Towards Redundancy Aware Network Stack (RANS)

Tufts UNIVERSITY

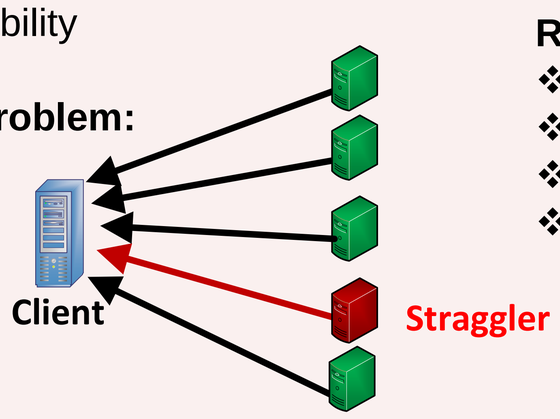
Ali Musa Iftikhar(Tufts), Fahad R. Dogar(Tufts), Owais Ahmad(LUMS), Ihsan Qazi (LUMS)

Motivation

High Performance needs of today's Datacenters:

- Predictable latency
- Fluid response times
- High availability

Straggler problem:



Reasons?

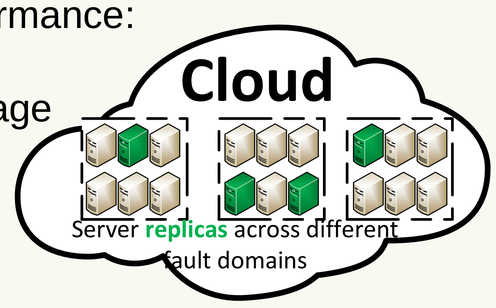
- Background tasks
- High load
- Failures
- etc

Keeping tail latency low is challenging

Replication to the Rescue!

Replication techniques to improve performance:

- Cluster file systems
- Amazon S3, Windows Azure Storage
- Facebook's Haystackk



Current approaches

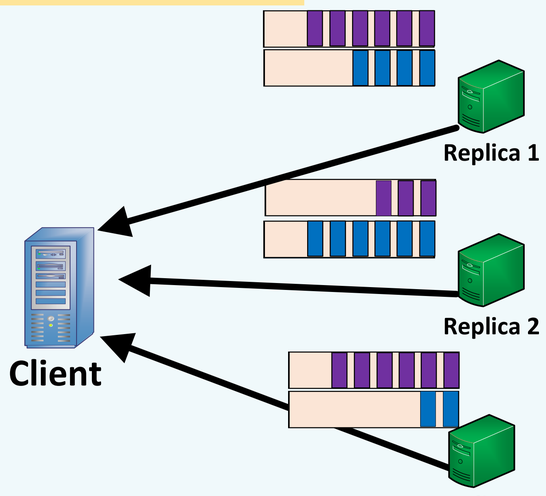
- Choose the best replica (**Difficult to predict** stragglers in advance)
- Adaptive replica selection (**Reactive, slow**)
- Initiate redundant requests – use first one that completes (beneficial only under low loads, **overloads the system** at higher loads)

Our Approach

GOAL: Making duplicate requests first class citizens of the network stack, so their overhead is made zero (or negligible)

Redundancy aware scheduling framework

- **Multiple Queues**: To isolate and classify requests as original and duplicate.
- **Strict Priority:** To prioritize the original requests over the duplicate ones
- **Purging stale requests:** To remove all the remaining requests as soon as any corresponding one completes.



Potential benefits of our approach

- Duplicate requests never hurt the original requests
- Reduced latency under unpredictable scenarios
- Information about stragglers not required
- Purging ensures maximum gains

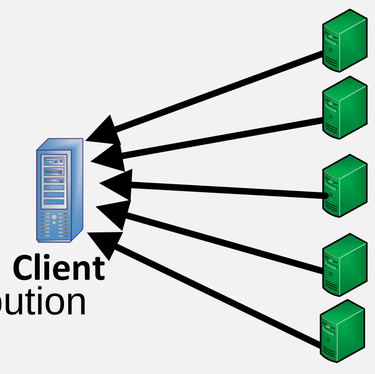
Challenges

- Current network stack does not have the ability to purge stale requests,
  purging will ensure maximum redundancy gains.
- This framework needs to be implemented to all resources,
  e.g Network, Storage, Applications.
- Each resource has its own set of challenges.
  - e.g. purging is difficult for the network

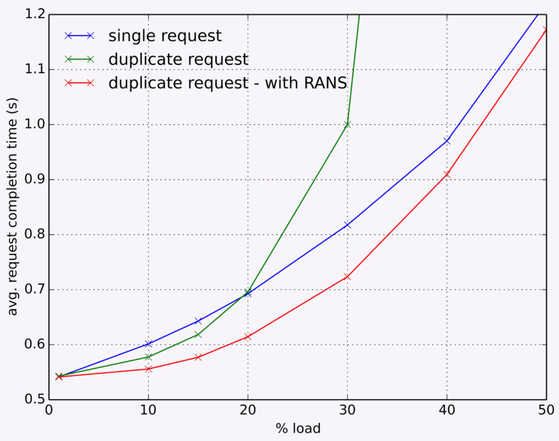
Experimental Setup

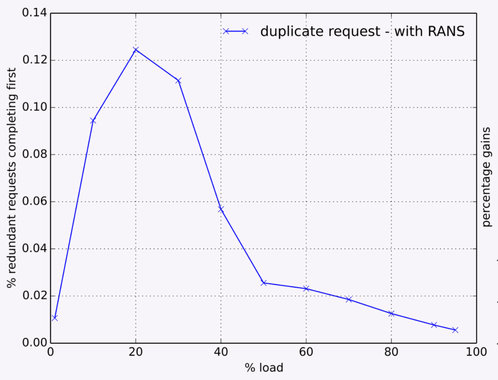
Experiments conducted on NS2

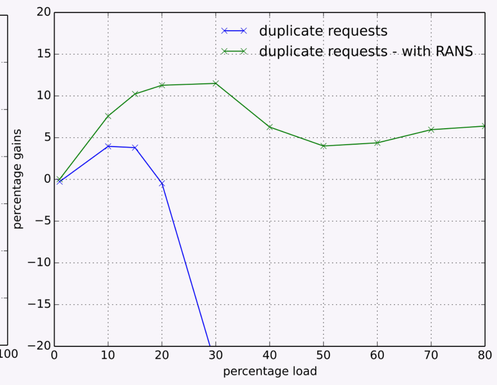
- Number of servers = 10
- File chunks size = 64MB
- Requests arrive with a Poisson distribution
- Simulation duration = 1000s
- Duplicate servers are chosen uniformly at random



Initial results







- Gains from the imbalance of server loads imbalance.
- More the sources of stragglers more the gains.

We expect to see higher gains with purging

Flow completion times improves for low loads, and higher loads.

Conclusion and Future Work

- Our results motivate that redundant requests should be made the rule rather than the exception
- Develop network functions that support purging - with minimal changes to the switching hardware
- Make applications RANS aware
- Evaluation on realistic workloads and testbeds