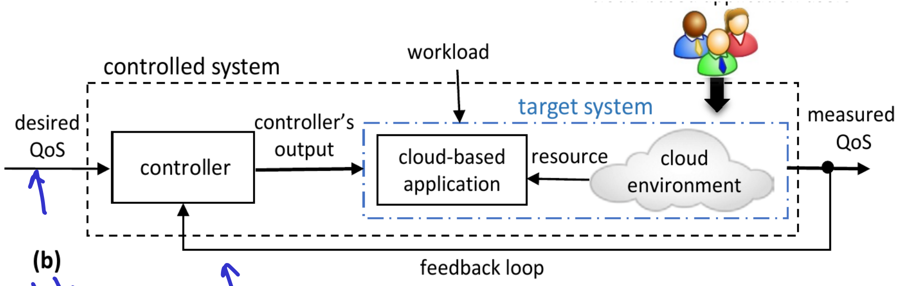
Threshold based policies: if metric above/below some threshold, then scale.

- Key challenges: How to determine threshold?
  - In most cases, heuristics work OK.
  - If CPU > 90%, add server
- Difference between metric and threshold can also be used to determine how much to scale.
  - $\Delta n = c \times (\text{metric} - \text{threshold})$
  - c is determined based on server capacity and workload
  - "Model-based" scaling, because this needs a server performance model
- Usually, metric is smoothed (exponential moving average), to prevent transient spikes from affecting scaling decisions.
- If CPU spikes to 100% for 1 second, and comes back down to 5%, we don't want to launch an armada of servers.



More Reactive Scaling

- Dynamic thresholds
- Machine learning approaches such as reinforcement learning
- Control theory. Use feedback loops



Quality of Service ≈ SLA

measured − desired = error

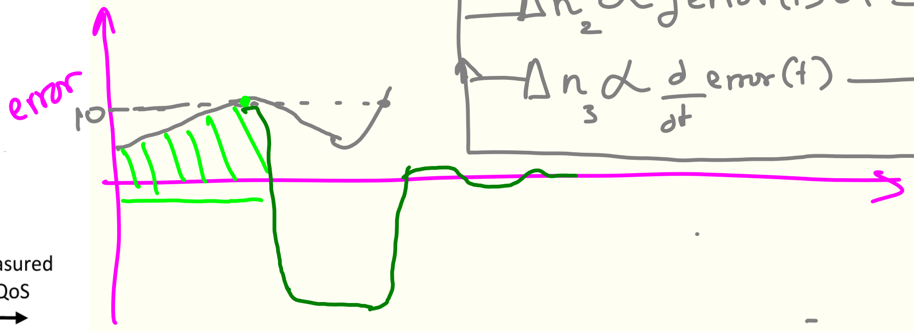
PID Controllers

$\Delta n_1 \propto \text{error}$

$\Delta n_2 \propto \int \text{error}(t)\,dt$

$\Delta n_3 \propto \frac{d}{dt}\text{error}(t)$

$+ \Delta n$

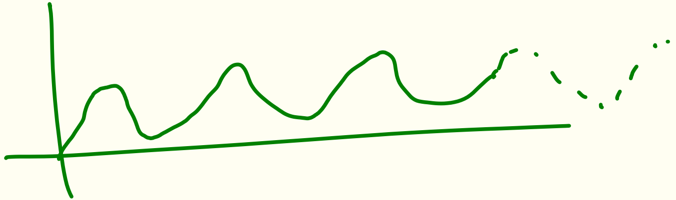


Servers → Performance may not be linear

Proactive Scaling

Key: Predict future workload to scale ahead of time

- Workload time series analysis to predict workload in some future time interval (say, 5 minutes).
- Common time-series techniques: moving averages, auto-regression, ARIMA, etc.
- Can build complex machine learning models for time series predictions (RNNs)



Practical Considerations

Key problem: Instability

1. Metric crosses threshold
2. Add more servers
3. Load on servers decreases *below* scale-down threshold
4. Scale down
5. Goto step 1

Solution: Hystersis and Inertia

- Don't scale down if reduced load is due to recent scale-up action
- Same principle used in thermostats etc.