NetApp™

Go further, faster

# A Congestion-Aware Network File System


**Alexandros Batsakis** ***NetApp****

Randal Burns *Johns Hopkins University*

Arkady Kanevsky *NetApp*

James Lentini *NetApp*

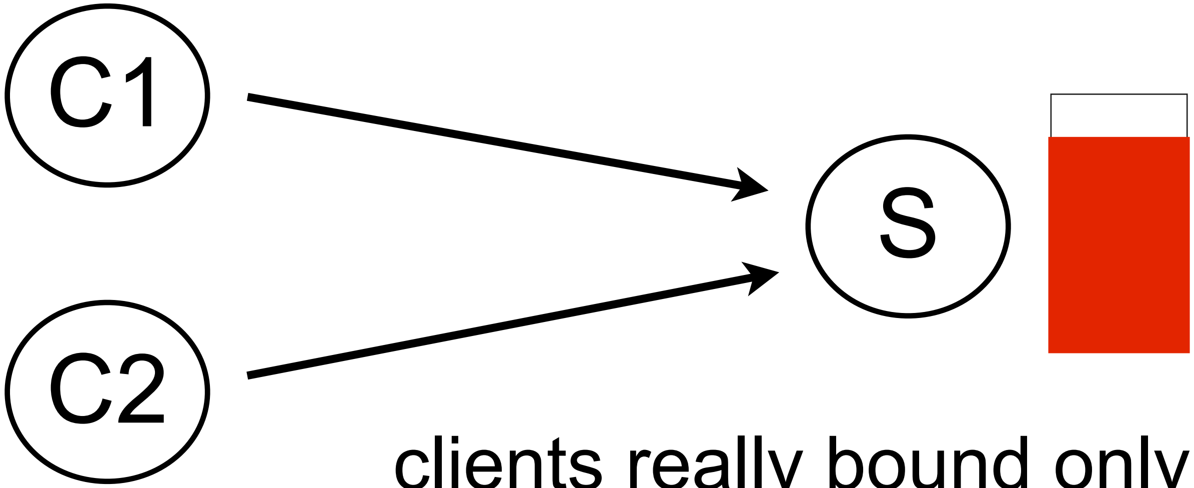
Thomas Talpey *NetApp*

# What Is This Talk About ?

- a framework for scheduling client operations in a distributed file system based on server's congestion

# What Is Wrong Now ? Selfishness

- clients try to maximize their own throughput
  - send requests to the server greedily
  - each request incurs a cost to the system
    - network, memory, disk
  - do not care about social impact (externalities)




clients really bound only by flow rate (their benefit)

# Clients Have (Good) Excuses

- server takes all responsibility (system-design)
- clients are
  - oblivious to server load
  - oblivious to other client population
- our objective is to teach clients to behave better
  - to care about the social impact of their actions
  - to become congestion-aware !

- implementation:
  CA-NFS: Congestion-aware Network File System

# CA-NFS Building Blocks

- monitor system usage and quantify congestion
- schedule client operations

# Assessing System Load / Congestion

NetApp

- how can one measure congestion ?
  - throughput, latency, time, cpu%, ... ???
  - black box, grey box, ... ???

- how can one compare load across
  - heterogeneous workloads ?
  - heterogeneous devices ?
    - 80% CPU usage vs 100 pending disk I/Os ?
  - heterogeneous machines ?

# Congestion Pricing

- unify congestion under a single metric

based on *B. Awerbuch, Y. Azar, and S. Plotkin, "Throughput-competitive online routing" , FOCS '93*

– congestion price = exp function of the resource utilization
– we adapt it to fit storage systems [auction model - proof in the paper]

$$P_i(u_i) = P_{max} \frac{\{k_i^{u_i} - 1\}}{\{k_i - 1\}}$$

– $u_i$ utilization of resource *i*
– *Pi, Pmax* price of resource *i*, max price
– $k_i$ degradation factor as the load-increases
  - device-specific e.g. disk vs network

# Resource Monitoring

- the theory makes no assumptions about the devices that are monitored
  - an expression of the utilization
    - real devices:
      - network, CPU, memory, disk
    - virtual devices (heuristics):
      - read-ahead effectiveness
      - cache effectiveness *{Batsakis et al "Awol" at FAST '08}*
    - can be extended to any device
      - SSDs, Infiniband, …

# Operation Scheduling

- NFS servers operate under false assumptions
  - client benefit increases with server throughput
  - all client operations are equally important

- client operations come at different priorities
  - explicitly: low-priority processes
    - out-of-protocol handling (QoS etc.)
    - see... future work
  - implicitly: synchronous vs asynchronous ops

# Client Operations & Implicit Priorities

- synchronous versus asynchronous ops
  - synchronous operations:
    - reads, metadata
    - must be performed on-demand (applications block)
  - asynchronous operations:
    - write, read-ahead
    - can be time-shifted depending on the client state
      - memory usage, application needs, ...
- our goal is to schedule client ops so that non-time critical (async) I/O traffic does not interfere with on-demand (sync) requests

# CA-NFS Operation – Reverse Auction

- clients and servers encode their resource constraints by increasing or decreasing their prices
- servers advertise their congestion prices to clients
- clients compare the server prices with their local prices and they decide to:
  – issue read-aheads prudently or aggressively
  – defer or accelerate a write

# Write Acceleration

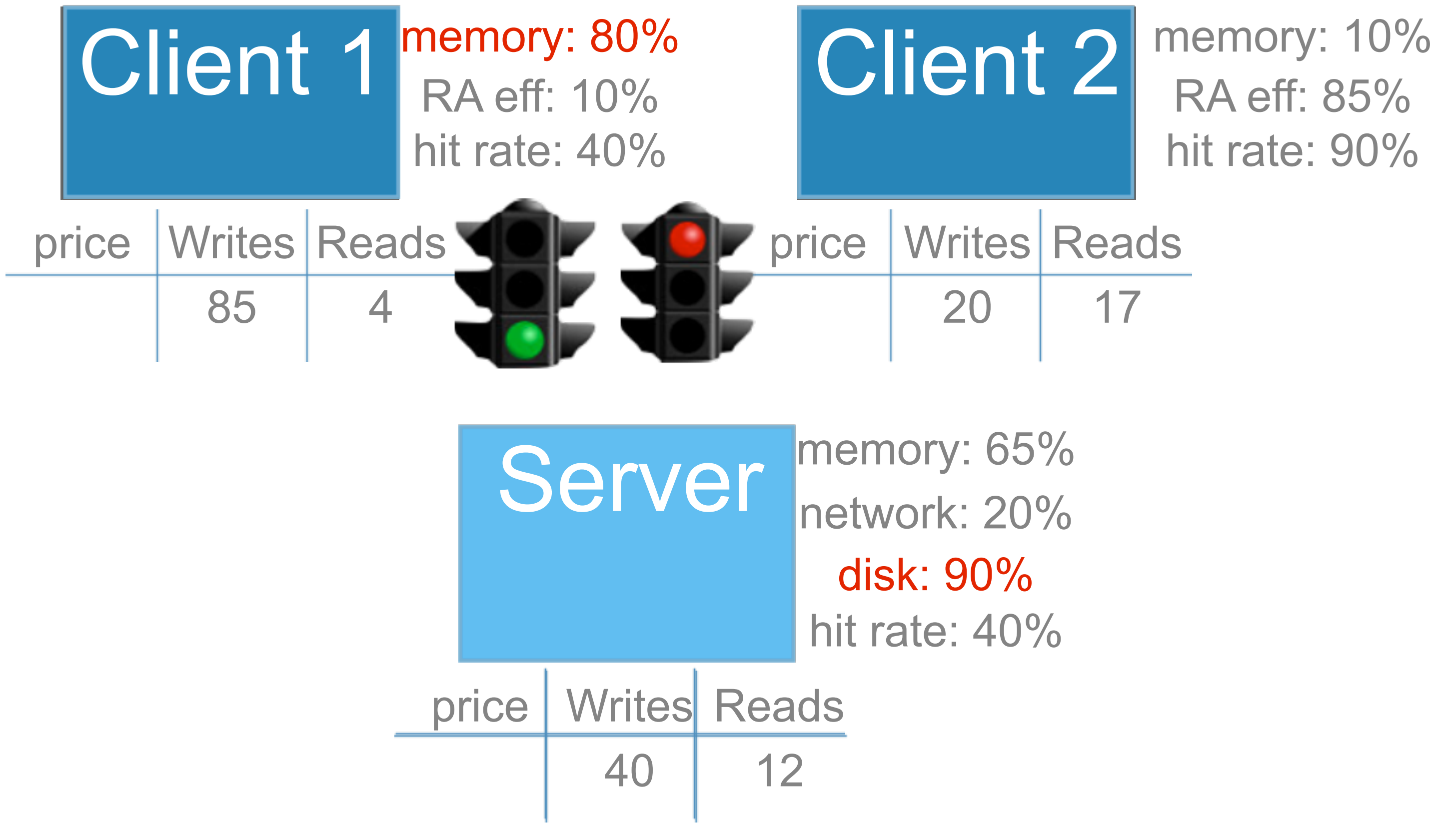
NetApp™

- CA-NFS clients notify the server to sync the data immediately upon a WRITE
  - no client buffering is needed
  - preserves the cache contents of the client (maintain hit rate)
- – if the server load is low, why sync later ?
- – saves client memory :)
  - no double buffering -- maintains client cache
- – consumes server resources immediately :-(

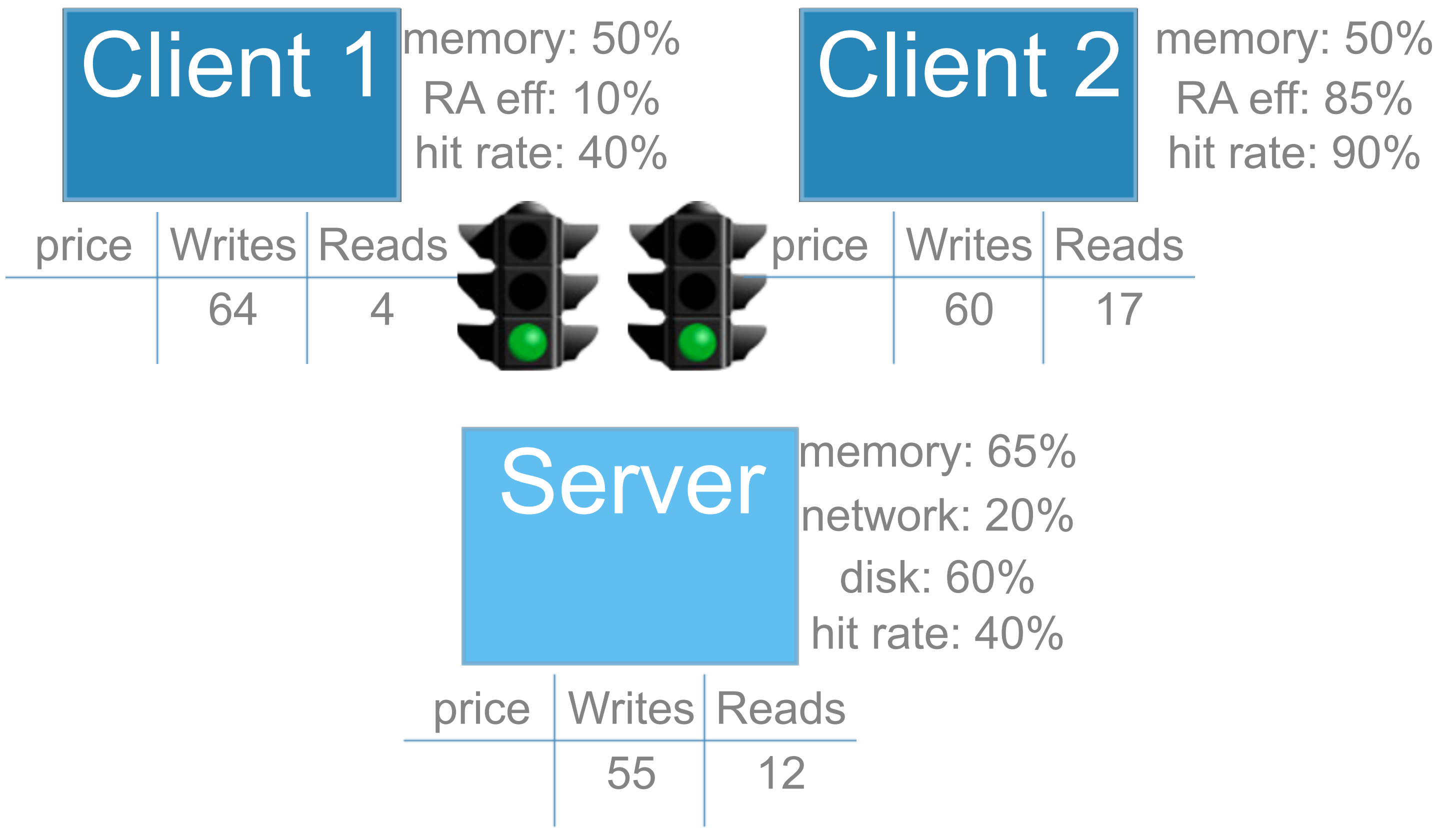
# Write Deferral

- CA-NFS clients keep data in local memory only and do not copy them to the server
  - if the server load is high postpone the write
  - saves server memory, disk and network I/O :-)
  - consumes clients memory :-(
  - faces the risk of higher latency for subsequent COMMIT operations upon close
    - but they would be slow anyways (high load)
    - some heuristics to throttle small write deferral

# CA-NFS in Practice



# CA-NFS in Practice

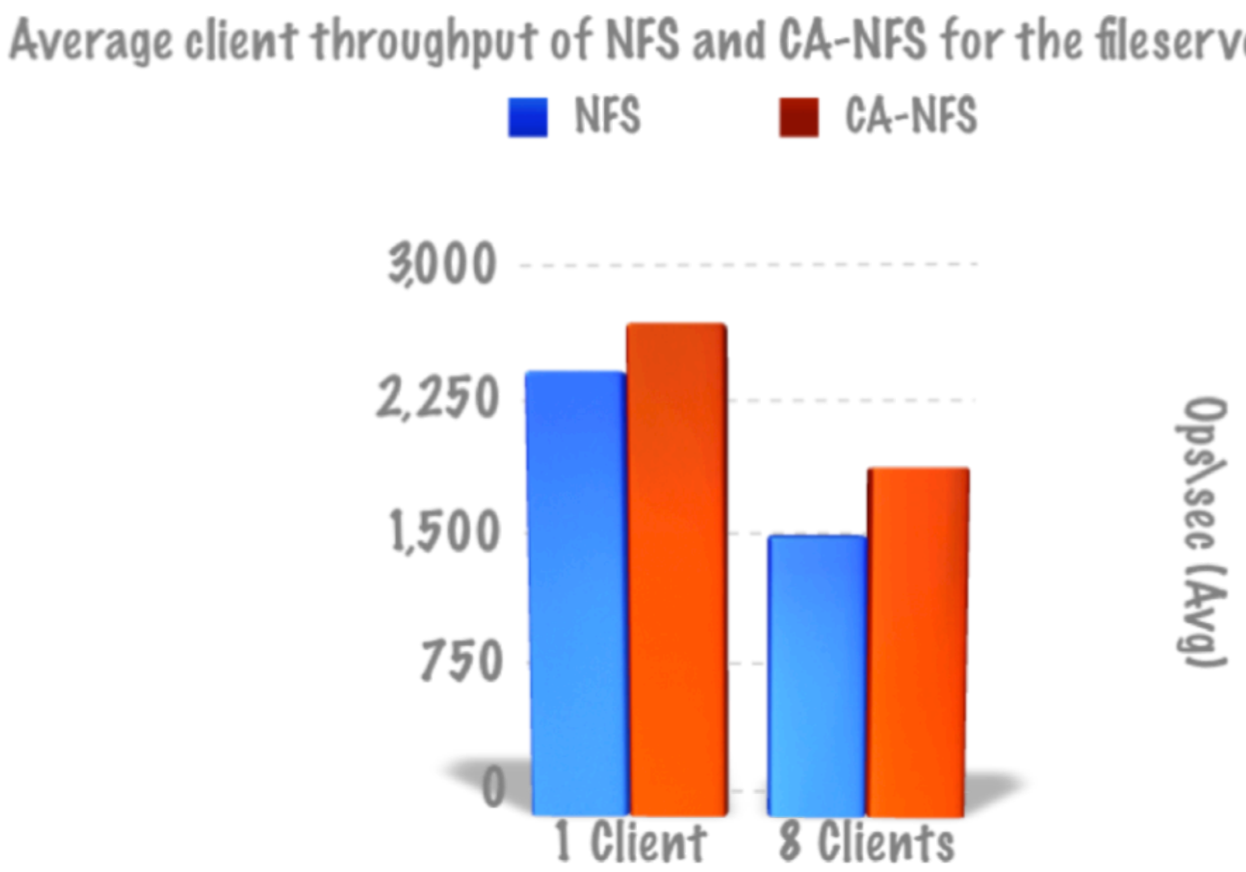



CA-NFS "exchanges" resource congestion among clients and the server

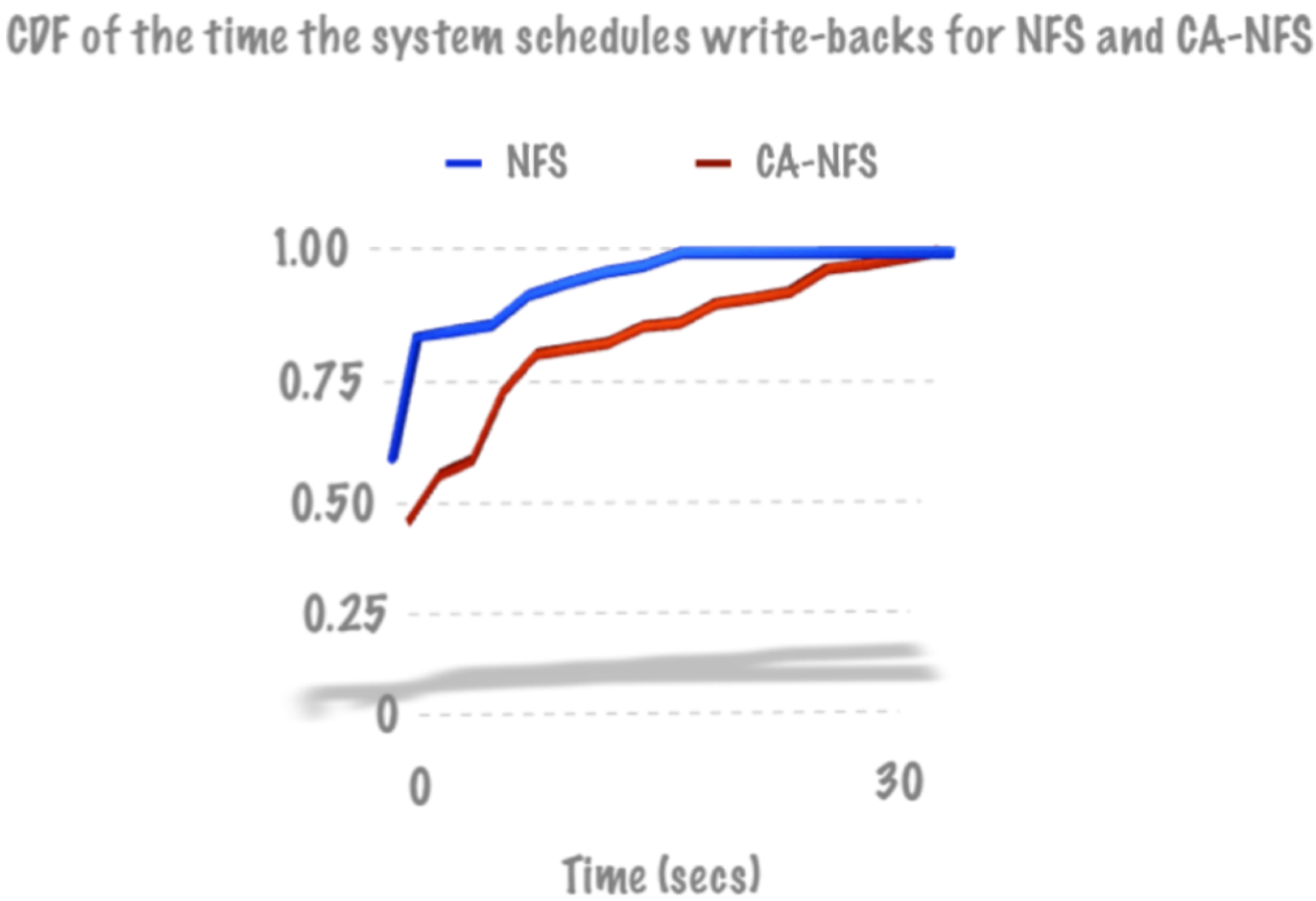
# Experimental Analysis

- two different workloads (filebench)
  - fileserver: 1000s of real NFS traces
    - creates, deletes, reads, writes, etc.
    - many asynchronous operations
  - oltp: based on a database I/O model
    - many small random reads and writes
    - mostly synchronous operations

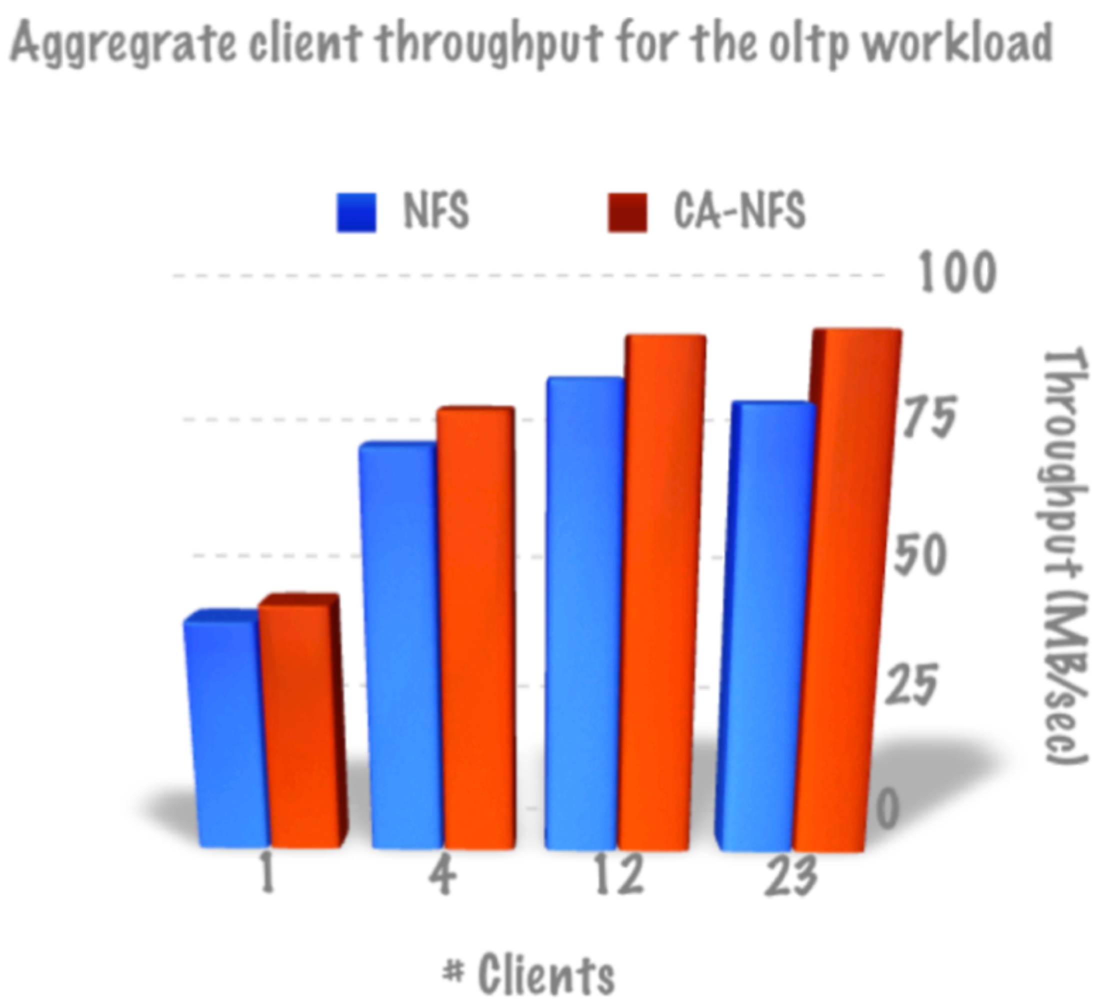
# Fileserver – Results I



# Fileserver – Results II



# OLTP - Results



# To Do (or Not To Do)

- smart scheduling of async operations is just a “proof-of-concept”
  - policies & priorities for synchronous operations
    - e.g. if price > 0.8 then stop application X
  - fairness over time
    - one client may drive prices up for everybody
  - resource reservations by differentiating prices
  - proportional sharing based on salaries
  - holistic flow control

# Parting Thoughts

- contribution: a framework to build performance management based on congestion
- case study of an "economic" anomaly
  - client benefit does not always increase with throughput
    - client requests come at different priorities
  - server cost always increases with load
  - benefit-based vs cost-based system design

Thank You

Questions?