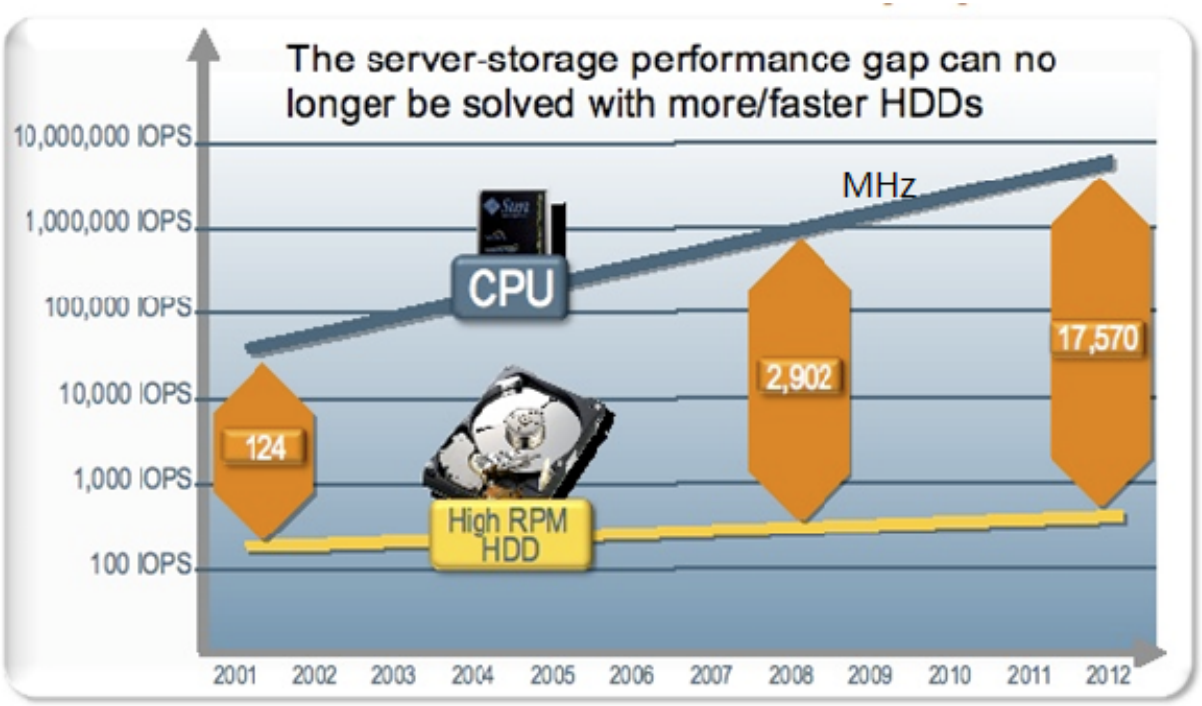
S4D-Cache: Smart Selective SSD Cache for Parallel I/O Systems

Shuibing He, Xian-He Sun, Bo Feng

Department of Computer Science

Illinois Institute of Technology

Speed Gap Between CPU and Hard Drive



http://www.velobit.com/storage-performance-blog/bid/114532/Living-With-The-2012-HDD-Shortage

ILLINOIS INSTITUTE OF TECHNOLOGY

Computing Becoming Data Intensive

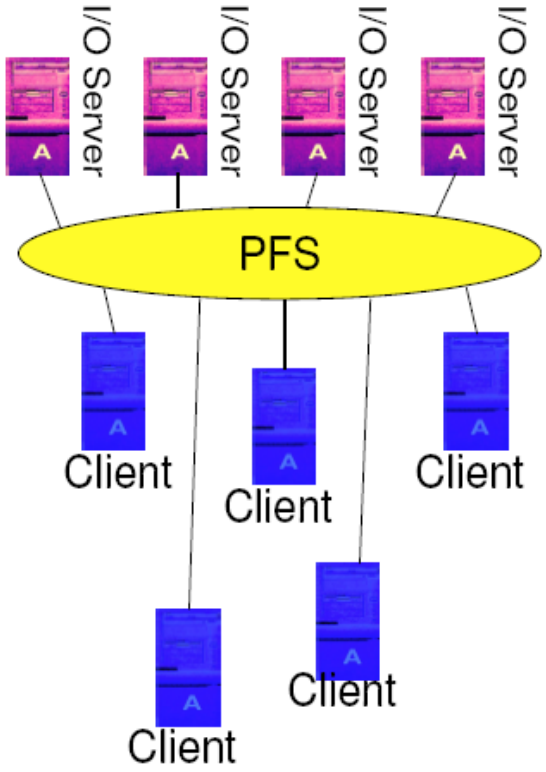
Selected 2012 INCITE applications at Argonne Leadership Computing Facility of ANL [1]

Project	On-line Data (TB)	Off-line Data (TB)
Combustion in Reactive Gases	1	17
CO2 Absorption	5	15
Seismic Hazard Analysis	600	100
Climate Science	200	750
Energy Storage Materials	10	10
Stress Corrosion Cracking	12	72
Nuclear Structure and Reactions	6	30
Reactor Thermal Hydraulic Modeling	100	100
Laser-Plasma Interactions	60	60
Vaporizing Droplets in a Turbulent Flow	2	4

[1] R. Latham, R. Ross, B. Welch, and K. Antypas, "Parallel I/O in Practice," Tutorial of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2013.

Argonne NATIONAL LABORATORY

To Meet The High I/O Demands: PFS



PVFS

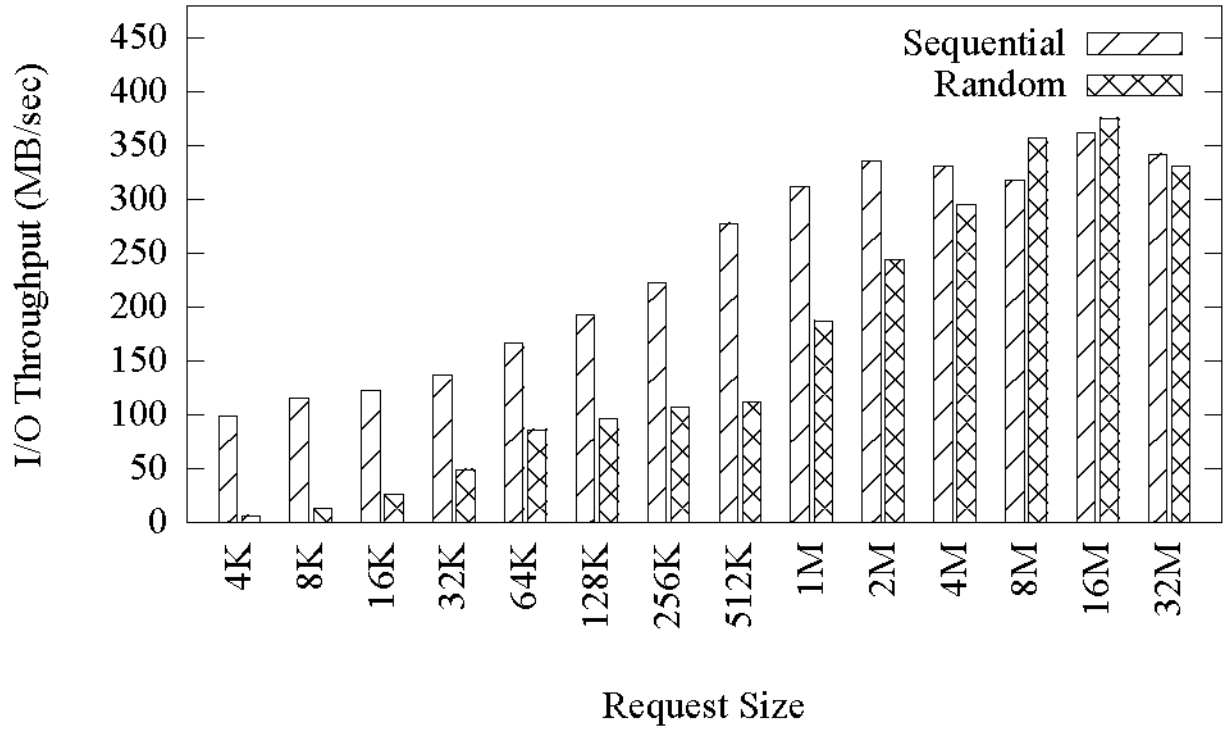
OrangeFS

lustre

panasas

IBM GPFS

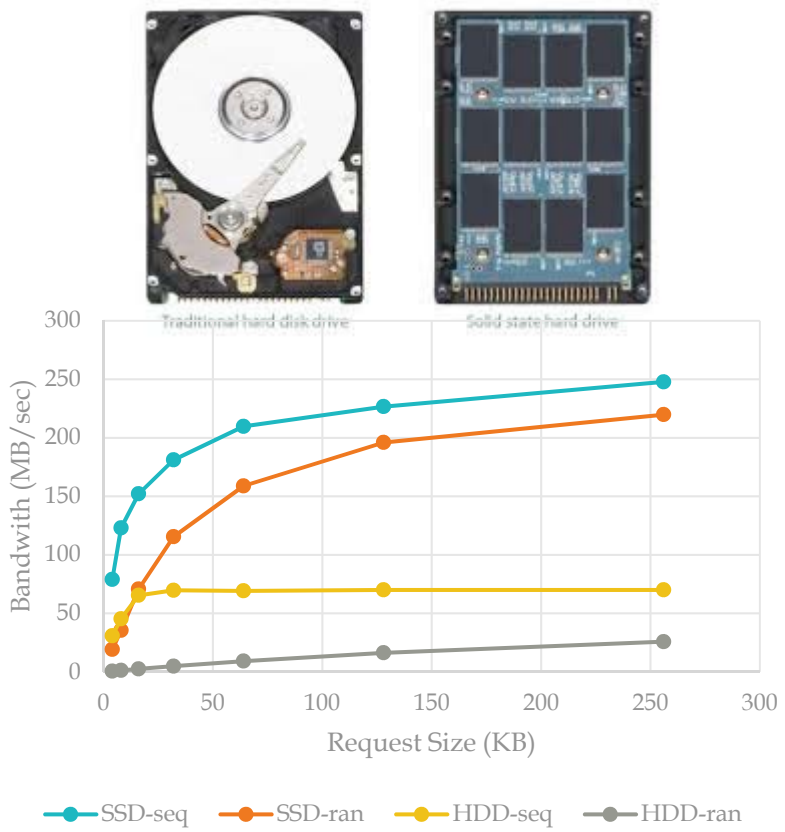
Poor performance of PFSs



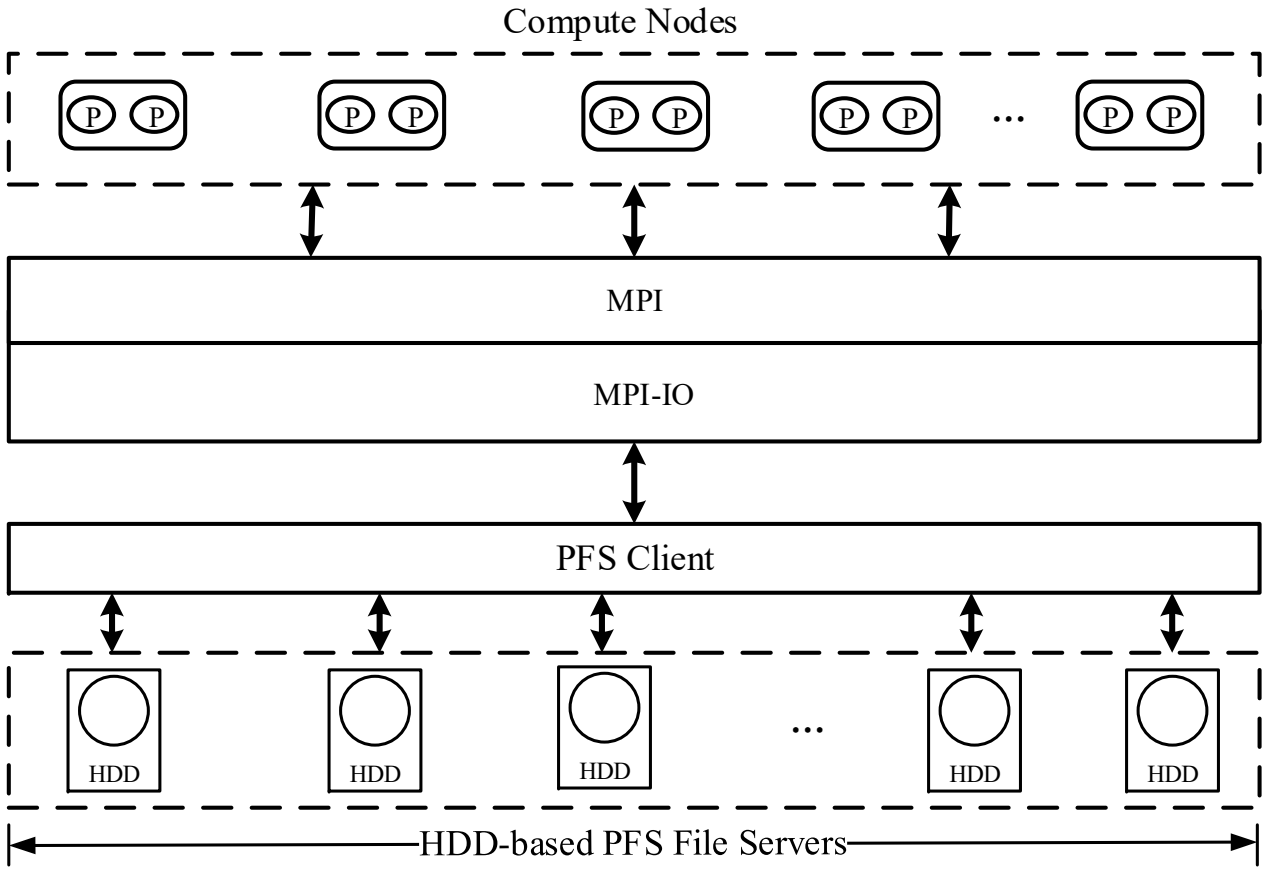
SSD provides new solutions for parallel I/O systems

ILLINOIS INSTITUTE OF TECHNOLOGY

- Benefits of SSDs
  - Higher storage density
  - Lower power consumption
  - Smaller thermal footprint
  - Higher performance



Traditional Parallel I/O Systems



Problem

- How to use SSDs to improve the performance of hybrid I/O systems?
- Challenge
  - I/O workload can change
  - SSDs and HDDs have different performance characteristic's
  - The performance and cost trade off

Outline

ILLINOIS INSTITUTE OF TECHNOLOGY

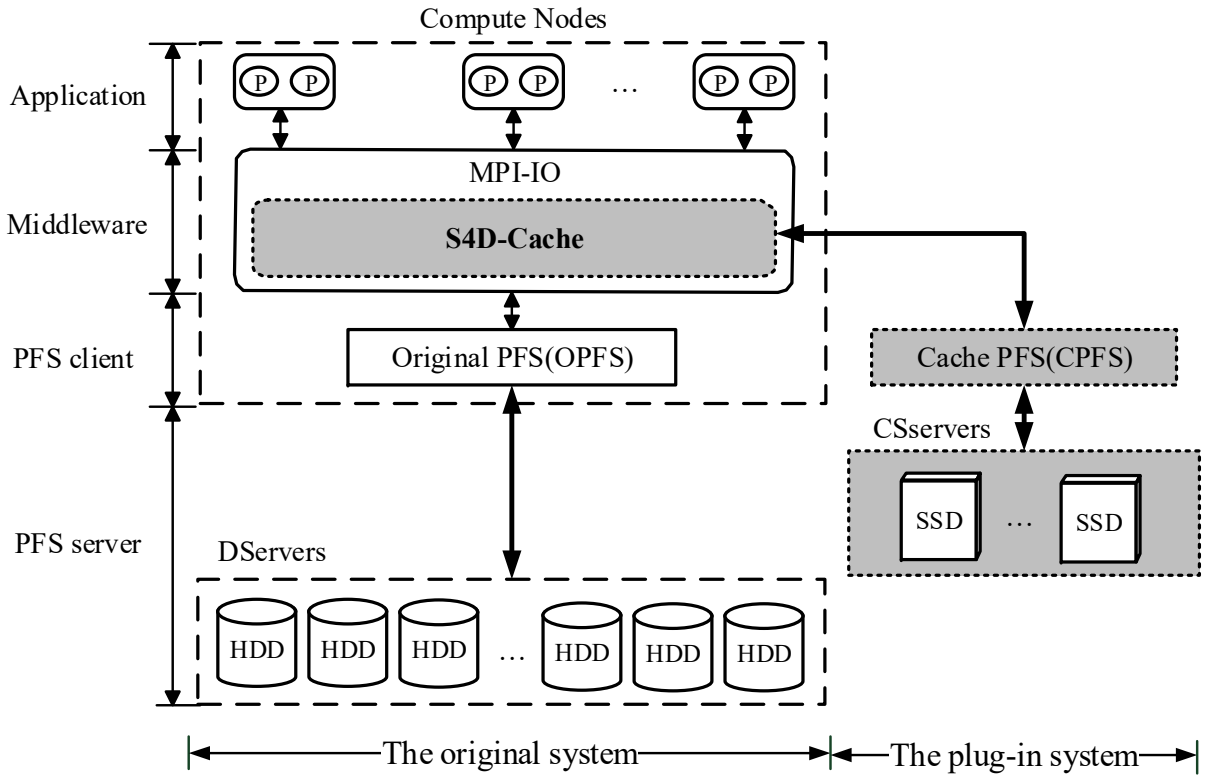
- Problem
- Our idea
- Design
- Evaluation

Key Idea: S4D-Cache

- Smart-Selective SSD Cache for Parallel I/O Systems
  - Integrate a small number of SSD-based file servers into the systems
  - Propose a centralizing SSD cache architecture
  - Selectively caching performance-critical data
    - Request parallelism
    - Request randomness

ILLINOIS INSTITUTE OF TECHNOLOGY

S4D-Cache Architecture Overview

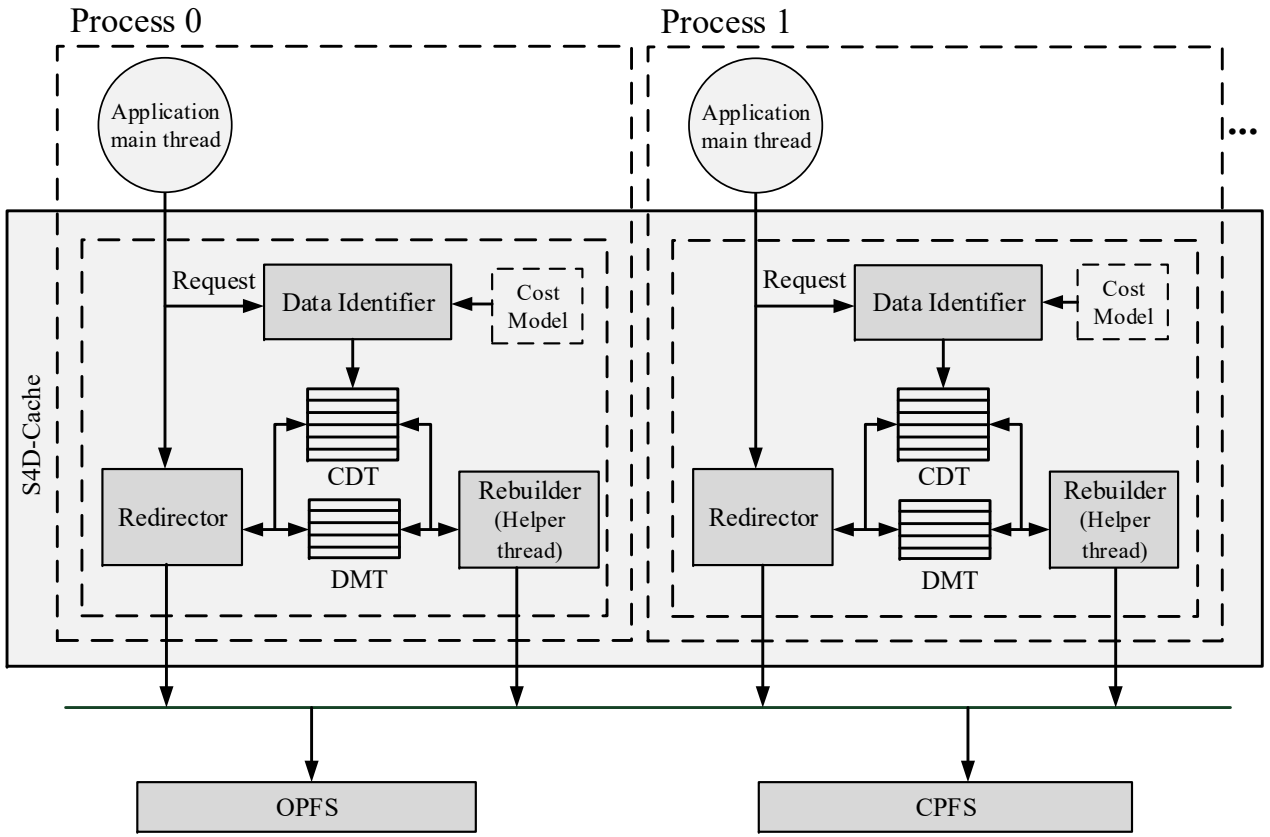


Design Benefits

- Key global data access information are accessible in this layer, and can be used to help improve performance
- The solution can support multiple file systems
- The plug-in design is transparent to applications
- Only a small cluster of SSDs are deployed into the system, which makes the design highly cost effective

ILLINOIS INSTITUTE OF TECHNOLOGY

Software structure



Data Access Cost Model

Table 1. Parameters (short in Pars) in cost analysis model.

Pars	Description
M	Number of HDD file servers
N	Number of SSD file servers (N<M)
str	Stripe size of parallel file system
d	Logical address distance between r_i and r_{i-1}
f	File offset of request r_i
r	Data size of request r_i
R	Average rotation delay for HDD
S	Maximum seek time for HDD
β_D	Cost of access one unit of data for HDD
β_C	Cost of access one unit of data for SSD

We calculate the data access cost for requests on DServers and CServers respectively.

ILLINOIS INSTITUTE OF TECHNOLOGY

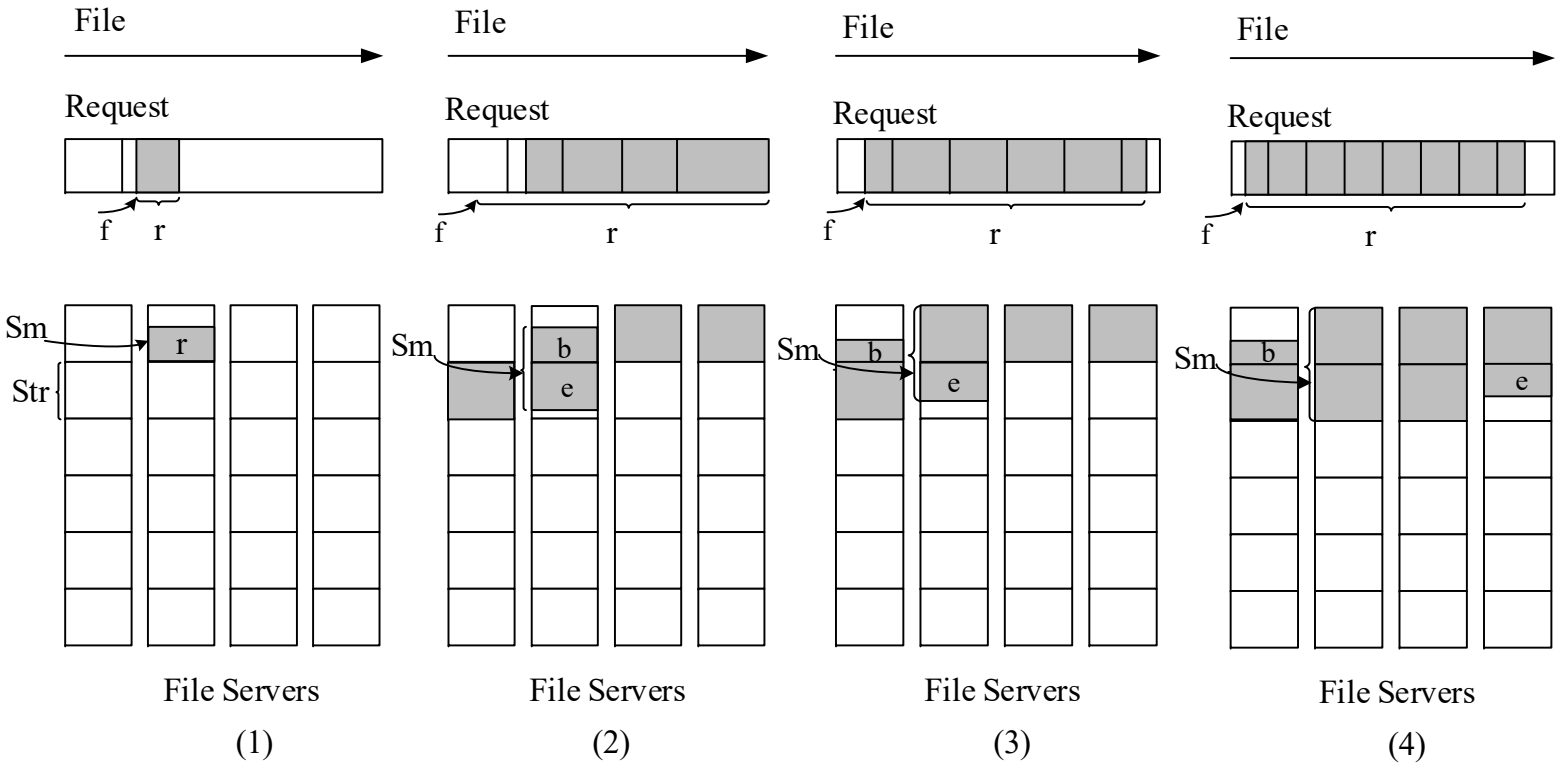
DServer Cost Model

- Request access cost T_D:
  - $T_D = T_s + T_t$ （1）
  - Ts: startup time, Tt: data transfer time
- Ts of sub-request
  - Involves seek time, rotation time on each server
  - T_S follows uniform distribution on [a,b]
    - a=F(d)+R, F(d) is a function for converting request distance d to seek [1]
    - b=S+R
  - $P(\alpha < x) = \frac{x-a}{b-a}, a \le x \le b$ （2）
- Ts of a file request on multiple DServers
  - Ts is the maximum of all the m servers
    - $X = \max(\alpha 1,\ \alpha 2,\ \ldots, \alpha m)$
    - $T_s = \int_a^b xf(x)dx = a + \frac{m}{m+1}(b-a)$ (4)

ILLINOIS INSTITUTE OF TECHNOLOGY

T*t* depends on sub-request size

Four cases of a file request distribution on DServers



ILLINOIS INSTITUTE OF TECHNOLOGY

Maximal sub-request size of a request (Sm)

TABLE II
MAXIMUM SIZE OF SUB-REQUEST IN DIFFERENT FILE ACCESSES CASES.

Case	Maximum size of sub-request (s_m)	Conditions
1	r	$\triangle = 0$
2	$max\{b + e + (\lceil \frac{\triangle}{M} \rceil - 1) * str,$ $\lceil \frac{\triangle}{M} \rceil * str\}$	$\triangle > 0 \& \triangle \% M = 0$
3	$max\{b + (\lceil \frac{\triangle}{M} \rceil - 1) * str,$ $e + (\lceil \frac{\triangle}{M} \rceil - 1) * str\}$	$\triangle > 0 \& \triangle \% M = 1$
4	$\lceil \frac{\triangle}{M} \rceil * str$	otherwise

- $T_t = s_m * \beta_D$ (5)

SServer Cost Model

- The startup time is ignored (High-end SSDs).
- Only use the data transfer time.

$$T_C = s_n * \beta_C \quad (7)$$

Critical data identification

ILLINOIS INSTITUTE OF TECHNOLOGY

- Performance benefit (B):
  - $B = T_D - T_C$ (8)
- A positive B means the request should be served at CServers
- Implications
  - Large request should be placed on DServers due to high parallelism
  - Random small requests should be placed on CServers

Cache metadata management

- Key data structure
  - Critical data table (CDT)
  - Data mapping table (DMT)

CDT

…			
D_file	D_offset	Length	C_flag
D_file	D_offset	Length	C_flag
…			

DMT

…					
D_file	D_offset	C_file	C_offset	Length	D_flag
D_file	D_offset	C_file	C_offset	Length	D_flag
…					

Selective Cache Algorithm

- Caching data on CServers based on three factors
  - The performance benefit (critical data)
  - Available free space on SServers
  - Type of I/O requests
- Read request is cached in a "lazy" way

Algorithm 1 Redirection Algorithm

Require: I/O Request: *req*, Data Mapping Table: DMT, Critical Data Table: CDT.

```
1:  if req misses in DMT then
2:    if req is write then
3:      if req is in CDT then
4:        find free space in CServers
5:        if free space is found then
6:          add new entry in DMT (mark dirty)
7:          change the req location as the DMT entry
8:        else
9:          find clean space in CServers
10:         if clean space is found then
11:           change the entry in DMT (mark dirty)
12:           change the req location as the DMT entry
13:         end if
14:       end if
15:     end if
16:   else
17:     if req is in CDT then
18:       set the C_flag of the entry in CDT
19:     end if
20:   end if
21: else
22:   change the req location as the DMT entry
23: end if
24: send request req
```

Data reorganization

- Functions
  - Flush dirty data back to DServers, indicating the data can be reclaimed
  - Read data from DServers to CServers, reset C_flag to 0 to show the data has been cached
- Reduce the interference to normal I/O requests
  - Multiple threads: main and help thread
  - Help thread issue low-priority I/O requests

Implementation

- Environments
  - I/O middleware MPICH2
  - PFS: PVFS2
- Cache metadata mapping table
  - Memory: Hash table
  - Storage file: Berkeley DB file on CServers
- I/O redirection module
  - MPI_File_open/read/write(),…
- Rebuilder
  - Multiple threads
  - Shared variables for communication between threads

Experimental Setup

ILLINOIS INSTITUTE OF TECHNOLOGY

	65-nodes SUN Fire Linux Cluster
CPU	Quad-Core AMD Opteron(tm) Processor 2376 * 2
Memory	4 * 2GB, DDR2 333MHz
Network	1 Gbps Ethernet
Storage	HDD: Seagate SATA II 250GB, 7200RPM SSD: OCZ PCI-E X4 100GB
OS	Linux kernel 2.6.28.10
File system	PVFS2 2.8.2

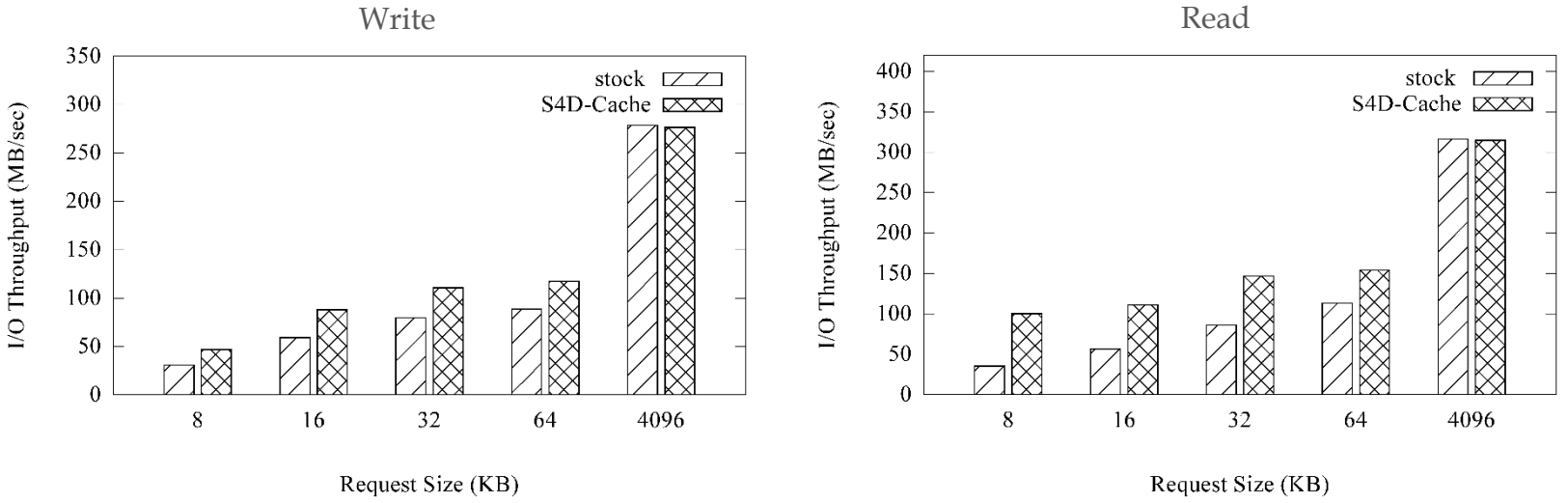
1: 32 computing nodes, eight HServers, four SServers
2: Stock system w/o S4D-Cache enabled
3: When S4D-Cache is enabled, the cache capacity is set to 20% of the application's data size.

Benchmark

- IOR
  - 10 instances of IOR are
    - six issue sequential I/O requests
    - remaining send random I/O requests
  - In each instance, default parameters
    - Access a shared 2GB file
    - 32 processes
    - request size is kept to 16KB
- HPIO
- MPI-Tile-IO

ILLINOIS INSTITUTE OF TECHNOLOGY

Varying Request Sizes



- For small requests, S4D-Cache significantly improve the I/O performance
- For large requests, less performance gains can be obtained
- Read has similar trend

Request distribution on different servers

ILLINOIS INSTITUTE OF TECHNOLOGY

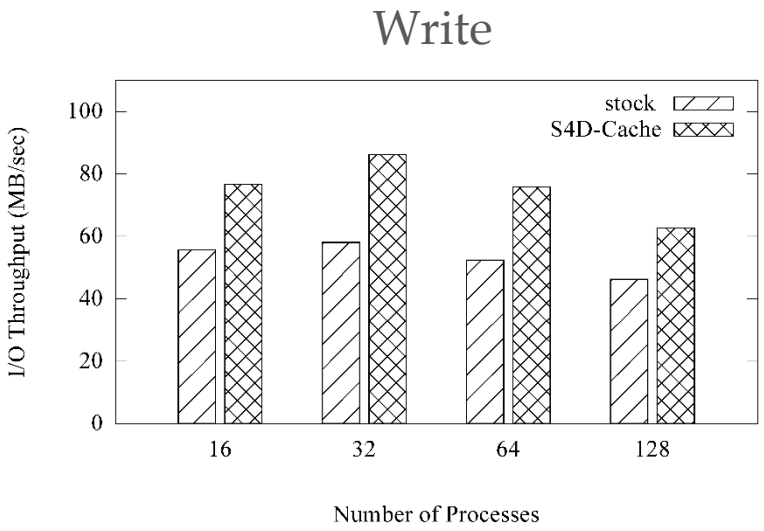
TABLE III
REQUEST DISTRIBUTION.

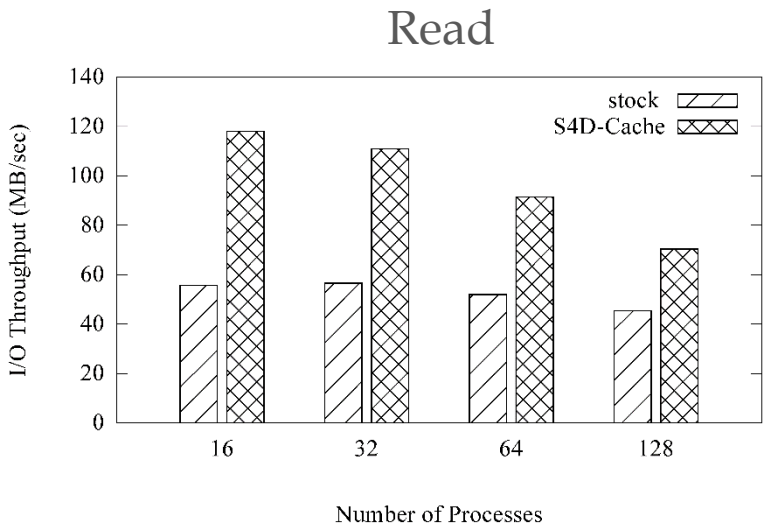
Request Size	DServers (%)	CServers (%)
16KB	16.3	83.7
4096KB	100.0	0.0

- Most of the random small requests are redirected to CServers, because CServer has good behaviors for them
- Nearly all large requests are absorbed by the original DServers, because DServers have higher I/O parallelism and better aggregated I/O throughput

ILLINOIS INSTITUTE OF TECHNOLOGY

Varying Number of Processes





- S4D-Cache improves the overall I/O bandwidth for various processes
- With the number of processes increasing, bandwidth gets lower because each DServer needs to serve more processes' requests and the competition among processes gets more severe.
- Good scalability in terms of the number of processes.

ILLINOIS INSTITUTE OF TECHNOLOGY

Varying SSD Cache Capacities

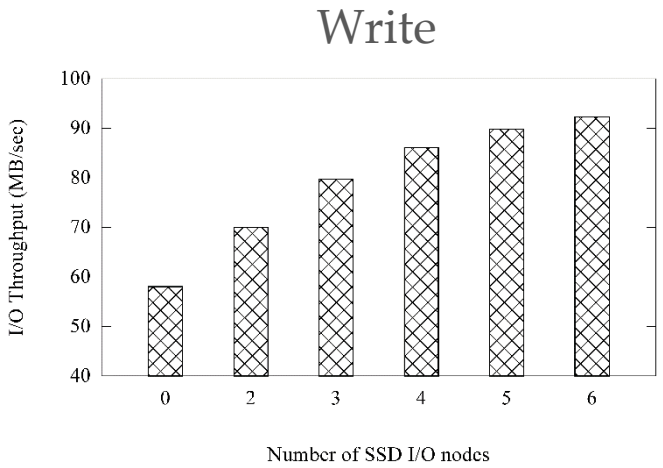
TABLE IV
I/O THROUGHPUTS OF IOR WITH VARIED SSD CACHE CAPACITIES.

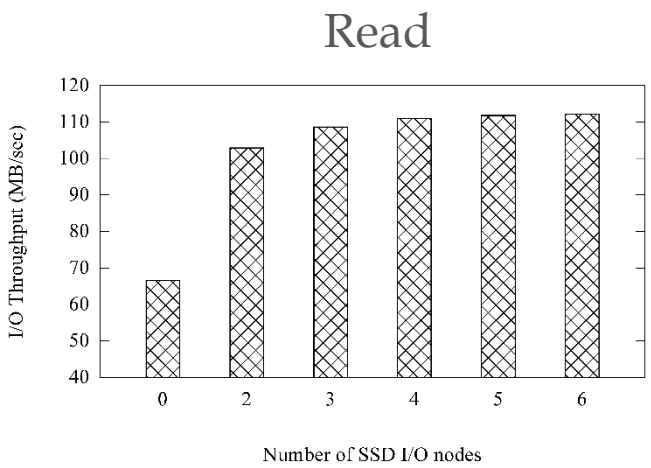
SSD Capacity	Throughput (MB/s)	Speedup (%)
0GB	58.03	0
2GB	69.34	19.5
4GB	86.15	48.4
6GB	90.89	56.6

- Throughput improves by increasing the capacity of CServers
- More random I/O requests can benefit from CServers.
- Continuously enlarging CServers will only bring limited performance improvement.

ILLINOIS INSTITUTE OF TECHNOLOGY

Varying Numbers of SSD File Servers



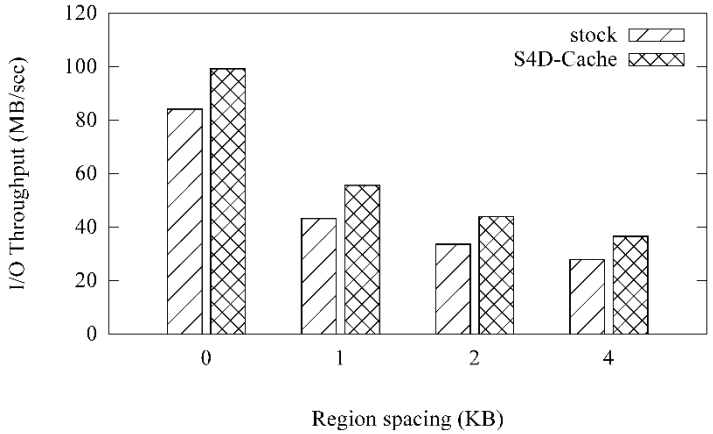


- With the number of CServers increasing, the I/O bandwidth improves because CServers provides better random performance.
- The I/O performance only slightly improves when the number of file servers is above four, because only a portion of the I/O workload is random and the improvement is bounded to these requests.
- Choosing a reasonable number of file servers is critical to make full use of the SSDs.

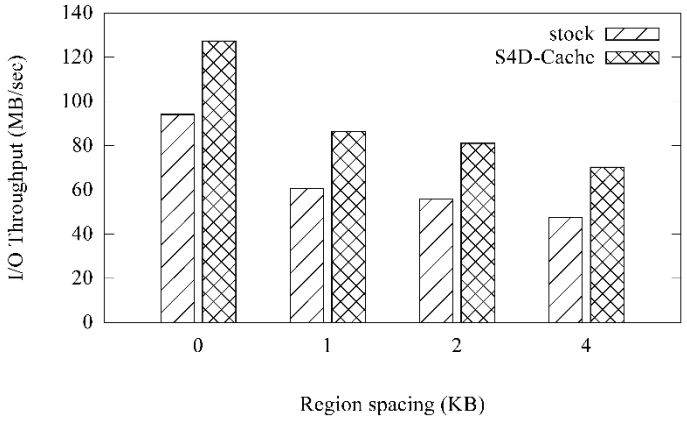
HPIO

ILLINOIS INSTITUTE OF TECHNOLOGY

Write



Read

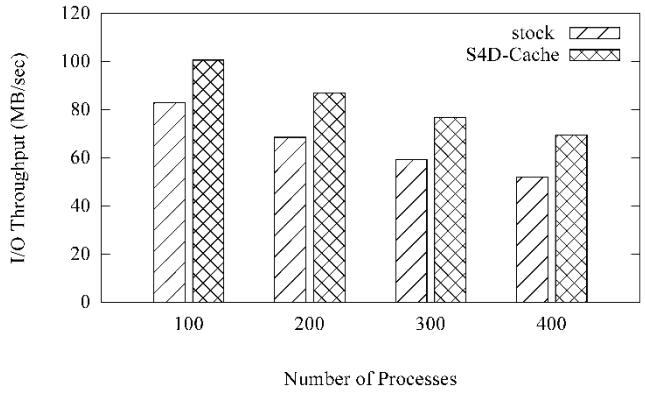


- S4D-Cache is effective with respect to HPIO benchmark
- Improvements for HPIO are not as significant as those for IOR because the workload is not as random as IOR.

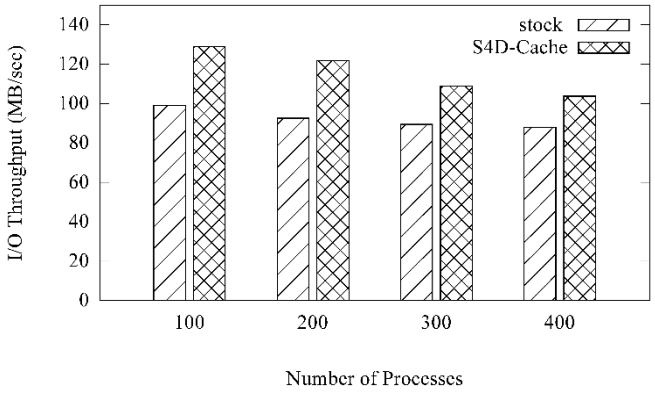
MPI-Tile-IO

ILLINOIS INSTITUTE OF TECHNOLOGY

Write



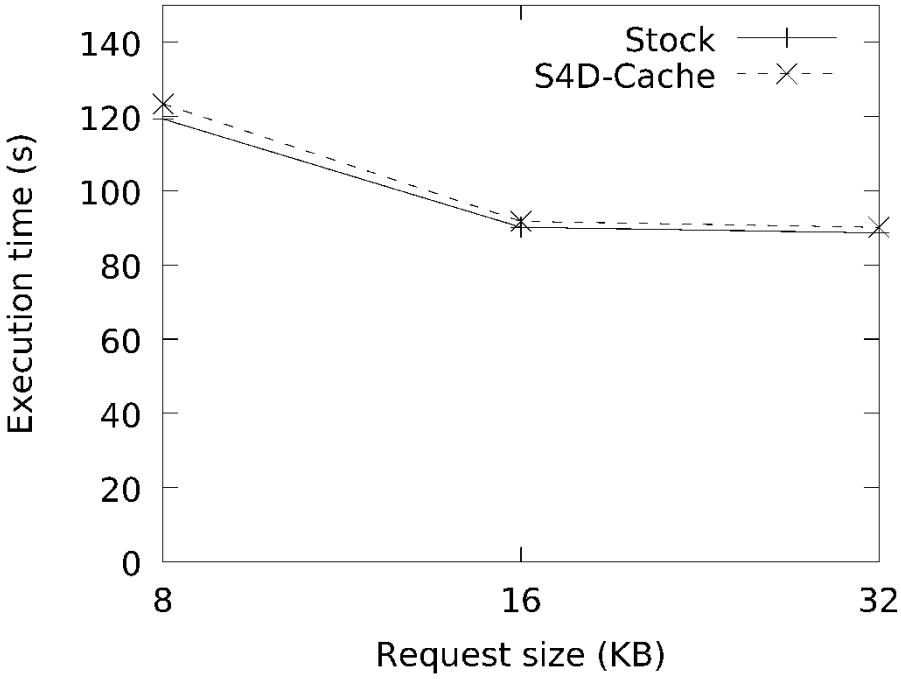
Read



- Performance improvement is still significant

Overhead

- Storage overhead: 0.6% space overhead to keep DMT in storage
- Performance overhead
  - Request cost analysis
  - Critical data identification
  - Cache metadata look-up
- When no requests are cached(Intentionally)
  - Overhead is almost unobservable.



ILLINOIS INSTITUTE
OF TECHNOLOGY

Conclusions

- A centralized SSD cache architecture for parallel I/O systems
- A cost model to evaluate the file request access time on CServers or DServers at the I/O middleware layer
- A selective caching algorithm to caching critical data on the limited SSD space
- The proposed caching system is feasible and effective

Thank you very much!

Shuibing He, Xian-He Sun, Bo Feng

Department of Computer Science

Illinois Institute of Technology