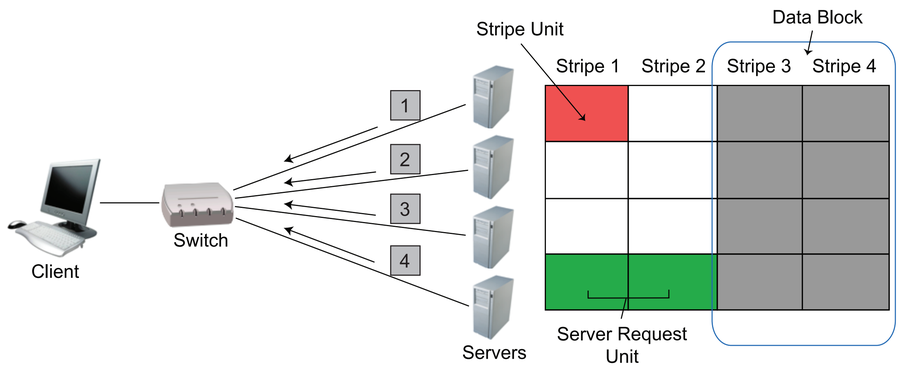
Solving TCP Incast in Cluster Storage Systems

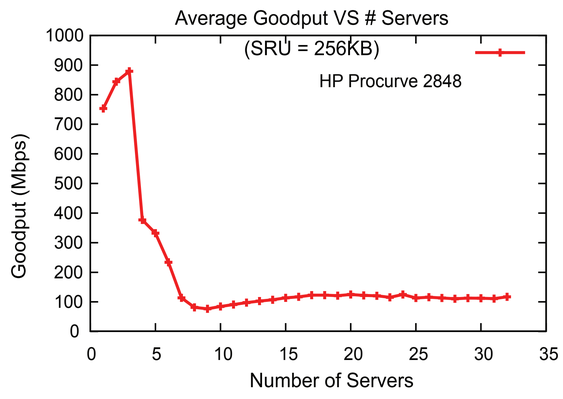
Vijay Vasudevan, Amar Phanishayee, Hiral Shah, Elie Krevat, David Andersen, Greg Ganger, Garth Gibson

TCP Incast Problem [Phanishayee et. al FAST2008]

- Cluster-based storage using TCP/IP over ethernet
- Data striped across many servers
- Client & servers separated by one or more switches
- Client requests a "block" of data and waits
  - A block is composed of one or more stripes
  - Each storage device serves its own stripe units
  - When all block data is received, client begins next request

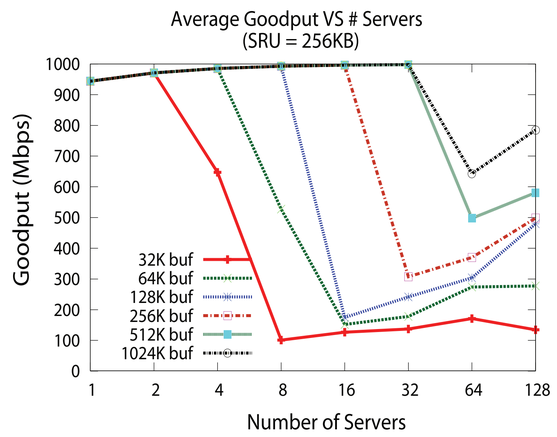


- Client can experience very poor TCP throughput
- Data from servers overflows switch buffers
- Buffer overflow causes significant packet loss
- Goodput as low as 1-10% of client link capacity!



Preventing Incast: Large Switch Buffers

- Large switch buffers delay onset of Incast
- Practical solution in use in the field today
- Switches with large buffers are expensive



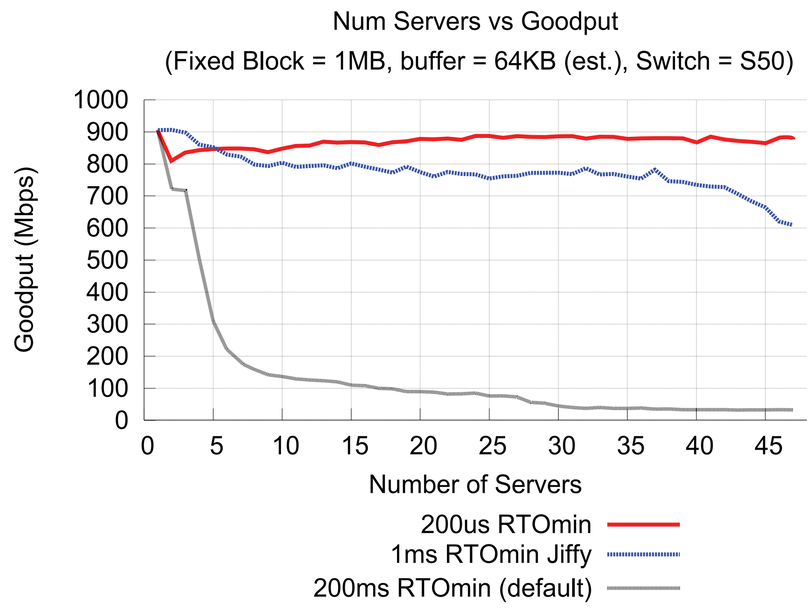
Conclusions

- TCP Incast throughput collapse can be avoided by using microsecond-granularity TCP timeouts
- Solution presented is practical, effective and safe
- CMU Tech report: CMU-PDL-09-101 February 2009
- Acknowledgement: Brian Mueller and friends at Panasas

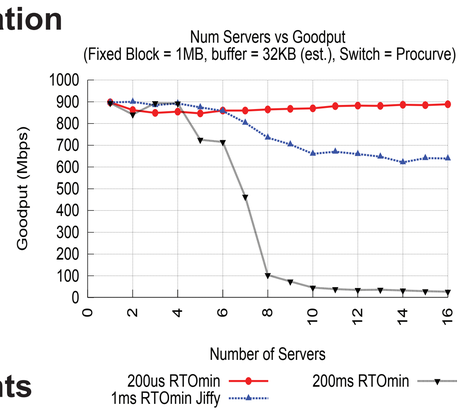
Preventing Incast: Fine grained TCP timeout

- Reduce minimum retransmision timeout 200ms → 200µs

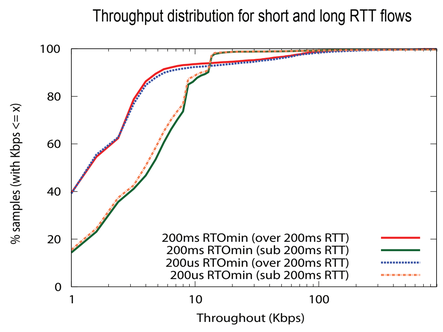
- **Is it Effective?**
  - 48-node cluster using Force10 S50 Switch
  - Each of N servers respond with 1 MB / N bytes
  - Achieves maximum throughput for up to 47 concurrent servers



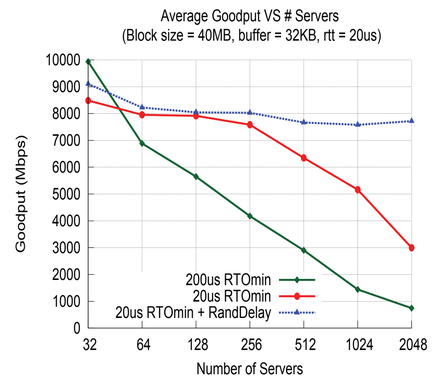
- **Is it Practical?**
  - Current Linux Implementation
    - Uses 'Jiffies' with 1ms granularity
    - Has 5ms lower bound on timeout
  - Our Implementation
    - Uses high resolution timers
    - Tracks RTT in µs
    - Redifines TCP constants



- **Is it Safe?**
  - Deployed two servers uploading identical files
  - No effect on performance of bulk-data TCP flows



- **Next-generation Datacenters**
  - Scaling to thousands of servers in simulation with 10Gbps ethernet
  - Improved throughput by adding random delay to desynchronize retransmissions
  - RTO = est. RTO * (1 + RAND(0.5))



Carnegie Mellon

pdsi

Parallel Data Laboratory

SVD09