



Lustre NRS Simulation

Yingjin Qian, Wang Di
Lustre Group

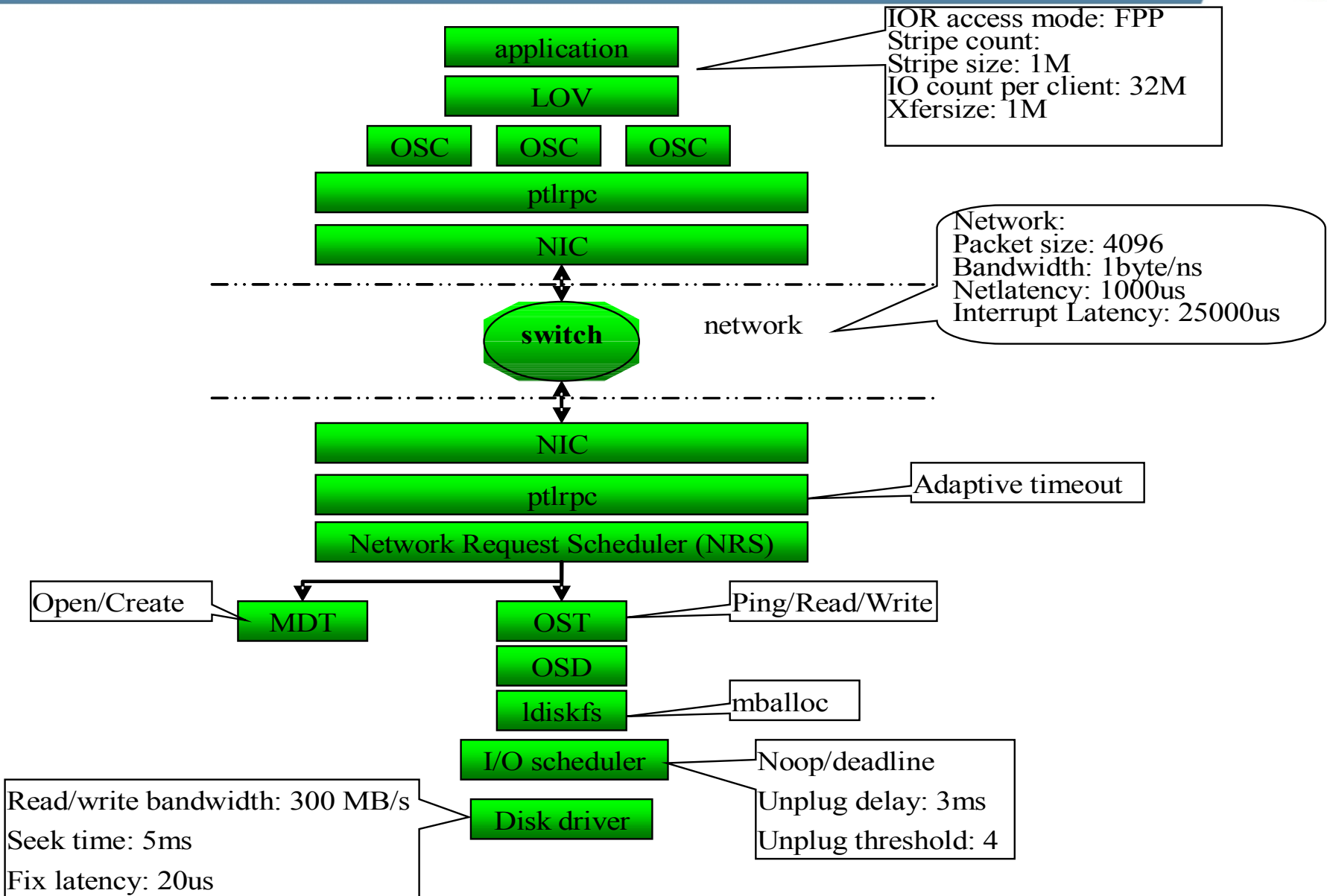


Agenda

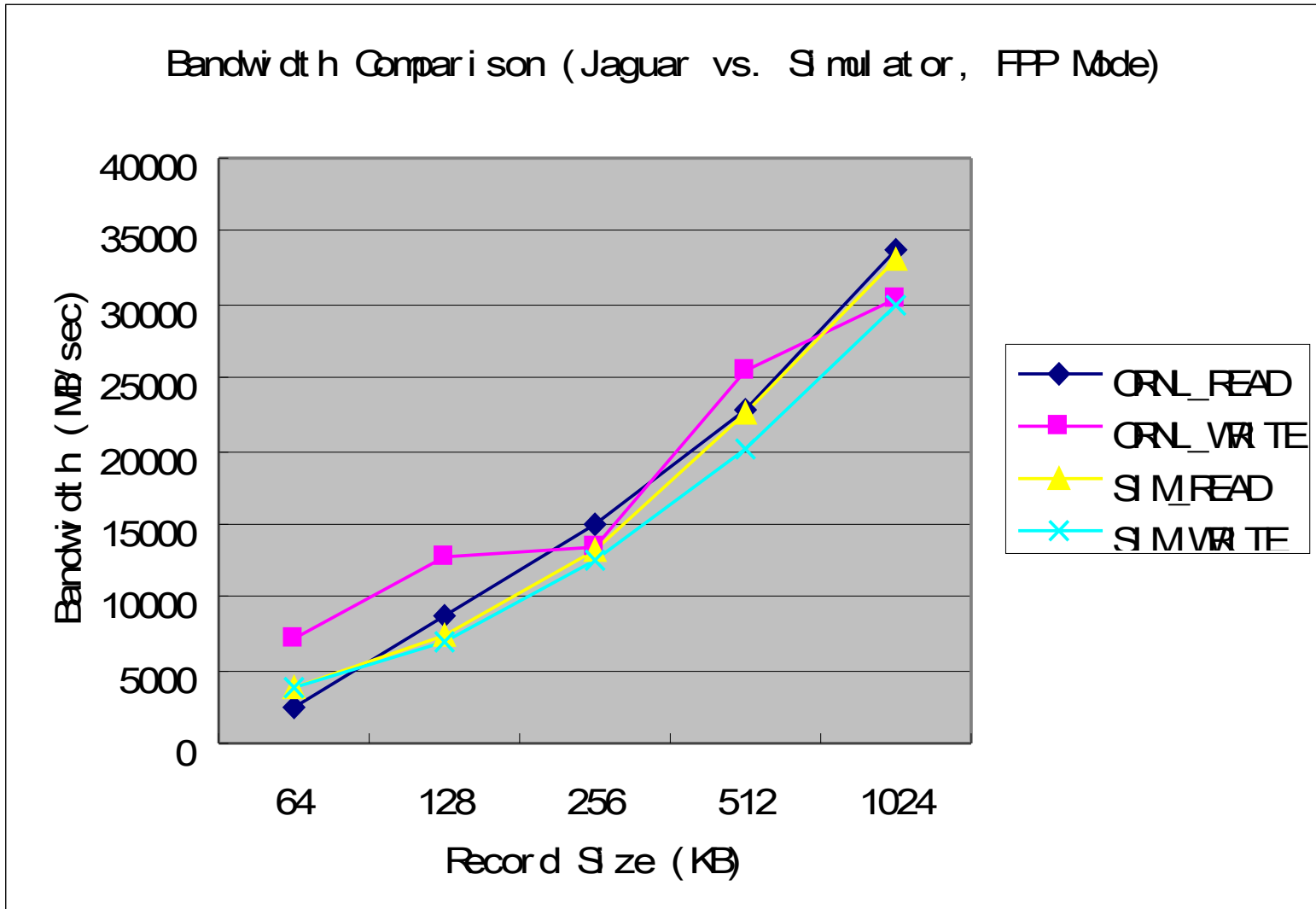
- Lustre NRS simulator
- Comparison and analysis
- NRS algorithms

Lustre NRS Simulator

- Main purpose
 - > Study Lustre I/O behaviors in large scale.
 - > Design various algorithms at very large scale on the simulator



Comparison and analysis (Jaguar)



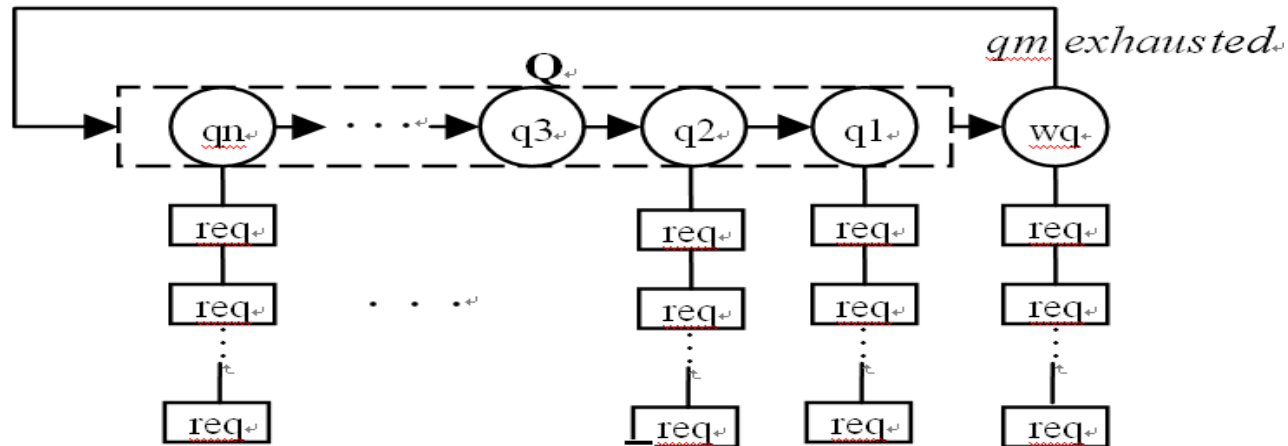
Network Request Scheduler (NRS)

- Current requests are dispatched in FCFS order
 - > As fair as the network;
 - > Over-reliance the disk elevator;
 - > Limited by disk queue depth;
- NRS manages the incoming requests and provides improved and consistent performance by reordering request execution to present a workload to the backend storage system.
 - > Work set == queued RPCs not # service threads.
 - > Work with mballocc block allocator

Block allocation

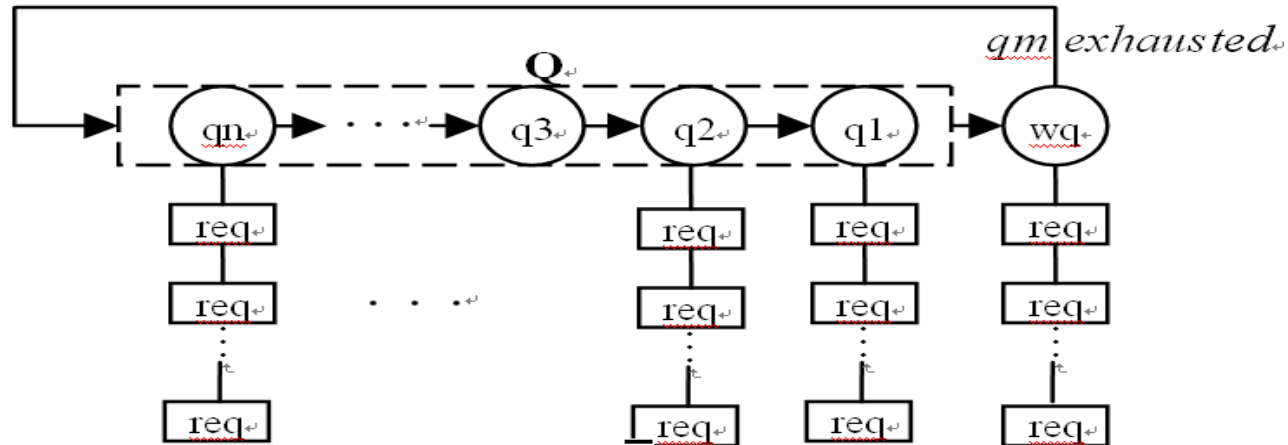
- Block allocation is the heart of a file system. The goal is to reduce disk seek time by reducing the file system fragmentation.
- Use pre-allocation technology, the pre-allocation size is very large when file size is larger than 1M.
- The principle of our NRS is to put the I/O requests belong to the same data object as close as possible.

NRS (FRR algorithm)



- File Object based Round Robin (FRR) NRS algorithm
 - > Each object has two separated I/O queues called sub queues to manage the incoming read/write requests respectively, and order them according to the offset in object.
 - > Under FRR service, sub queue per object is serviced in a round robin fashion and in each round each queue is provided with a fixed **quantum** of I/O service.

NRS (FRR algorithm)



- File Object based Round Robin (FRR) NRS algorithm
 - > Each storage server has a global FIFO queue (Q) to manage sub queues with outstanding I/O requests.
 - > At the beginning of each service round remove first sub queue from (Q) as the (WQ), dequeue I/O requests from it for execution until (WQ) is empty or its associated quantum is exhausted.

NRS (FRR algorithm with deadline)

- FRR does not ensure the fairness in object: I/O requests to an object may keep on arriving, scheduling policy ignores for a very long time a request, because it prefers to handle other requests that close to last serviced one.
- Deadline is Introduced: use a deadline list included same requests sorted according to their deadline.
 - > First check deadline.
 - If the deadline of first requests is elapsed, choose the request to service;
 - Otherwise, choose the request from current work sub queue.

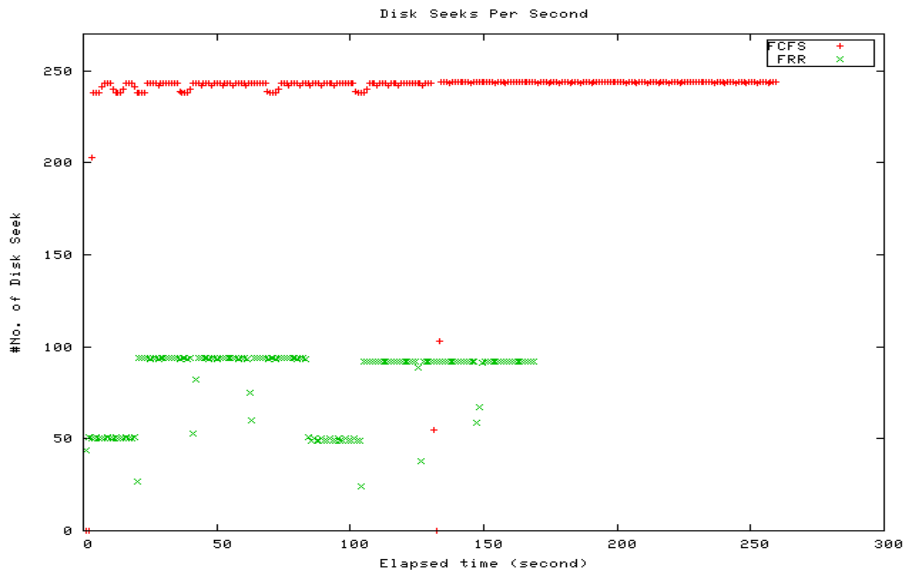
Dynamic and mandatory deadline

- For normal I/O requests, the deadlines are set dynamically
 - > Amount of pending I/Os
 - > The number of queued RPCs
- Urgent I/Os, i.e. I/O resulting from lock callback for cleaning the cache data on client; space grant updates.
 - > Client indicates the max service time on server (**S**).
 - > $T(\text{deadline}) = T(\text{arrival}) + \mathbf{S}$.
- *Comparison with two level priority*
 - > I/O requests in High level queue may starve the ones in lower level queue.
 - > With deadline guarantee, it eliminates the starvation.

NRS (FCFS vs. FRR)

- Performance comparison on single OST

OST	Read	Write	Avg Disk seeks per second
FCFS	246.36 MB/s	240.36 MB/s	240
FRR	373.49 MB/s	381.49 MB/s	70



Simulated parameters:

- Client count: 1000
- Raw disk bandwidth: 500 MB/s
- Total Disk seek time: 2ms (consider as raid0 device with 5 SATA disks)
- Stripe size: 1M
- Access mode FPP

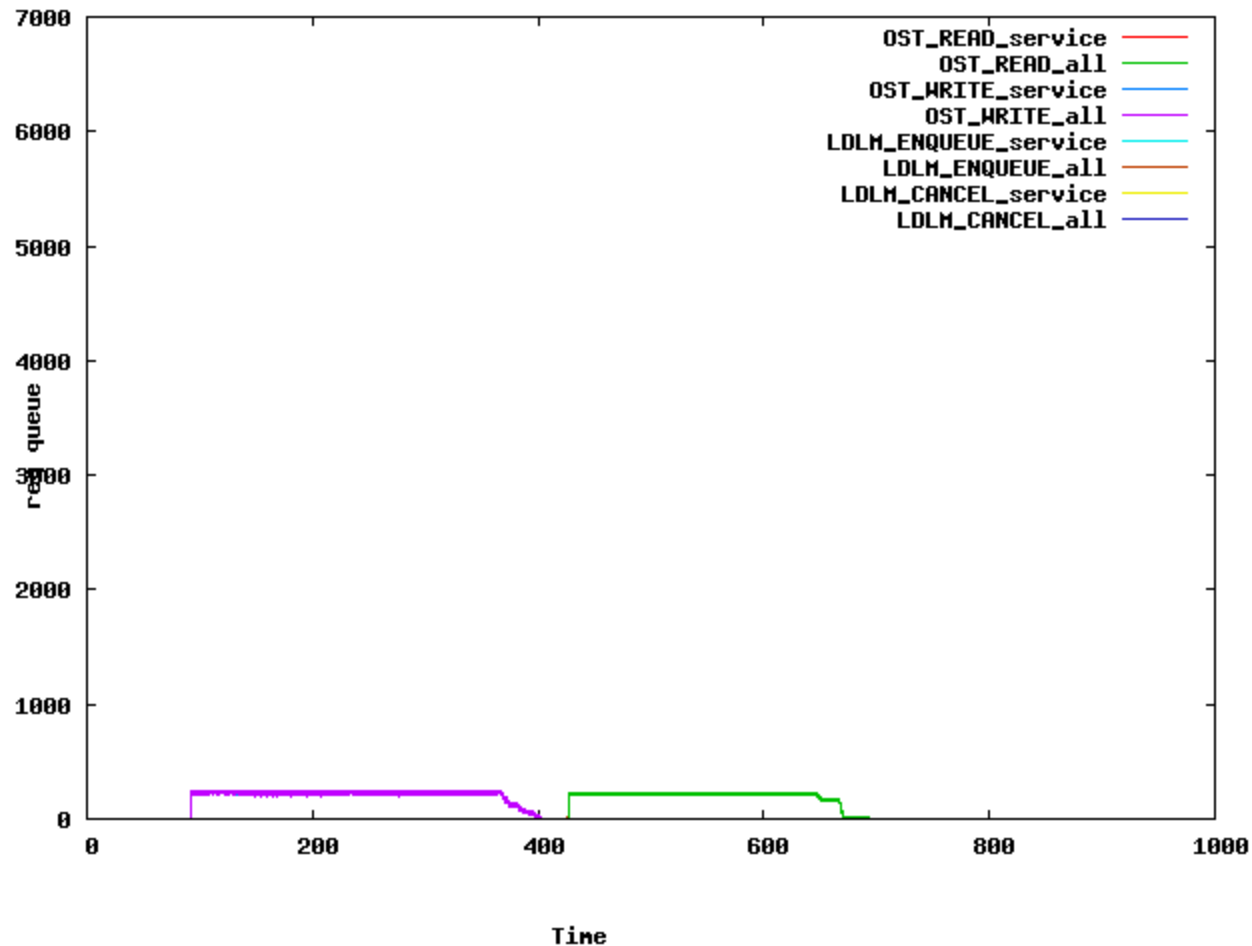
OSTs	Read	Write
144 OSTs	30638 MB/s	34743 MB/s
FCFS	46304 MB/s	46279 MB/s
FRR		

It shows significant performance improvement by simulation: 30%+

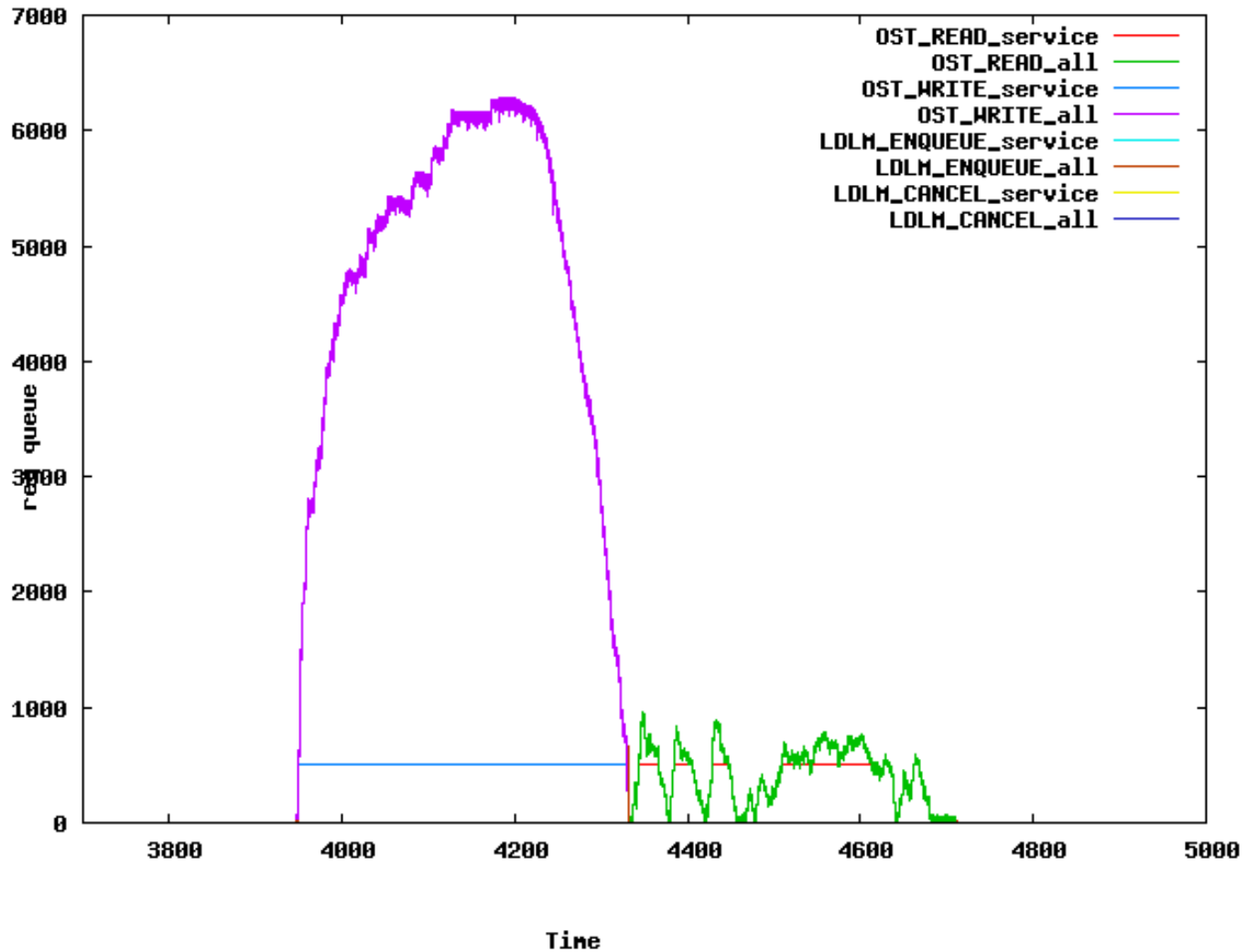
Appendix: other topics

- I/O analysis on Jaguar
- Adaptive timeout algorithm
- Scalability
- Performance virtualization

I/O analysis (FPP)



I/O analysis (SF)



I/O analysis

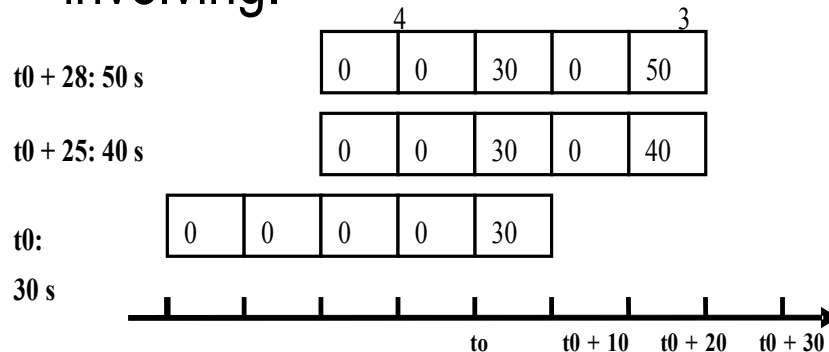
- Environment (small cluster)
 - > 112 clients, 4 OSS each is attached by 7 OSTs
 - > Running lustre-1.6.5
 - > IOR test with both SF and FPP mode
 - > Transfer size 1M, block size 1024M
 - > SF mode – stripe count = -1, stripe size = 1M
 - > FPP mode – stripe count = 1, stripe size = 1M
 - > Y axis means request load for each kind of RPC on OSS.
X axis means time.

I/O analysis

- Write (number of queued RPCs)
 - > SF mode – $112 \text{ (client)} * 8 \text{ (max in_flight)} * 7 = \sim 6100$
 - > FPP mode – $4 \text{ (client, 4 client for each OST)} * 8 \text{ (max_in_flight)} * 7 \text{ (OST number)} = \sim 200$
- Read
 - > Read ahead – the maximal read-ahead window is 40M (default). It can only send $(40/28)$ 1 or 2 RPCs to each OST by grouping I/O (synchronous processing).
- Question ?
 - > Read beats write in both FPP and SF.

Adaptive timeout algorithm

- RPC RTT is divided into two parts:
 - > $RTT(rpc) = T(net) + T(service)$
 - $T(net)$ denotes network latency
 - $T(service)$ denotes service time on OST. it is relatively large compare with $T(net)$ especially when server loading is involving.



Sliding time window algorithm (STW)

time window length: $H = 50s$

N sub time windows: $N = 5$, each with a record value

time step = 10s

AT: Service time estimating algorithm

- **MAX** algorithm – $STW = (H, T, v[N])$
 - > Estimate($T(\text{service})$) = $MAX(v[i]), i = 0, 1, 2, \dots, N-1$.
- Line least square curve fitting (**LCF**) algorithm

the recording value is a time-value couple in the form $\langle t_i, v_i \rangle$ ($i = 0, 1, \dots, N-1$) where the t_i is the RPC's $T_{arrival}$ and v_i is the RPC's service time; it records the value with max service time adding in the i th sub time window.

predicts the current service time according to trend of past RPCs' service times, it is much smarter than MAX algorithm.

$$Estimate(v) = f(t) = a_0 + a_1 * t;$$

$$a_1 = \left[\sum_{i=0}^{N-1} t_i v_i - \left(\sum_{i=0}^{N-1} t_i \right) \left(\sum_{i=0}^{N-1} v_i \right) / N \right] / \left[\sum_{i=0}^{N-1} t_i^2 - \left(\sum_{i=0}^{N-1} t_i \right)^2 / N \right];$$

$$a_0 = \left(\sum_{i=0}^{N-1} v_i \right) / N - a_1 \left(\sum_{i=0}^{N-1} t_i \right) / N$$

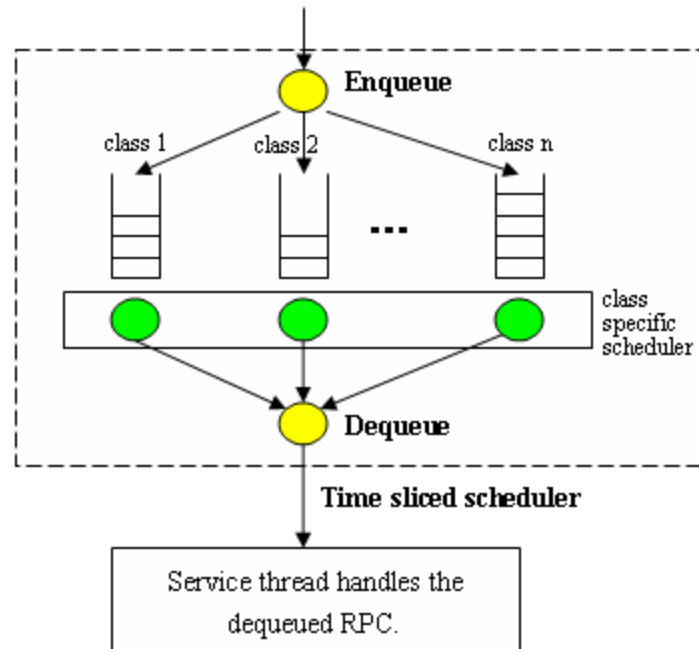
AT: Service time estimating algorithm

- Estimated average execution time (**AET**) algorithm
 - > Record form: $\langle C(i), W(i) \rangle$
 - $C(i)$: number of RPCs finished in the i th sub window
 - $W(i)$: the amount of time server is in work in the i th sub window
 - > Average RPC execution time
 - $aet(i) = W(i) / C(i)$;
 - > The estimated average execution time
 - $Estimate(aet) = \text{MAX}(aet(i)), i = 0, 1, 2, \dots, N-1$
 - > Estimate service time
 - $Estimate(T(\text{service})) = Estimate(aet) * nr_queued_rpcs.$

AT

- AT algorithm simulation:
 - > burst workload to 1 OST from 32000 clients.
 - > Each client sends 4 I/O RPCs with 1M data.
 - > Clients are divided into 16 sets in average.
 - > The time skew between the client sets is 5s.
 - > Time-out rate
 - Fixed: 86%
 - MAX: 40%
 - LCF: 9%
 - AET: 9%

Performance virtualization



- It achieves performance virtualization by time sliced NRS together with a set of application-dependent NRSes.

Performance virtualization

- At the coarser granularity, the time slice NRS provides virtual slices of the shared I/O resource to the different applications (or client classes);
- At the finer granularity, during each of an applications' time slices, its scheduler determines the “order” in which I/O requests are dispatched from its class specific NRS' request queue.
- The former provides fairness and performance isolation across the different applications; while the latter, in conjunction with the former, aligns service with application data delivery requirements.

Performance virtualization

- To verify time sliced NRS, the class specific application-dependent scheduler uses FCFS algorithm, we sample the aggregate I/O bandwidth in certain time interval.
- Parameters:
 - > Disk unplug: 6 ms
 - > Client count: 1000
 - > Access mode: FPP, write only
 - > Client class number: 2 (class A, B, each has 500 clients)

Simulation result

Parameters:

Time epoch: 10 sec

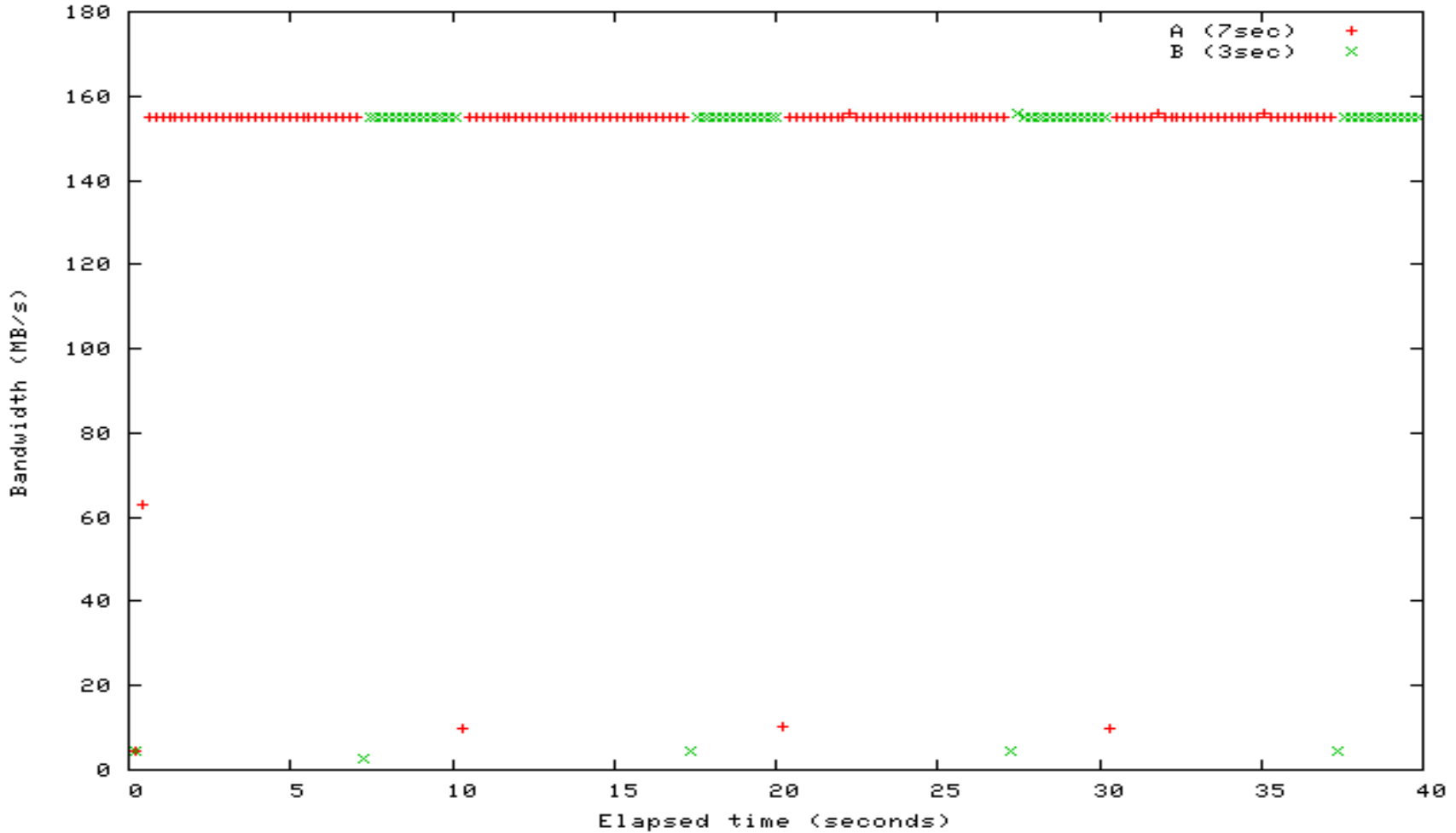
time slice of client class A: 7 sec

time slice of client class B: 3 sec

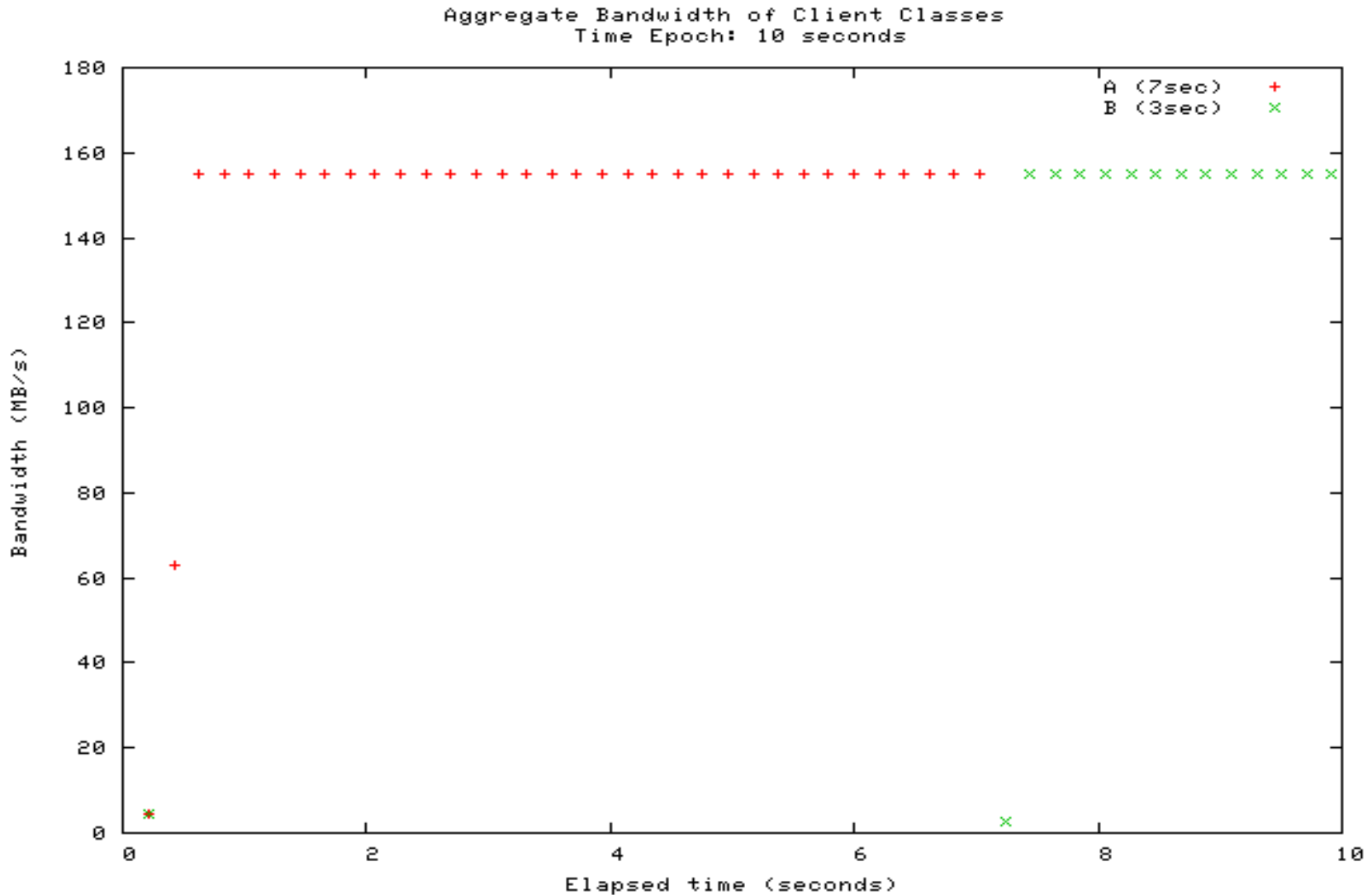
sample interval: per 0.1 sec

Simulation result

Aggregate Bandwidth of Client Classes
Time Epoch: 10 seconds

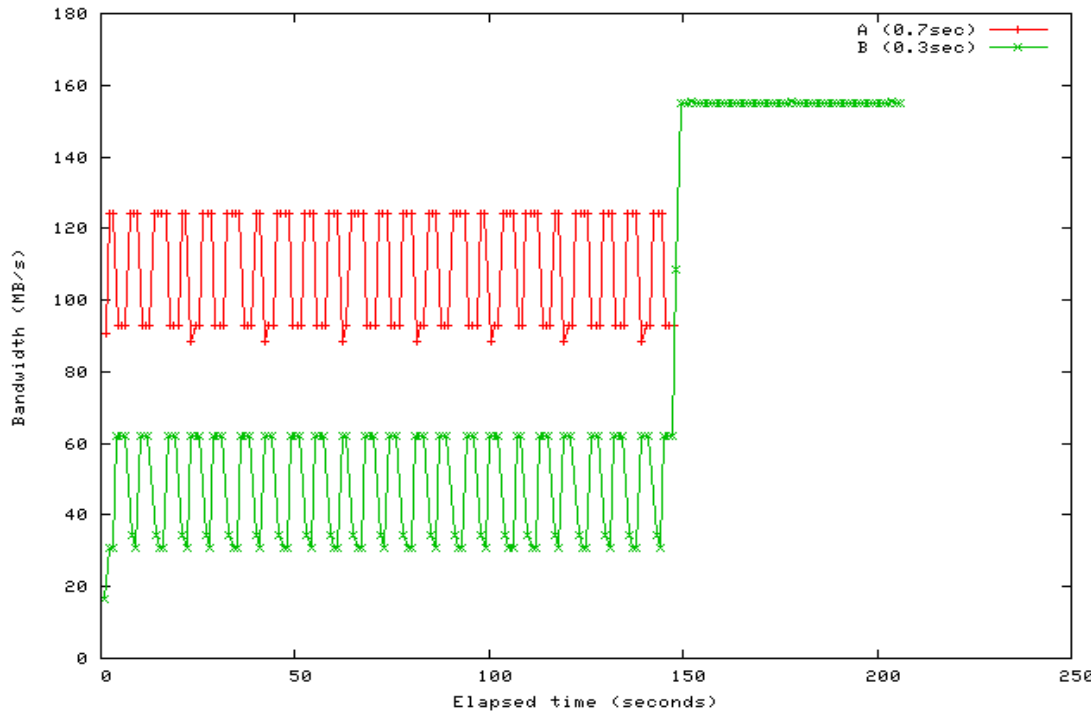


Simulation result



Simulation result

Aggregate Bandwidth of Client Classes
Time Epoch: 1 second



Parameters

time epoch: 1s

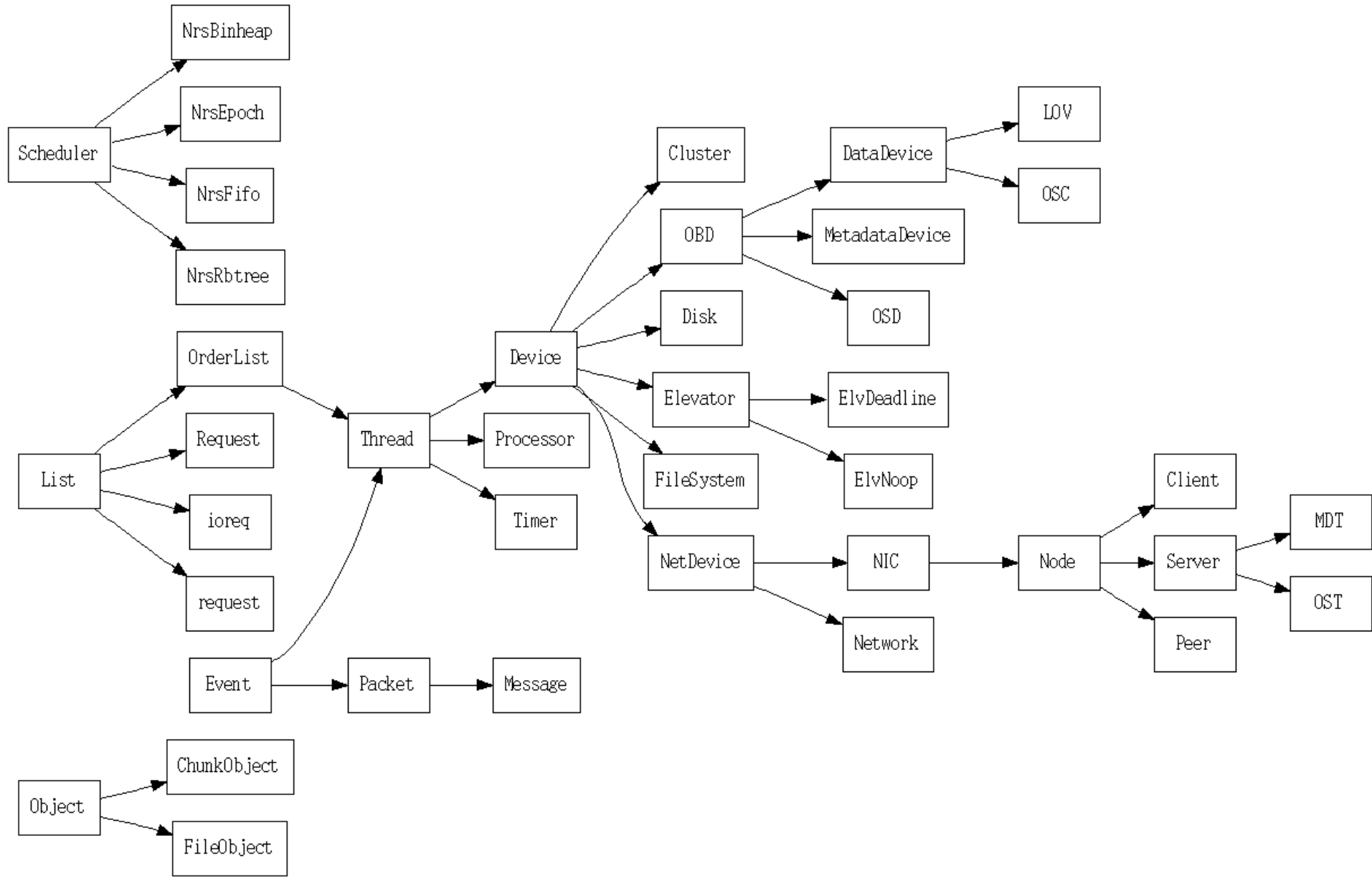
time slice of client class A: 0.7s

time slice of client class B: 0.3s

sample interval: per 1 sec

The three graphs above show that our time sliced algorithm can work well for bandwidth control at coarser granularity level; but it's not very stable for small time epoch.

Lustre Simulator (class inheritance)



Lustre simulator

- It simulated the components:
 - > Disk driver, RAID0 block device
 - > Linux I/O scheduler: Noop and deadline algorithms
 - > File system, Mballocc block allocation algorithm
 - > Network, NIC
 - > Three Lustre subsystems: Client (OSC/LOV), OST, MDT
 - > Network Request Scheduler (NRS) module together various algorithms
 - > Thread, Thread pool module
 - > Ptlrpc layer, adaptive time-out
 - > Support Direct I/O and simple Open/Create path

Lustre simulator

- Client module consists of LOV/OSC modules together with user applications to launch I/Os and wait for their completions.
- LOV splits the I/O requests into various OSCs according to offset and stripe size.
- Server (OST/MDT) has a backend file system and a thread pool with tunable thread count that can execute the requests in parallel.
- Each server has a Network Request Scheduler (NRS) to manage the incoming RPC requests according to the preset NRS algorithm.

Lustre simulator

- MDT implements a simple Open/Create meta-data path.
- OST implements the basic READ/WRITE/PING state machine.
- Implement a Ptlrpc layer for the design and improvement of AT algorithms.
- Details on Lustre simulator can be found in https://bugzilla.lustre.org/show_bug.cgi?id=13634.

Comparison and analysis (Jaguar)

- Experiment:
 - > xfersize varying from 64k to 1M,
 - > aggregate file size 1024GB,
 - > IOR FPP access mode.

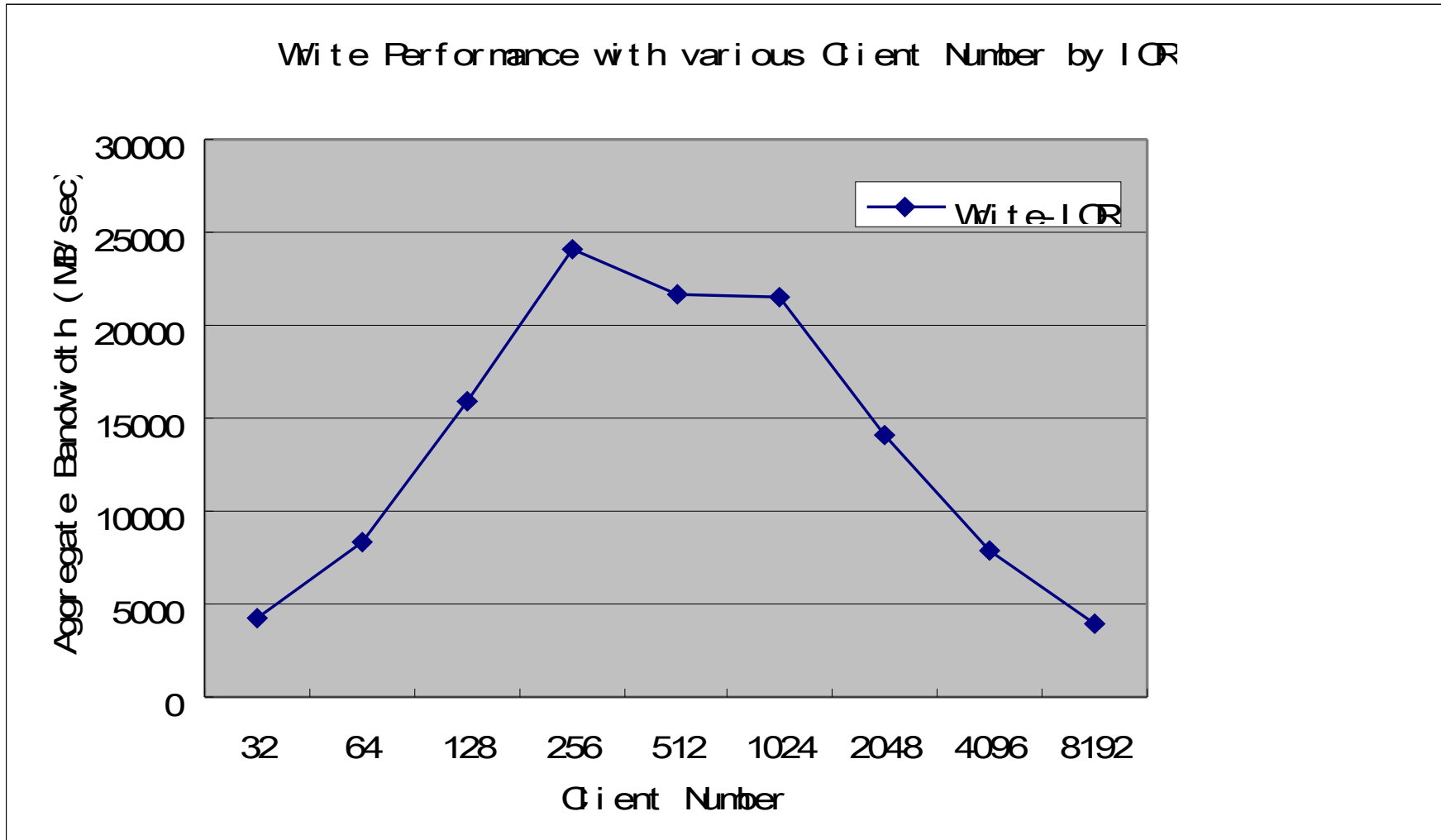
	Jaguar (GB/s)	Simulator (GB/s)
READ (1M)	33.7	33.1
WRITE(1M)	30.3	30
READ(64K)	2.3	3.8
WRITE(64K)	7	3.9

The reason of the difference with small xfersize 64k:

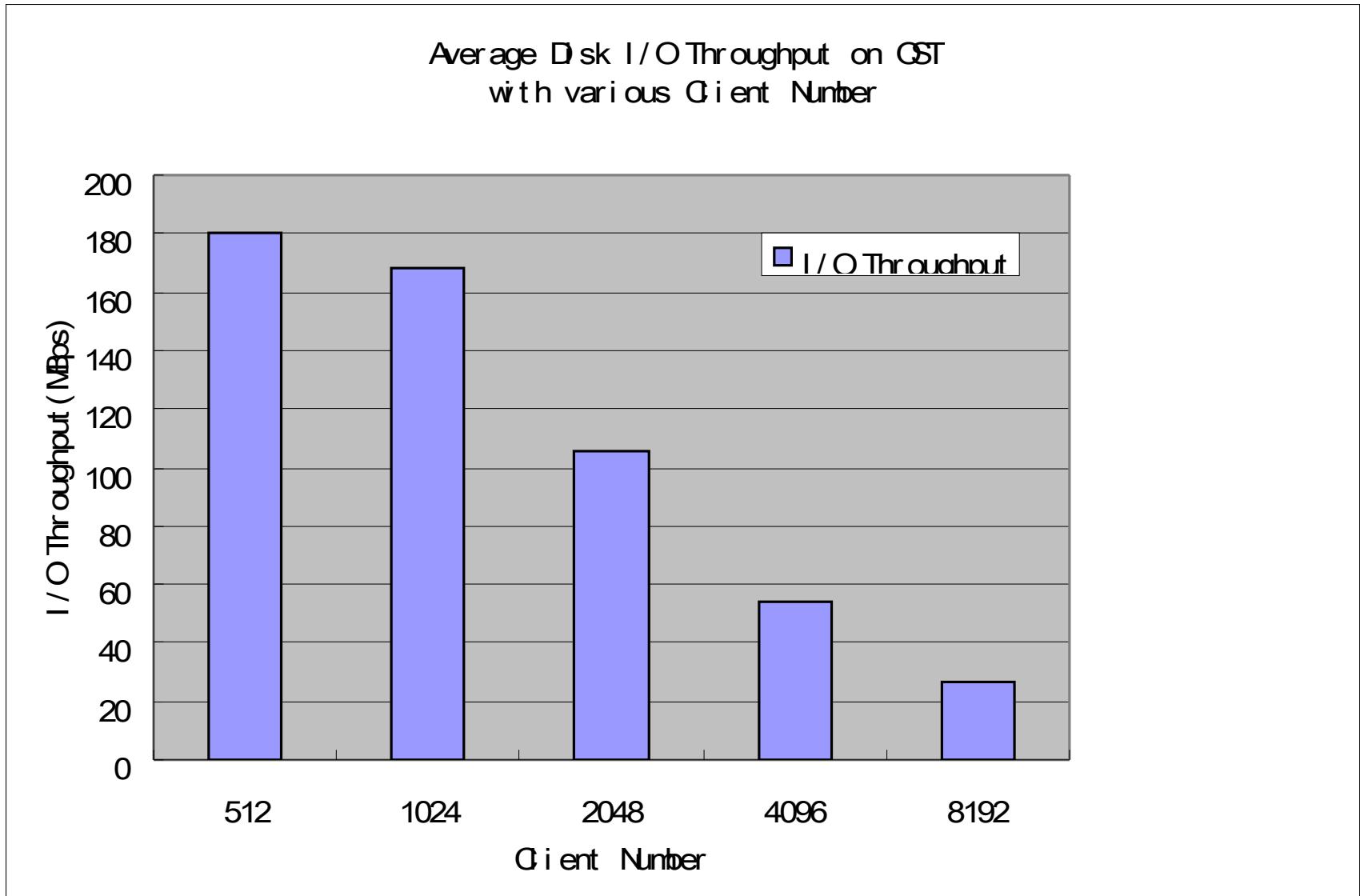
S2A 9550 DDN storage has very big hardware cache. It considers as completion when data write into disk cache; for random small read it results in cach miss and needs to read directly from hard disk. But the simulator doesn't implement the disk cache subsystem.

The general trends of two curves are almost same.

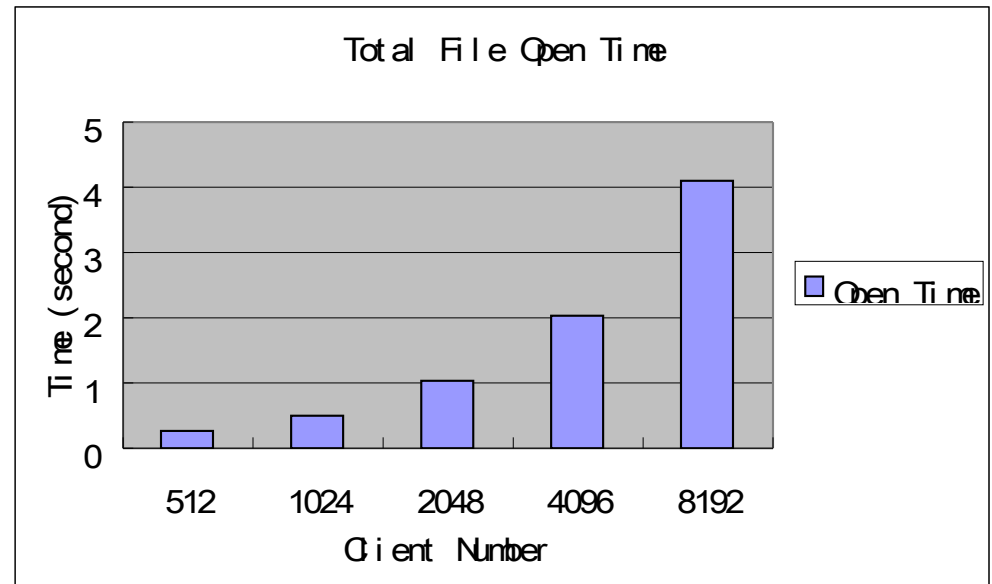
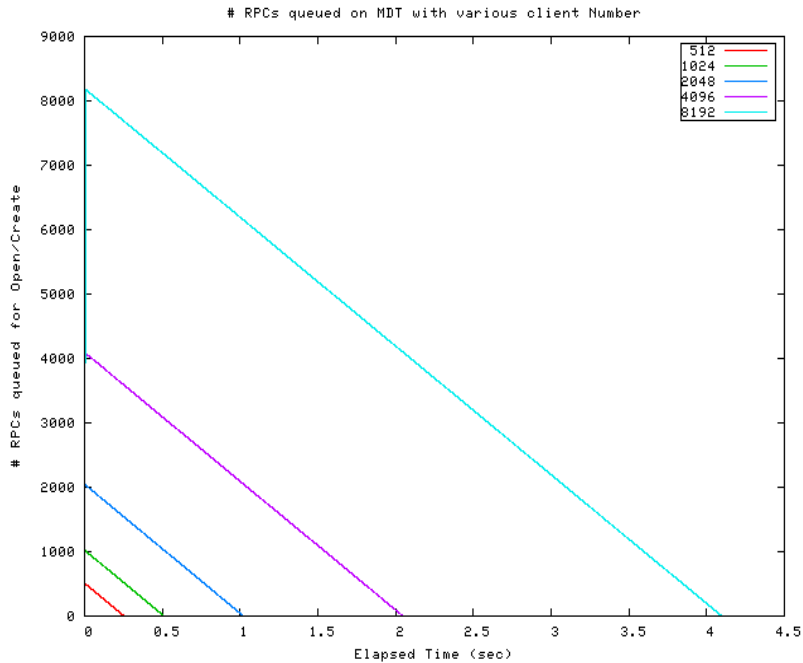
Comparison and analysis (Scalability)



Comparison and analysis (Scalability)



Comparison and analysis (Scalability)



Comparison and analysis (Scalability)

- Simulation environment
 - > 144 OSTs, 1 MDS,
 - > Client number varying from 32 to 8192.
 - > Aggregate size of files kept constant at 16GB
- The reason of aggregate performance drop:
 - > As the number of clients increases, the I/O throughput per client decreases and the I/O throughput to backend disk is not enough to make full use of its bandwidth.
 - > Increased time in opening/creating the separated files in FPP mode.



Thanks & Questions

Yingjin.Qian@sun.com



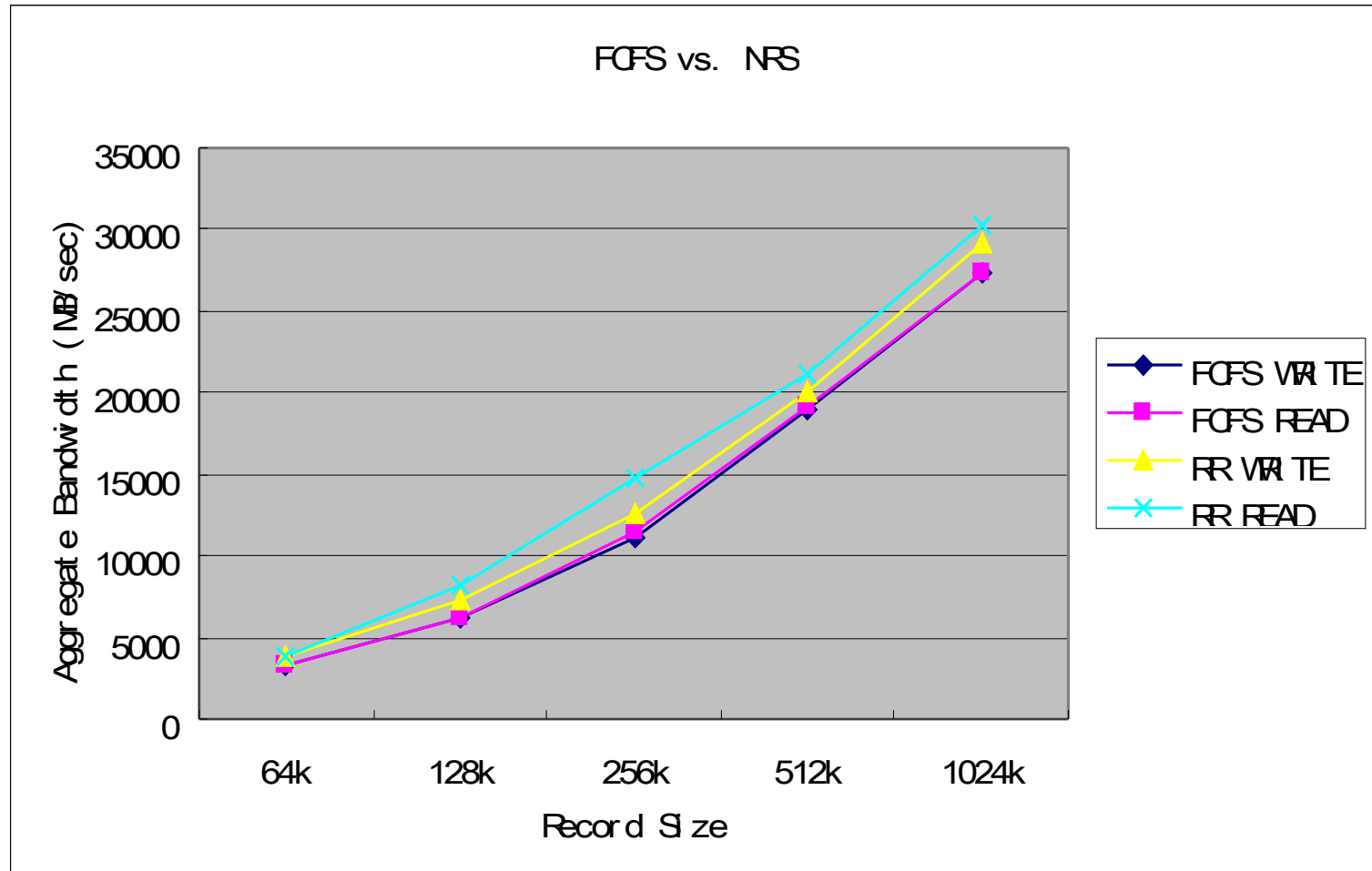
Network Request Scheduler (NRS)

- Multiple block allocation algorithm (used by Idiskfs)
 - > Optimize the concurrent block allocation for distributed parallel I/O.
 - > Extent based allocation, large allocation (1MB at a time)
 - > Pre-allocation size depends on the total size (S) in first try goal:
 - $S \in (1M, 4M]$, pre-allocation size 2M;
 - $S \in (4M, 8M]$, pre-allocation size 4M;
 - $S > 8M$ and the requested allocation size is less than 8M, it's 8M.

NRS (greedy algorithm)

- Make the I/O requests belong to same data object as close as possible.
- Greedy algorithm:
 - > Use rbtree to manage all queued RPCs.
 - > I/O requests sort by object id and then offset in object.

NRS (greedy round robin algorithm)



NRS (simple round robin algorithm)

