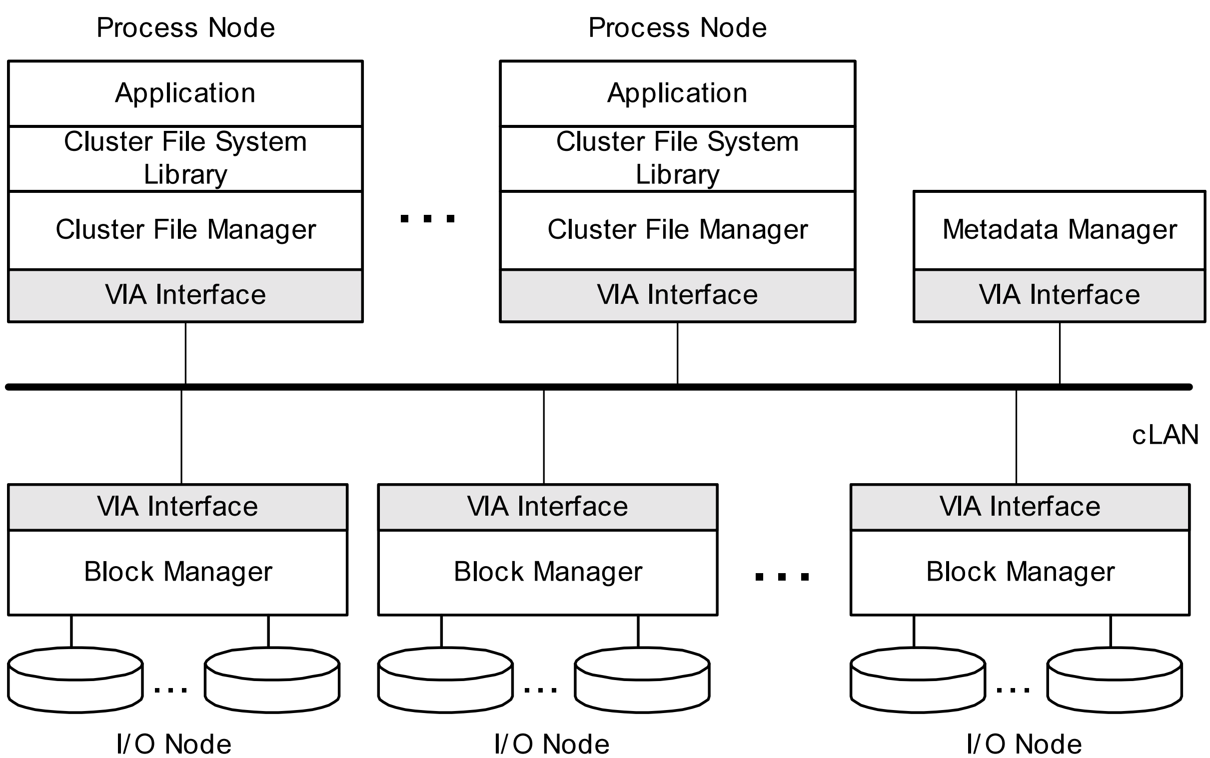
PFSL

- *P*arallel *F*ile *S*ystem for *L*inux clusters
- *Communications are in base VIA*
- Implemented in user level
- PFSL library
  - The applications must be coded in PFSL library to facilitate PFSL service
  - Several functions are defined

    pfsl_open(), pfsl_close(), pfsl_read(), pfsl_write(), pfsl_lseek(), etc.

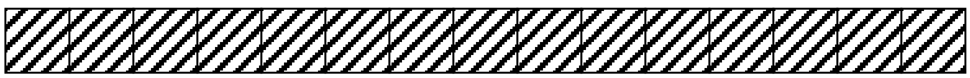
Configuration of PFSL

- Cluster File Managers(CFMs)
  - providing an interface to applications
- Block Managers(BMs)
  - store file blocks and service them to File Servers
- Metadata Manager(MM)
  - maintain metadata of whole file system

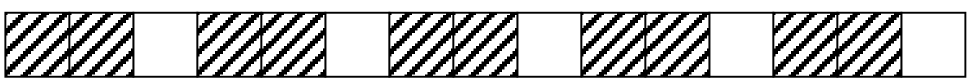


Characteristics of Parallel Scientific Workloads

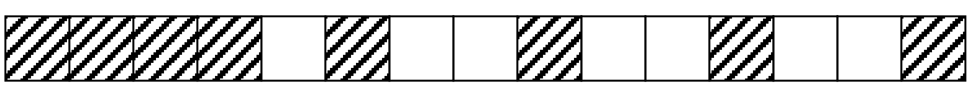
- Sequential Access Patterns



- Interleaved Access Patterns
  - Mostly simple-strided access



- Mixed Access Patterns
  - Sequential + Interleaved



☞ We made & used test workloads based on these results

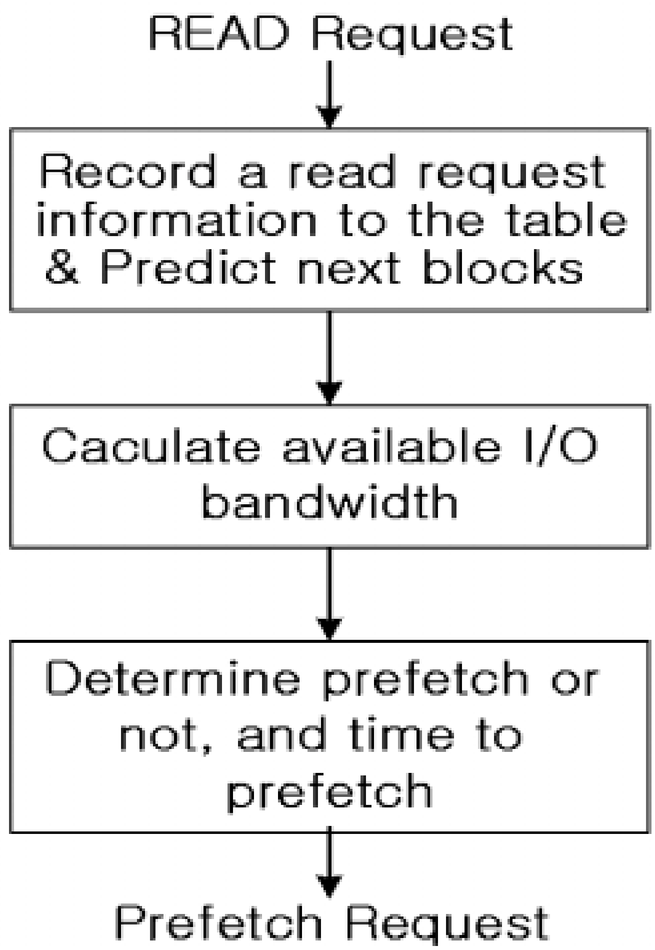
Problems of previous prefetching policy

- OBA(One-Block Ahead) Prefetching
  - Poor performance with not sequential accesses
- Hint-based Prefetching
  - Needs additional coding
  - Poor compatibility
- File access history-based
  - Heavy cost of maintaining access logs and predicting data blocks

☞They do not consider the I/O situation when the system try to prefetch.

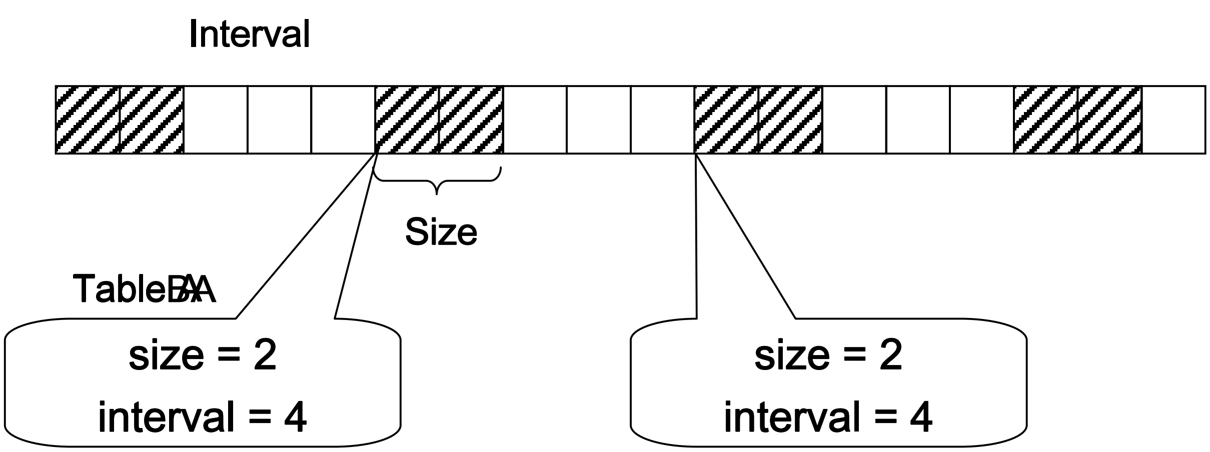
☞Need to determine whether or not to prefetch, plus the time to prefetch

Table-Comparison Prefetching Procedure

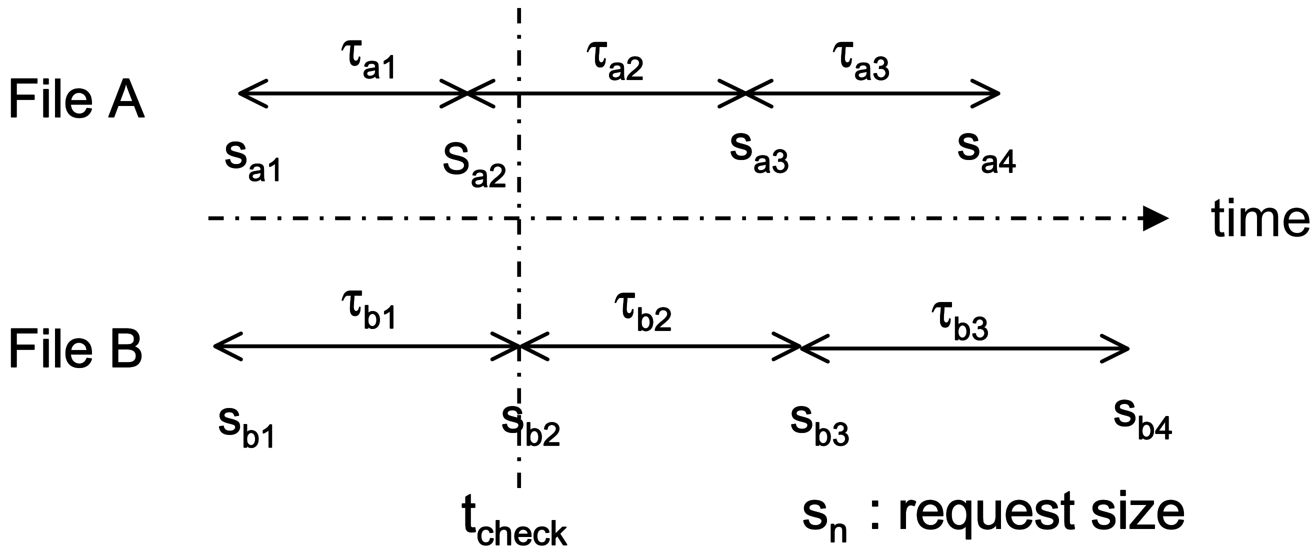


Predicting Data Blocks

- Record the request information to 2 tables, in turns
- If the size & the interval of 2 tables are equal
  - **We assume the access pattern is simple strided**
  - **Prefetch the next data block**



Calculating the Available I/O Bandwidth



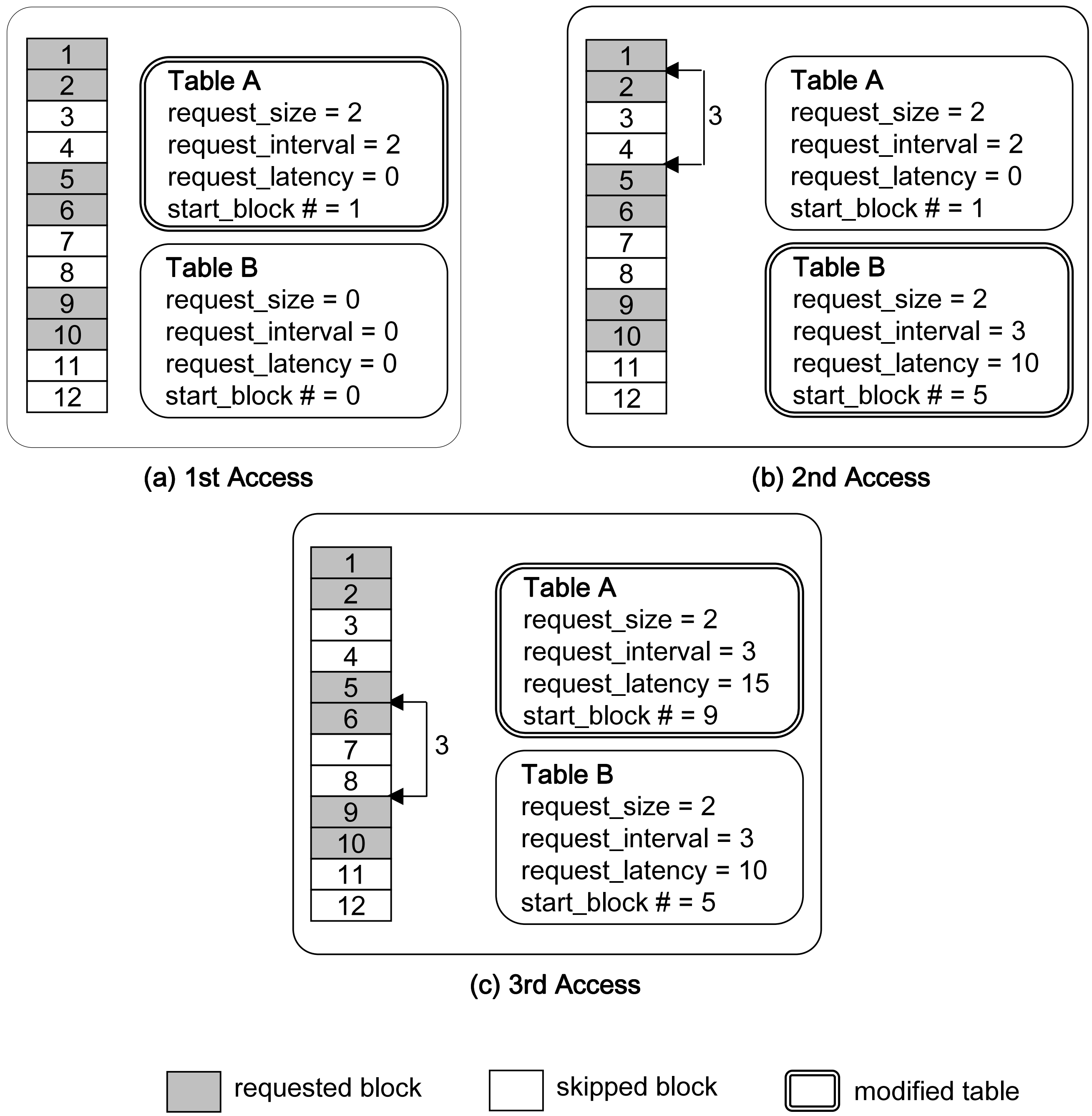
s_n : request size

τ_n : request latency

- BW_{used} (at t_{check}) = $s_{a2} / \tau_{a1} + s_{b2} / \tau_{b1}$
- BW_{avail} (at t_{check}) = $BW_{max} - BW_{used}$

8

Calculating the Available I/O Bandwidth



Prefetching Policy

①*if (BWavail < 0)* **: Excessive I/O Request**

No prefetching

②*else if (BWavail < 1/2 BWmax)* **: No Sufficient I/O B/W**

if (request_interval == 1) **: Conswcutive Request**

Enqueue current read request and prefetching to the file request queue together

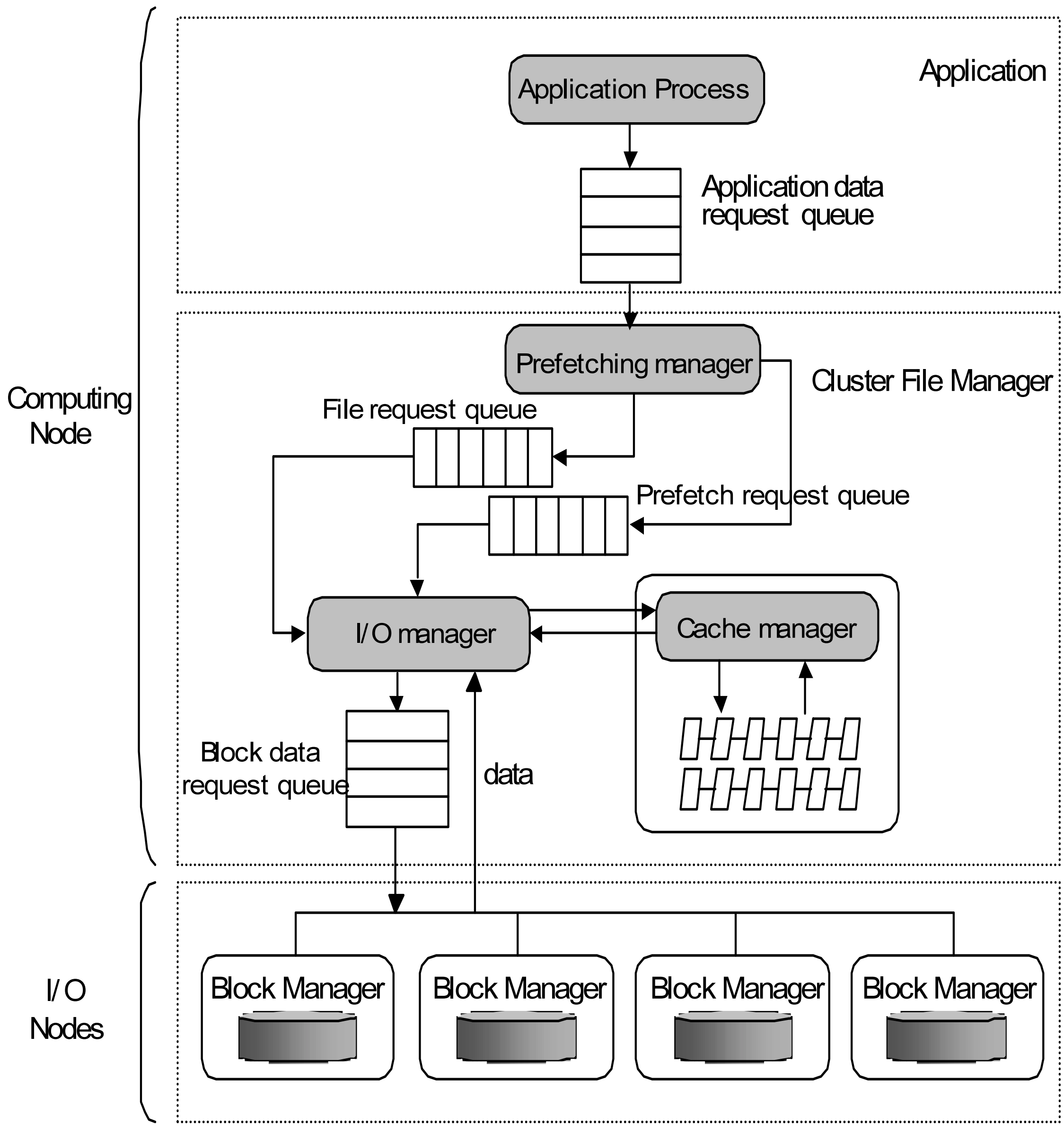
else

Enqueue prefetching to the prefetching request queue

③*else* **: Sufficient I/O B/W**

Prefetching to the prefetching request queue

Implementation of Table-Comparison Prefetching



Test Environment

- System Environment

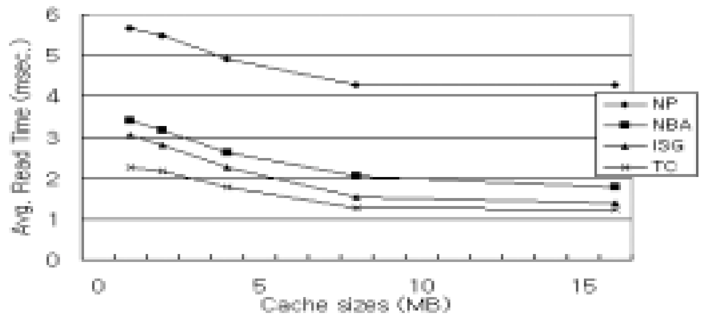
Parameters	Value
CPU	Intel Pentium Ⅱ 266Mhz
RAM	256 MB/s
O/S	Linux kernel 2.2.12
Compiler	GNU C++ ver 2.91.66

- Workload Characteristics

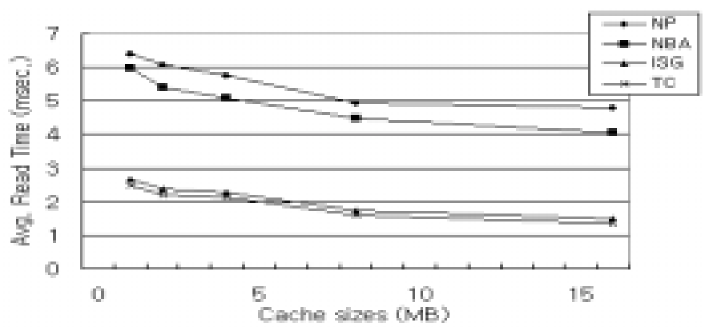
workload	Max. request rate(MB/s)	Total read size (MB)	Avg. read size per request(kB)	Number of total requests	Number of processes
LIGHT	11.2	29.7	48.4	700	3
MEDIUM	19.2	54.7	56	1000	4
HEAVY	32	103.9	62.6	1700	5

Test Result (1)

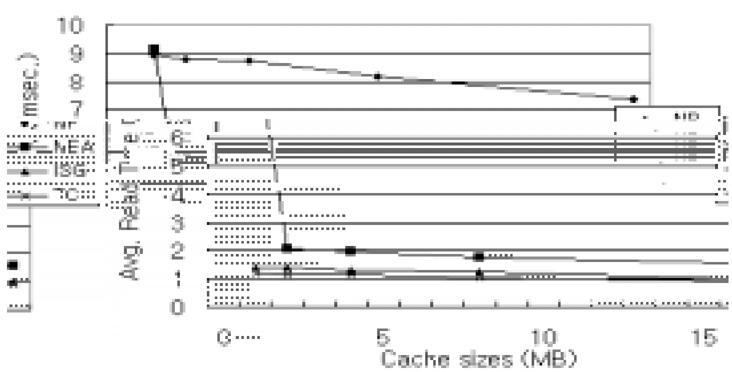
- Average Read Time per Request
  - Light Workload



  - Medium Workload

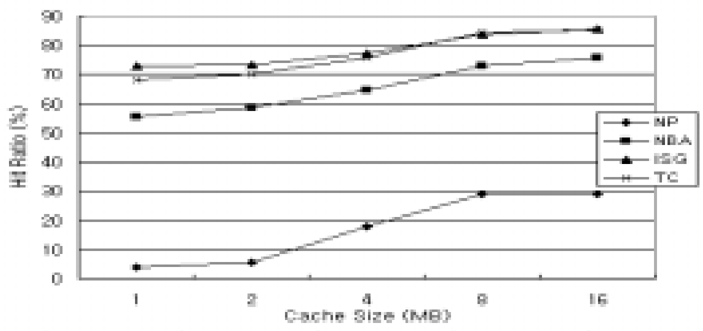


  - Heavy Workload

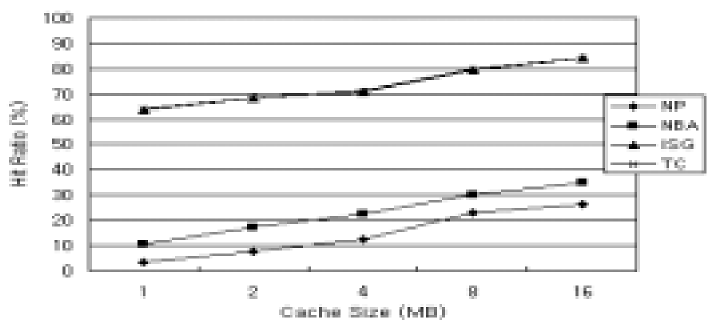


Test Result (2)

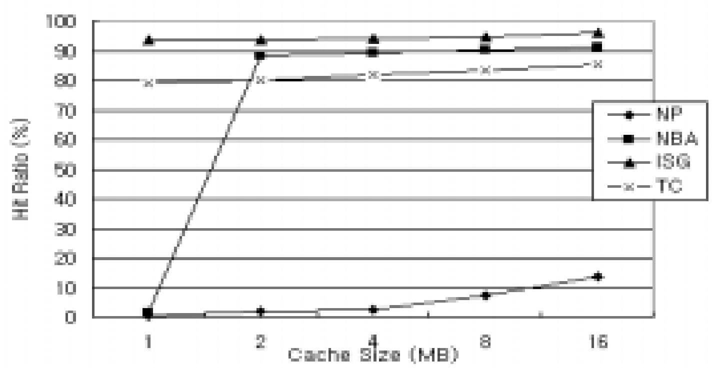
- Cache Hit Rate
  - Light Workload



  - Medium Workload

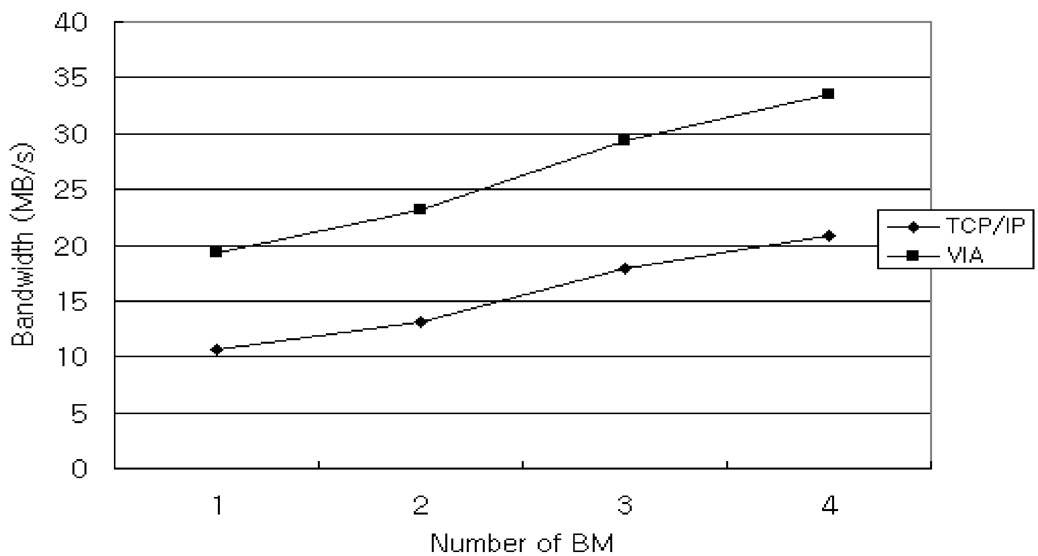


  - Heavy Workload



Test Result (3)

- Consecutive Read Performance Comparison
  (TCP/IP-based vs. VIA-based PFSL)



Conclusion

- VIA can relieve the communication overhead of traditional messaging protocol.
- Table-comparison Prefetching has simple algorithm.
  - ☞ Low System Cost
- Effective on the *Parallel Scientific Workload.*
- Using the Available I/O Bandwidth
  - – Determine whether or not to prefetch, and control the time to prefetch
  - – *Improve the Total System Performance !!!*