# Workload Model, Job Scheduling, and Resource Allocation

Kuo-Chan Huang
kchuang@mail.ntcu.edu.tw
Department of Computer and Information Science
National Taichung University

26,28,30/07/2010

# Parallel Processing Platforms

- Single-processor computers
  - Vector processor, superscalar, …
- Multi-processor computers
  - SMP, NUMA, …
- Multi-computer systems
  - Cluster, grid, …

# Job Types

- Serial
- Parallel
- Interactive
- batch

# Resource Sharing Modes

- dedicated mode
- space sharing
- time sharing

# Different Performance Requirements

- High performance
- High throughput

# Benefits of Load sharing with Cloud Computing

# Resource Sharing Experiments

- No sharing
- Fully sharing resources (single global job queue)
- Sharing idle resources only

# Environment

| | Cluster 1 | Cluster 2 |
|---|---|---|
| CPU number | 128 | 128 |
| Job number | 5000 | 5000 |
| Workload input | SDSC IBM SP2 | SDSC IBM SP2 |

# No Sharing

| | Cluster 1 | Cluster 2 |
|---|---|---|
| Average waiting time(sec.) | 3862 | 8735.94 |
| Resource utilization | 66.73% | 78.97% |
| System completion time(sec.) | 5222857 | 4633183 |

# Fully Sharing Resources

| | Cluster 1 | Cluster 2 |
|---|---|---|
| Average waiting time(sec.) | 1500.67 | 2051.5 |
| Resource utilization | 69.18% | 69.18% |
| System completion time(sec.) | 5163515 | 5163515 |

# Sharing Idle Resources Only

| | Cluster 1 | Cluster 2 |
|---|---|---|
| Average waiting time(sec.) | 2655.86 | 3140.69 |
| Resource utilization | 68.6% | 69.8% |
| System completion time(sec.) | 5160782 | 5163515 |
| Remote jobs | 607 | 819 |

# How to Protect Short Jobs From Long Waiting Time?

# Equal Partition

| | A whole cluster | | | |
|---|---|---|---|---|
| | Total | Long | Medium | Short |
| Average waiting time(s) | 2.20054e+07 | 1.11542e+07 | 2.53849e+07 | 2.9477e+07 |
| Maximum waiting time(s) | 46830315 | 46320104 | 46830315 | 46824312 |
| System efficiency | 99.6707 | | | |
| Total system time(s) | 63637413 | | | |

| | Partitioned into three spools | | | |
|---|---|---|---|---|
| | Total | Long | Medium | Short |
| Average waiting time(s) | 1.83969e+07 | 3.83782e+07 | 1.67545e+07 | 58060.4 |
| Maximum waiting time(s) | 100500952 | 100500952 | 44024989 | 258799 |
| System efficiency | | 97.2949 | 91.0358 | 81.2617 |
| Total system time(s) | 119547002 | 119547002 | 63636473 | 19736781 |

# Adaptive Partition

| | A whole cluster | | | |
|---|---|---|---|---|
| | Total | Long | Medium | Short |
| Average waiting time(s) | 2.03912e+07 | 2.74461e+07 | 1.4607e+07 | 730347 |
| Maximum waiting time(s) | 57722550 | 57722550 | 57549124 | 18357958 |
| System efficiency | 99.52% | | | |
| Total system time(s) | 77289484 | | | |

| | Partitioned into three spools | | | |
|---|---|---|---|---|
| | Total | Long | Medium | Short |
| Average waiting time(s) | 2.29003e+07 | 3.33406e+07 | 1.34124e+07 | 930.659 |
| Maximum waiting time(s) | 76675196 | 76675196 | 34660733 | 16858 |
| System efficiency | 79.87% | 96.31% | 51.99% | 3.21% |
| Total system time(s) | 96304563 | | | |

# Adaptive Partition with Moldable Property

| | A whole cluster | | | |
|---|---|---|---|---|
| | Total | Long | Medium | Short |
| Average waiting time(s) | 1.27663e+07 | 2.81429e+07 | 9.58302e+06 | 573098 |
| Maximum waiting time(s) | 47929451 | 47929451 | 43425355 | 9778758 |
| System efficiency | 98.62% | | | |
| Total system time(s) | 64314336 | | | |

| | Partitioned into three spools | | | |
|---|---|---|---|---|
| | Total | Long | Medium | Short |
| Average waiting time(s) | 2.25935e+07 | 2.30207e+07 | 2.21317e+07 | 2.26282e+07 |
| Maximum waiting time(s) | 49834919 | 49834919 | 46159089 | 47243432 |
| System efficiency | 91.34% | 93.04% | 90.54% | 91.46% |
| Total system time(s) | 69442636 | | | |

# Moldable Jobs

# Effects of Parallelism Limit

| | Limit=32 with 100% efficiency | Limit=96 with 100% efficiency | Limit=32 with 90% efficiency | Limit=96 with 90% efficiency |
|---|---|---|---|---|
| Average waiting time(sec.) | 6.89629e+06 | 7.17462e+06 | 8.14743e+06 | 1.19398e+07 |
| Maximum waiting time(sec.) | 21652629 | 23674586 | 24806444 | 29343502 |
| System efficiency | 99.58% | 95.08% | 99.57% | 94.03% |
| Total system time(sec.) | 33671055 | 35264343 | 36825171 | 48936949 |

# Reasonable Waiting Time

# Reasonable Waiting Time

- The waiting time encountered by a job is reasonable compared to its execution time.

# Waiting Ratio

- The ratio of a job's waiting time to its execution time is a good indicator to tell if the job's waiting time is reasonable.

# An Example

- Assume job B arrives after job A by 10 minutes and at first both jobs have to wait in a queue because of unavailable resources. Assume jobs A and B request for the same amount of processors, but job A requires much longer execution time than job B, *e.g.* 1000 minutes to 10 minutes. If some resources become available in 10 minutes after job B arrives, but only enough for just one job, which job should be set to run first?

# Different Schedules

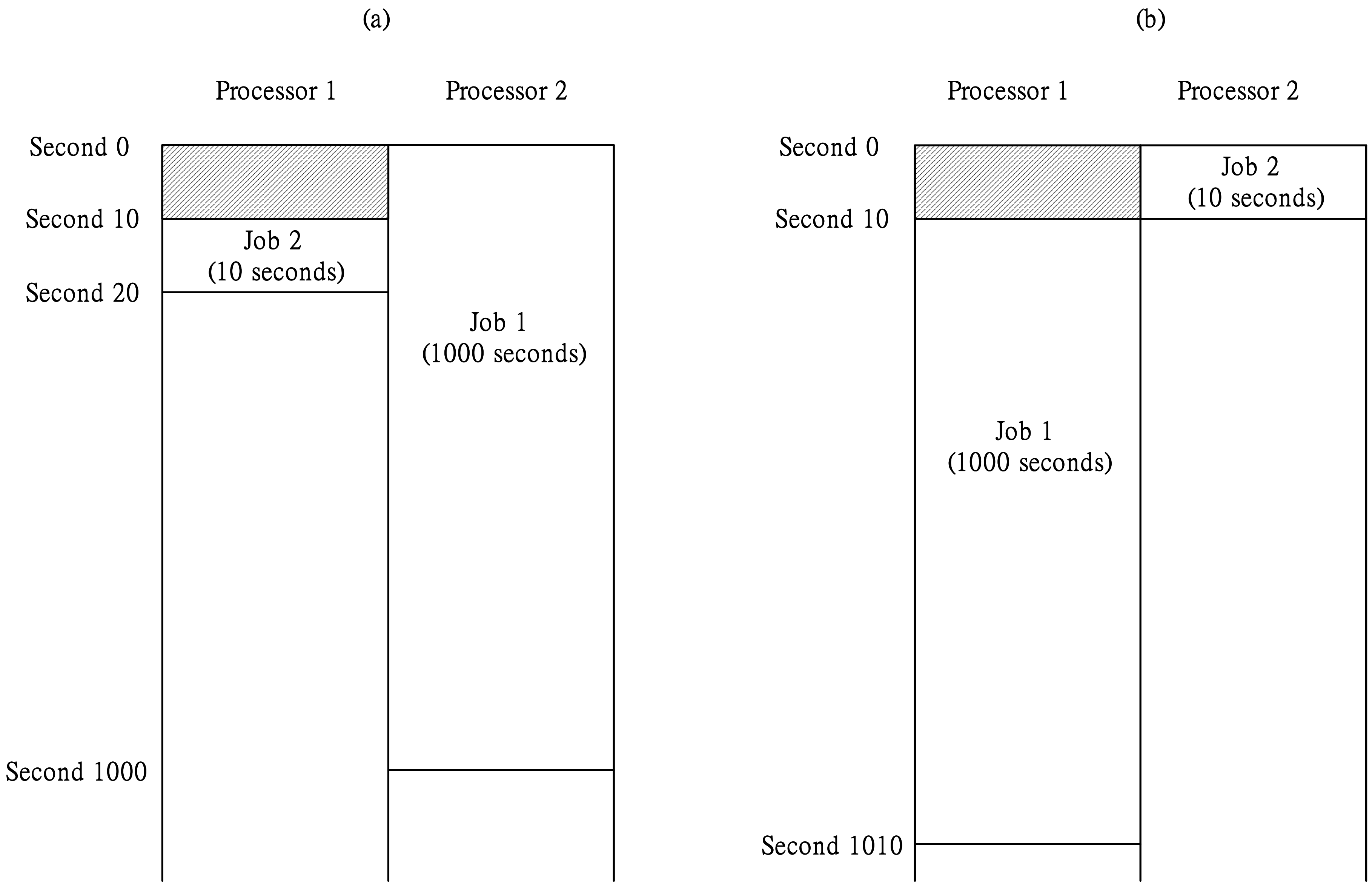
- *First-come first-served* (FCFS)
  - Job A will be executed first, leading to waiting times of 20 and 1010 minutes, waiting ratios of 0.02 and 101 for jobs A and B, respectively.
- Let job B run first
  - Waiting time of 30 and 10 minutes, waiting ratios of 0.03 and 1 for jobs A and B, respectively.

# Single-Processor Computer

- Since jobs have to be processed sequentially, minimum average waiting time implies least average waiting ratio.
- A schedule leading to least average waiting ratio can be found through solving the minimum average waiting time problem.
- It can be solved optimally by the shortest-job-first greedy algorithm.

# Parallel Computer

- The situation is more complicated.
- Minimum average waiting time does not guarantee the least average waiting ratio.



(a)
Processor 1
Processor 2
Second 0
Second 10
Job 2
(10 seconds)
Second 20
Job 1
(1000 seconds)
Second 1000
(b)
Processor 1
Processor 2
Second 0
Job 2
(10 seconds)
Second 10
Job 1
(1000 seconds)
Second 1010

| | Case (a) | | | Case (b) | | |
|---|---|---|---|---|---|---|
| Waiting ratio | Job 1 | Job 2 | Average | Job 1 | Job 2 | Average |
| | 0 | 1 | 0.5 | 0.01 | 0 | 0.005 |

# Different Scheduling and Allocation Policies

- Multi-queue, multi-pool
- Multi-queue, one-pool
  - Scan
  - queue priority
- One-queue, one-pool
  - First-come, first-served (FCFS)
  - Largest-waiting-ratio-first
  - Shortest-job-first

# Characteristics of Workload Log on SDSC's SP2

| | Number of jobs | Maximum execution time (sec.) | Average execution time (sec.) | Maximum number of processors per job | Average number of processors per job |
|---|---|---|---|---|---|
| Group 1 | 4053 | 21922 | 267.13 | 8 | 3 |
| Group 2 | 6795 | 64411 | 6746.27 | 128 | 16 |
| Group 3 | 26067 | 118561 | 5657.81 | 128 | 12 |
| Group 4 | 19398 | 64817 | 5935.92 | 128 | 6 |
| Group 5 | 177 | 42262 | 462.46 | 50 | 4 |
| Total | 56490 | | | | |

# Configuration of processor pools for the simulations

| | Number of processors | | | | |
|---|---|---|---|---|---|
| One-queue methods | 442 | | | | |
| Multi-queue methods | group 1 | group 2 | group 3 | group 4 | group 5 |
| | 8 | 128 | 128 | 128 | 50 |

# Performance results for the 442-node configuration

| | | Average queue length | Average waiting ratio | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | total | group 1 | group 2 | group 3 | group 4 | group 5 |
| Multi-queue | Multi-pool | 4.75 | 21.46 | 3.67 | 14.91 | 37.38 | 6.27 | 0 |
| | Scan | 0.21 | 0.18 | 0.03 | 0.78 | 0.15 | 0.05 | 0 |
| | Queue priority | 0.22 | 0.25 | 0.03 | 1.27 | 0.13 | 0.11 | 0 |
| One-queue | FCFS | 0.24 | 0.37 | 0.67 | 1.09 | 0.22 | 0.26 | 0 |
| | Largest-waiting-ratio-first | 0.21 | 0.09 | 0.15 | 0.23 | 0.08 | 0.03 | 0 |
| | Shortest-job-first | 0.19 | 0.04 | 0.03 | 0.11 | 0.06 | 0.01 | 0 |

# An example comparing shortest-job-first and largest-waiting-ratio-first

- Assume job 1 arrives at second zero and job 2 at second 10, requiring 1000 and 10 seconds for execution, respectively. Further, assume both request the same amount of processors. If at second zero the resources are not available and at second 10 the resources become available but only enough for one job.

| | Arrival time (sec.) | Execution time (sec.) | Waiting ratios at second 10 | Final waiting ratios resulting from shortest-job-first | Final waiting ratios resulting from largest-waiting-ratio-first |
|---|---|---|---|---|---|
| Job 1 | 0 | 1000 | 0.01 | 0.02 | 0.01 |
| Job 2 | 10 | 10 | 0 | 0 | 100 |
| Average | | | | 0.01 | 50.005 |

- One thing to be noted is that the shortest-job-first method has a chance of suffering from the starvation problem.

# Evaluation of Non-FCFS Policies for Variable Partitioning Based Job Scheduling

# Background

- Variable Partitioning Based Job Scheduling
- FCFS
  - fragmentation
- Backfilling
  - Estimation of execution time

# Non-FCFS Policies

- Backfilling
  - Conservative
  - Aggressive (EASY)
- First available
- Smallest first
- Largest first

# Simulation Configuration

## Characteristics of the workload log on SDSC's SP2

| | Number of jobs | Maximum execution time (sec.) | Average execution time (sec.) | Maximum number of processors per job | Average number of processors per job |
|---|---|---|---|---|---|
| Queue 1 | 4053 | 21922 | 267.13 | 8 | 3 |
| Queue 2 | 6795 | 64411 | 6746.27 | 128 | 16 |
| Queue 3 | 26067 | 118561 | 5657.81 | 128 | 12 |
| Queue 4 | 19398 | 64817 | 5935.92 | 128 | 6 |
| Queue 5 | 177 | 42262 | 462.46 | 50 | 4 |
| Total | 56490 | | | | |

# Average queue lengths of workloads with different load factor values

| | FCFS | First Available | Smallest First | Largest First | Backfilling |
|---|---|---|---|---|---|
| Load Factor=1.0 | 0.41 | 0.33 | 0.33 | 0.35 | 0.35 |
| 1.5 | 0.73 | 0.54 | 0.52 | 0.55 | 0.57 |
| 2.0 | 1.35 | 0.85 | 0.80 | 0.91 | 0.89 |
| 2.5 | 3.41 | 1.64 | 1.52 | 1.75 | 1.92 |
| 3.0 | 9.46 | 3.30 | 2.71 | 3.96 | 4.69 |
| 4.0 | 208.33 | 37.59 | 30.14 | 71.55 | 54.26 |

# Experimental Results

**Average waiting time for different scheduling policies**

| | FCFS | First Available | Smallest First | Largest First | Backfilling |
|---|---|---|---|---|---|
| Load Factor=1.0 | 102.71 | 49.17 | 47.40 | 51.86 | 50.73 |
| 1.5 | 335.20 | 142.49 | 134.14 | 150.79 | 157.63 |
| 2.0 | 956.25 | 370.18 | 343.39 | 417.00 | 396.85 |
| 2.5 | 2815.84 | 938.98 | 833.67 | 1075.84 | 1082.79 |
| 3.0 | 8328.50 | 2280.82 | 1909.66 | 3059.78 | 2863.94 |
| 4.0 | 350945.34 | 51191.40 | 49653.18 | 95794.07 | 65481.10 |

**Average waiting ratio for different scheduling policies**

| | FCFS | First Available | Smallest First | Largest First | Backfilling |
|---|---|---|---|---|---|
| Load Factor=1.0 | 0.77 | 0.17 | 0.15 | 0.21 | 0.16 |
| 1.5 | 1.89 | 0.51 | 0.40 | 0.53 | 0.53 |
| 2.0 | 4.08 | 0.86 | 0.79 | 1.15 | 0.85 |
| 2.5 | 10.38 | 1.68 | 1.29 | 2.03 | 1.75 |
| 3.0 | 26.92 | 3.15 | 2.19 | 5.27 | 3.22 |
| 4.0 | 840.93 | 62.63 | 21.99 | 153.65 | 50.72 |

## Max waiting time for different scheduling policies

| | FCFS | First Available | Smallest First | Largest First | Backfilling |
|---|---|---|---|---|---|
| Load Factor=1.0 | 34954 | 38814 | 38814 | 38814 | 34954 |
| 1.5 | 57605 | 61864 | 71250 | 61864 | 57520 |
| 2.0 | 80424 | 124123 | 174054 | 98256 | 80420 |
| 2.5 | 130643 | 478048 | 496785 | 427821 | 119601 |
| 3.0 | 211529 | 753552 | 959295 | 886564 | 191421 |
| 4.0 | 1955908 | 8876911 | 28375296 | 19951289 | 1103174 |

**Max waiting ratio for different scheduling policies**

| | FCFS | First Available | Smallest First | Largest First | Backfilling |
|---|---|---|---|---|---|
| Load Factor=1.0 | 1377.84 | 560.22 | 272.17 | 560.22 | 249.19 |
| 1.5 | 1544.51 | 495.29 | 489.43 | 495.29 | 447.10 |
| 2.0 | 1640.79 | 589.54 | 589.54 | 589.54 | 541.85 |
| 2.5 | 1702.02 | 658.58 | 625.37 | 1006.77 | 602.65 |
| 3.0 | 1855.74 | 838.80 | 1780.38 | 2120.17 | 639.40 |
| 4.0 | 21015.27 | 10827.62 | 59653.56 | 19426.28 | 4426.99 |

Standard deviation of waiting time for different scheduling policies

| | FCFS | First Available | Smallest First | Largest First | Backfilling |
|---|---|---|---|---|---|
| **Load Factor=1.0** | **1170.82** | **794.97** | **789.76** | **818.80** | **792.75** |
| **1.5** | **2678.54** | **1573.64** | **1581.04** | **1651.43** | **1663.42** |
| **2.0** | **5454.09** | **3253.52** | **3130.04** | **3407.55** | **3097.36** |
| **2.5** | **10918.70** | **6206.17** | **6476.86** | **7011.77** | **5858.50** |
| **3.0** | **22453.32** | **12702.28** | **15343.95** | **16536.30** | **11125.66** |
| **4.0** | **615628.45** | **227038.52** | **714550.88** | **412196.85** | **182894.95** |

**Standard deviation of waiting ratio for different scheduling policies**

| | FCFS | First Available | Smallest First | Largest First | Backfilling |
|---|---|---|---|---|---|
| **Load Factor=1.0** | **15.80** | **3.87** | **3.03** | **4.93** | **2.91** |
| **1.5** | **24.03** | **7.18** | **6.47** | **7.79** | **7.08** |
| **2.0** | **35.83** | **11.03** | **10.41** | **14.06** | **10.14** |
| **2.5** | **63.02** | **15.78** | **13.25** | **19.86** | **15.19** |
| **3.0** | **118.83** | **23.28** | **22.82** | **42.61** | **21.24** |
| **4.0** | **2570.26** | **353.87** | **523.54** | **869.26** | **289.64** |

## Detailed performance change comparisons of waiting times and ratios for non-FCFS policies

| | | Load Factor=1.0 | 1.5 | 2.0 | 2.5 | 3.0 | 4.0 |
|---|---|---|---|---|---|---|---|
| **First Available** | **less** | **681** | **1332** | **2928** | **6221** | **12499** | **29037** |
| | **More** | **100** | **180** | **465** | **730** | **1085** | **1601** |
| | **equal** | **55709** | **54978** | **53097** | **49539** | **42906** | **25852** |
| | **Equal(nonzero)** | **803** | **1089** | **1413** | **1628** | **1907** | **963** |
| **Smallest First** | **less** | **845** | **1622** | **3437** | **6762** | **13131** | **29318** |
| | **More** | **163** | **276** | **585** | **1001** | **1546** | **1926** |
| | **equal** | **55482** | **54592** | **52468** | **48727** | **41813** | **25246** |
| | **Equal(nonzero)** | **576** | **703** | **784** | **816** | **814** | **357** |
| **Largest First** | **less** | **693** | **1367** | **2943** | **6267** | **12595** | **28003** |
| | **More** | **236** | **354** | **751** | **1183** | **1711** | **3011** |
| | **equal** | **55561** | **54769** | **52796** | **49040** | **42184** | **25476** |
| | **Equal(nonzero)** | **655** | **880** | **1112** | **1129** | **1185** | **587** |
| **Backfilling** | **less** | **631** | **1254** | **2781** | **6165** | **11984** | **29516** |
| | **More** | **0** | **0** | **0** | **0** | **0** | **0** |
| | **equal** | **55859** | **55236** | **53709** | **50325** | **44506** | **26974** |
| | **Equal(nonzero)** | **953** | **1347** | **2025** | **2414** | **3507** | **2085** |

# Discussion

- Non-FCFS methods can effectively improve the overall system utilization and performance. The simulation results indicate that the *smallest first* non-FCFS policy can reduce the waiting time to one-eighth and the waiting ratio to one-fortieth of the original values for the FCFS policy.
- As the worst case is concerned, the *backfilling* policy is superior .
- Setting threshold value may be able to improve the performance of the worst case for the non-FCFS policies.

# Multi-cluster Computing Environment

# Cross-Site Parallel Computation

- Slowdown ratio = $\frac{ExecutionTimeAcrossSiteBoundaries}{ExecutionTimeWithinSingleSite}$
- Reducing the frequency of cross-site parallel computation could improve system performance.
- Both kinds of allocation methods for single-site and cross-site parallel jobs could influence the frequency.

# Processor Allocation Methods for Reducing Cross-Site Parallel Computation

- Allocating single-site parallel jobs
  - First fit
  - Best fit
  - Worst fit
  - Median fit
  - Random fit
- Allocating cross-site parallel jobs
  - Fixed Order
  - Larger first
  - Smaller first

# Configuration of Multi-cluster Environment

- The processors on all clusters run at the same speed.

| | total | cluster 1 | cluster 2 | cluster 3 | cluster 4 | cluster 5 |
|---|---|---|---|---|---|---|
| Number of processors | 442 | 8 | 128 | 128 | 128 | 50 |

Average waiting time for different slowdown ratios (sec.)

| Slowdown ratio | First Fit | Median Fit | Random Fit | Best Fit | Worst Fit |
|---|---|---|---|---|---|
| 1 | 50.36 | 50.36 | 50.36 | 50.36 | 50.36 |
| 2 | 61.71 | 65.08 | 80.10 | 60.48 | 93.93 |
| 4 | 117.57 | 166.13 | 92.48 | 64.05 | 530.58 |
| 5 | 200.71 | 959.88 | 133.83 | 99.44 | 2219.90 |

Average waiting ratio for different slowdown ratios

| Slowdown ratio | First Fit | Median Fit | Random Fit | Best Fit | Worst Fit |
|---|---|---|---|---|---|
| 1 | 0.37 | 0.37 | 0.37 | 0.37 | 0.37 |
| 2 | 0.47 | 0.50 | 0.65 | 0.47 | 0.76 |
| 4 | 0.90 | 1.28 | 0.86 | 0.51 | 5.06 |
| 5 | 1.88 | 10.11 | 1.14 | 0.75 | 23.11 |

**Average waiting time for different heuristic methods (sec.)**

| Slowdown ratio | Best Fit with Fixed Order | Best Fit with Smaller First | Best Fit with Larger First |
|---|---|---|---|
| 1 | 50.36 | 50.36 | 50.36 |
| 2 | 60.48 | 60.13 | 60.23 |
| 4 | 64.05 | 64.07 | 63.71 |
| 5 | 99.44 | 176.57 | 66.12 |

**Average waiting ratio for different heuristic methods**

| Slowdown ratio | Best Fit with Fixed Order | Best Fit with Smaller First | Best Fit with Larger First |
|---|---|---|---|
| 1 | 0.37 | 0.37 | 0.37 |
| 2 | 0.47 | 0.47 | 0.47 |
| 4 | 0.51 | 0.51 | 0.51 |
| 5 | 0.75 | 1.61 | 0.53 |

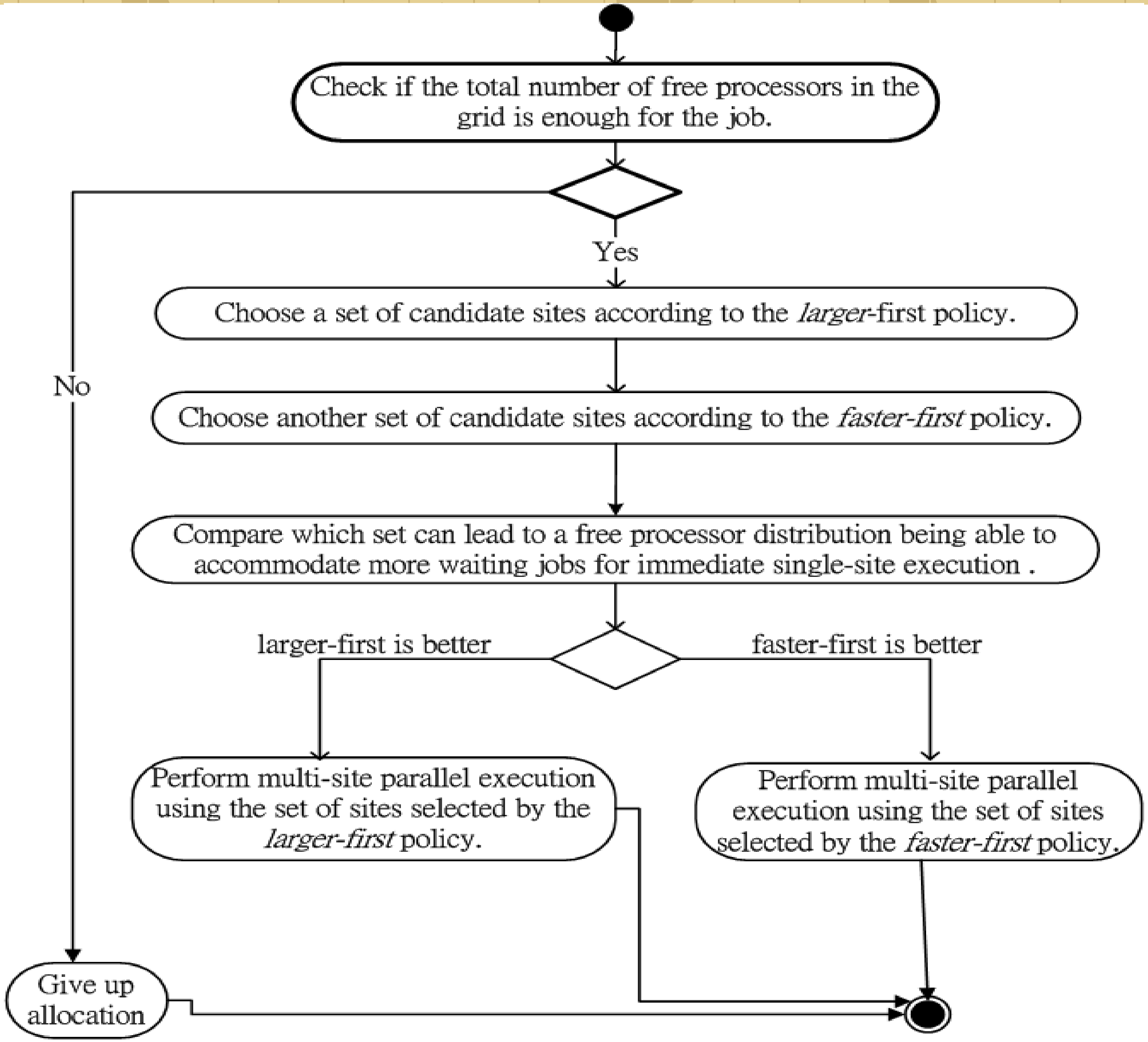
# An Integrated Approach

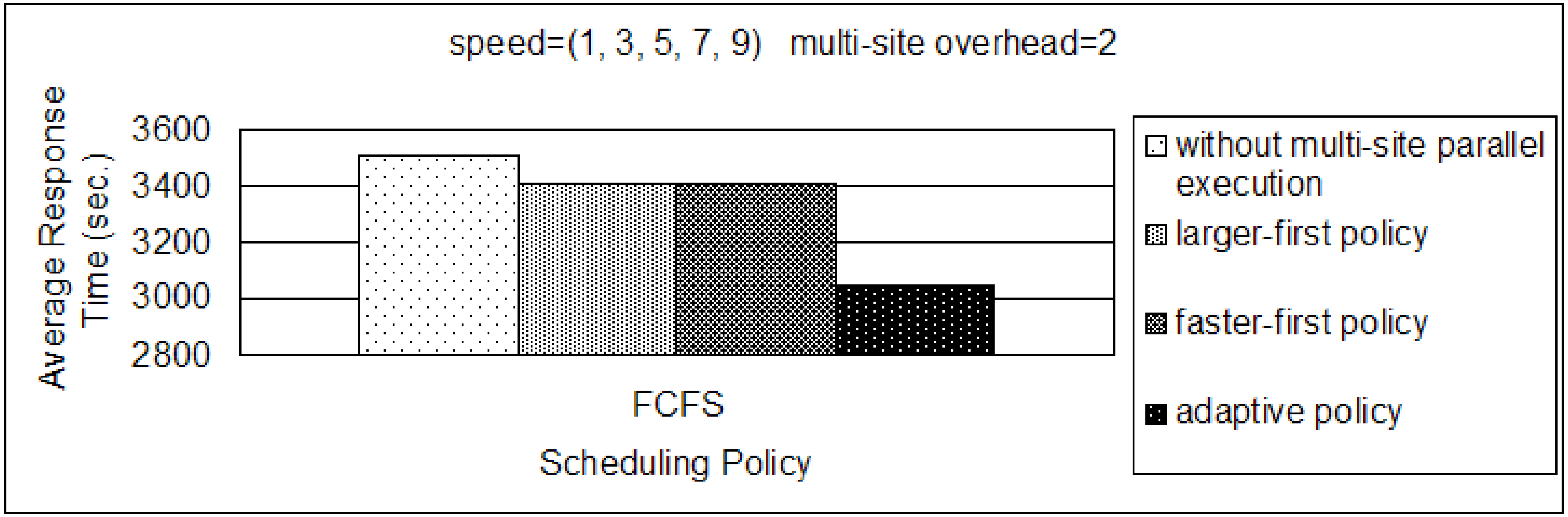
- The previous single-pool centralized queue method.
  - FCFS for job scheduling without special processor allocation methods for reducing cross-site parallel computation.
- The proposed integrated approach
  - Smallest first policy for job scheduling and the best-fit with larger first policy for processor allocation.

| Slowdown ratio | Average waiting time (sec.) | | Average waiting ratio | |
|---|---|---|---|---|
| | Single-pool centralized queue | The integrated approach | Single-pool centralized queue | The integrated approach |
| 5 | 200.71 | 28.54 | 1.88 | 0.08 |
| 4 | 117.57 | 23.68 | 0.90 | 0.07 |
| 2 | 61.71 | 23.30 | 0.47 | 0.07 |
| no slowdown | 50.36 | 22.18 | 0.37 | 0.06 |

# Adaptive Policy in Heterogeneous Multi-cluster Environment

- Single-cluster allocation
  - dynamically changes between the *best-fit* and the *fastest-one* policies
- Multi-cluster allocation


Check if the total number of free processors in the grid is enough for the job.
Yes
No
Choose a set of candidate sites according to the *larger*-first policy.
Choose another set of candidate sites according to the *faster-first* policy.
Compare which set can lead to a free processor distribution being able to accommodate more waiting jobs for immediate single-site execution .
larger-first is better
faster-first is better
Perform multi-site parallel execution using the set of sites selected by the *larger-first* policy.
Perform multi-site parallel execution using the set of sites selected by the *faster-first* policy.
Give up allocation



speed=(1, 3, 5, 7, 9) multi-site overhead=2
Average Response Time (sec.)
3600
3400
3200
3000
2800
FCFS
Scheduling Policy
without multi-site parallel execution
larger-first policy
faster-first policy
adaptive policy

# Searching for Better Load Sharing Methods in Multi-Cluster Environment

# Load Sharing Policies

- Independent clusters
- Forwarding to no-need-to-wait site
- Forwarding to shortest-queue site
- Forwarding to least-load site $\frac{\sum_i (Job(i).runtime \times Job(i).parallelism)}{Number\_of\_processors\_in\_cluster}$
- Multi-pool centralized queue
- Single-pool centralized queue
  - Slowdown ratio $\frac{ExecutionTimeAcrossSiteBoundaries}{ExecutionTimeWithinSingleSite}$
- One big cluster

# Two-Level Scheduling

- Empty queue only
- Shortest queue first
- Least load first
- Forwarding to shortest-queue site with two local queues

# Characteristics of Workload Log on SDSC's SP2

| | Number of jobs | Maximum execution time (sec.) | Average execution time (sec.) | Maximum number of processors per job | Average number of processors per job |
|---|---|---|---|---|---|
| Queue 1 | 4053 | 21922 | 267.13 | 8 | 3 |
| Queue 2 | 6795 | 64411 | 6746.27 | 128 | 16 |
| Queue 3 | 26067 | 118561 | 5657.81 | 128 | 12 |
| Queue 4 | 19398 | 64817 | 5935.92 | 128 | 6 |
| Queue 5 | 177 | 42262 | 462.46 | 50 | 4 |
| Total | 56490 | | | | |

# Configuration of the Computing Grid

| | Total | Site 1 | Site 2 | site3 | Site 4 | Site 5 |
|---|---|---|---|---|---|---|
| Number of processors | 442 | 8 | 128 | 128 | 128 | 50 |

# Performance Evaluation of Load Sharing Policies

| Load sharing methods | Average waiting time(sec.) | Standard deviation | Average waiting ratio | Standard deviation |
|---|---|---|---|---|
| Independent clusters | 2772.63 | 10797.80 | 21.46 | 148.07 |
| Local queue based methods | | | | |
| Forwarding to no-need-to-wait site | 111.08 | 1658.17 | 0.51 | 8.76 |
| Forwarding to shortest-queue site | 91.80 | 1560.22 | 0.41 | 15.59 |
| Forwarding to least-load site | 86.28 | 1477.90 | 0.30 | 9.32 |
| Centralized queue based methods | | | | |
| Multi-pool centralized queue | 127.64 | 1487.69 | 1.03 | 20.53 |
| Single-pool centralized queue (slowdown ratio: 6) | 2184.36 | 17251.00 | 23.84 | 273.75 |
| Single-pool centralized queue (slowdown ratio: 5) | 200.71 | 2845.86 | 1.88 | 37.81 |
| Single-pool centralized queue (slowdown ratio: 4) | 117.57 | 1749.76 | 0.90 | 19.42 |
| Single-pool centralized queue (slowdown ratio: 2) | 61.71 | 946.55 | 0.47 | 13.88 |

| | | | | |
|---|---|---|---|---|
| **Single-pool centralized queue (no slowdown)** | **50.36** | **774.95** | **0.37** | **11.00** |
| **One big cluster** | **50.36** | **774.95** | **0.37** | **11.00** |
| **Two-level scheduling** | | | | |
| **Empty-queue-only multi-pool grid** | **67.86** | **1239.00** | **0.22** | **6.34** |
| **Shortest-queue-first multi-pool grid** | **75.23** | **1361.69** | **0.23** | **5.14** |
| **Least-load-first multi-pool grid** | **73.22** | **1331.80** | **0.28** | **8.50** |
| **Methods with two local queues** | | | | |
| **Forwarding to shortest-queue site** | **94.51** | **1764.42** | **0.34** | **10.47** |
| **Forwarding to shortest-queue site (threshold=max. waiting time)** | **91.40** | **1647.28** | **0.34** | **10.46** |

| Load sharing methods | Maximum waiting time(sec.) | Maximum waiting ratio |
|---|---|---|
| Independent clusters | 144925 | 4420.26 |
| Forwarding to no-need-to-wait site | 63732 | 652.70 |
| Forwarding to shortest-queue site | 86421 | 2059.92 |
| Forwarding to least-load site | 63732 | 1141.42 |
| Multi-pool centralized queue | 34620 | 1329.44 |
| Single-pool centralized queue (slowdown ratio: 6) | 323130 | 9808.96 |
| Single-pool centralized queue (slowdown ratio: 5) | 99864 | 1879.77 |
| Single-pool centralized queue (slowdown ratio: 4) | 78659 | 1336.05 |
| Single-pool centralized queue (slowdown ratio: 2) | 33699 | 1307.00 |
| Single-pool centralized queue (no slowdown) | 30579 | 1204.71 |

| | | |
|---|---|---|
| **One big cluster** | **30579** | **1204.71** |
| **Empty-queue-only multi-pool grid** | **61017** | **887.49** |
| **Shortest-queue-first multi-pool grid** | **63732** | **336.95** |
| **Least-load-first multi-pool grid** | **63732** | **1412.74** |
| **Forwarding to shortest-queue site with two local queues** | **144957** | **1364.53** |
| **Forwarding to shortest-queue site with two local queues (threshold=max. waiting time)** | **105022** | **1364.53** |

# Summary

- Load sharing mechanisms can greatly improve the overall system performance.
- More accurate estimation of workload in each site can improve performance of the local queue based methods.
- Shorter waiting time do not necessarily deliver smaller waiting ratios
- Two-level scheduling methods lead to smaller waiting ratios than the one big cluster

# Performance Evaluation of Adaptive Processor Allocation Policies for Moldable Parallel Batch Jobs

# Partition Specification

- Fixed.
- Variable.
- Adaptive.
- Dynamic.

# Job Flexibility

- Rigid.
- Moldable.
- Evolving.
- Malleable.

# Application Characteristics

- Batch processing.
- Moldable.

# Processor Allocation Policies

- Parallelism limit.
- Adaptive processor allocation
  - No adaptive scaling.
  - Adaptive scaling down.
  - Adaptive scaling up and down.
  - Restricted scaling up and down.

# SDSC's SP2 Workload

| | Number of jobs | Maximum execution time (sec.) | Average execution time (sec.) | Maximum number of processors per job | Average number of processors per job |
|---|---|---|---|---|---|
| Queue 1 | 4053 | 21922 | 267.13 | 8 | 3 |
| Queue 2 | 6795 | 64411 | 6746.27 | 128 | 16 |
| Queue 3 | 26067 | 118561 | 5657.81 | 128 | 12 |
| Queue 4 | 19398 | 64817 | 5935.92 | 128 | 6 |
| Queue 5 | 177 | 42262 | 462.46 | 50 | 4 |
| Total | 56490 | | | | |

# Original Workload

| Parallelism limit | Performance metrics | No adaptive scaling | Adaptive scaling down | Adaptive scaling up and down | Restricted scaling up and down |
|---|---|---|---|---|---|
| 96 | Waiting time | 34489 | 2666 | 11796 | 3034 |
| | Completion time | 39972 | 17716 | 13191 | 17255 |
| 64 | Waiting time | 30751 | 2555 | 11739 | 3091 |
| | Completion time | 36246 | 18058 | 13122 | 16629 |
| 32 | Waiting time | 13849 | 2546 | 9731 | 2939 |
| | Completion time | 19915 | 17823 | 12499 | 15680 |
| 16 | Waiting time | 8857 | 2037 | 7410 | 2578 |
| | Completion time | 16905 | 17044 | 12945 | 15678 |

# Uniform Distribution

| Parallelism limit | Performance metrics | No adaptive scaling | Adaptive scaling down | Adaptive scaling up and down | Restricted scaling up and down |
|---|---|---|---|---|---|
| 96 | Waiting time | 29930 | 2286 | 4584 | 2302 |
| | Completion time | 30501 | 4277 | 5123 | 4088 |
| 64 | Waiting time | 10792 | 2209 | 4638 | 2267 |
| | Completion time | 11477 | 4316 | 5189 | 4097 |
| 32 | Waiting time | 5746 | 2167 | 4222 | 2180 |
| | Completion time | 6914 | 4471 | 5325 | 4306 |
| 16 | Waiting time | 4114 | 2146 | 3683 | 2122 |
| | Completion time | 6352 | 5119 | 5889 | 5053 |

# Normal Distribution

| Parallelism limit | Performance metrics | No adaptive scaling | Adaptive scaling down | Adaptive scaling up and down | Restricted scaling up and down |
|---|---|---|---|---|---|
| 96 | Waiting time | 38358 | 3263 | 4848 | 3481 |
| | Completion time | 38909 | 4843 | 5381 | 4773 |
| 64 | Waiting time | 8409 | 3390 | 4895 | 3291 |
| | Completion time | 9028 | 5029 | 5440 | 4641 |
| 32 | Waiting time | 5024 | 3189 | 4764 | 3212 |
| | Completion time | 6125 | 4882 | 5855 | 4816 |
| 16 | Waiting time | 4420 | 3519 | 4373 | 3520 |
| | Completion time | 6606 | 5977 | 6557 | 5967 |

# Summary

- Evaluation of several adaptive processor allocation policies for moldable parallel batch jobs on space-sharing parallel computers.
- More than eight times of performance improvement is achievable.
- *(Restricted) Adaptive scaling up and down* policy delivers better performance.
- Actual effects of adaptive processor allocation are very complicated and may depend on processor number distributions.

# Adaptive Processor Allocation in Heterogeneous Computational Grid

# The Problem

- How to handle the situation where a parallel job cannot fit in any single site in the grid environment.
  - Multi-site parallel execution
  - Adaptive processor allocation
- Both approaches might incur extended execution time.

# Adaptive Processor Allocation

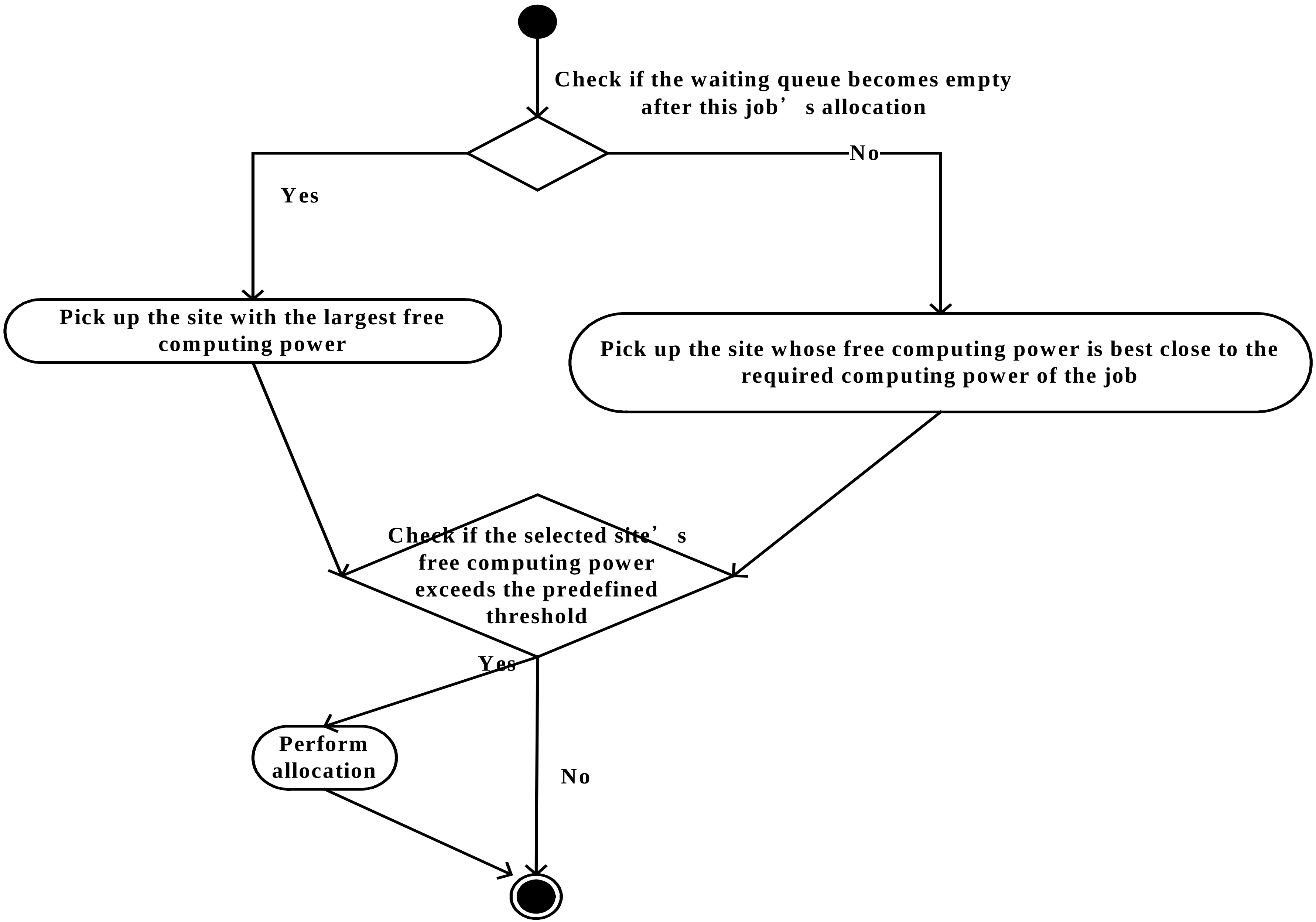
- Space sharing
- Moldable jobs
- Heterogeneous computational grid

# SDSC's SP2 Workload

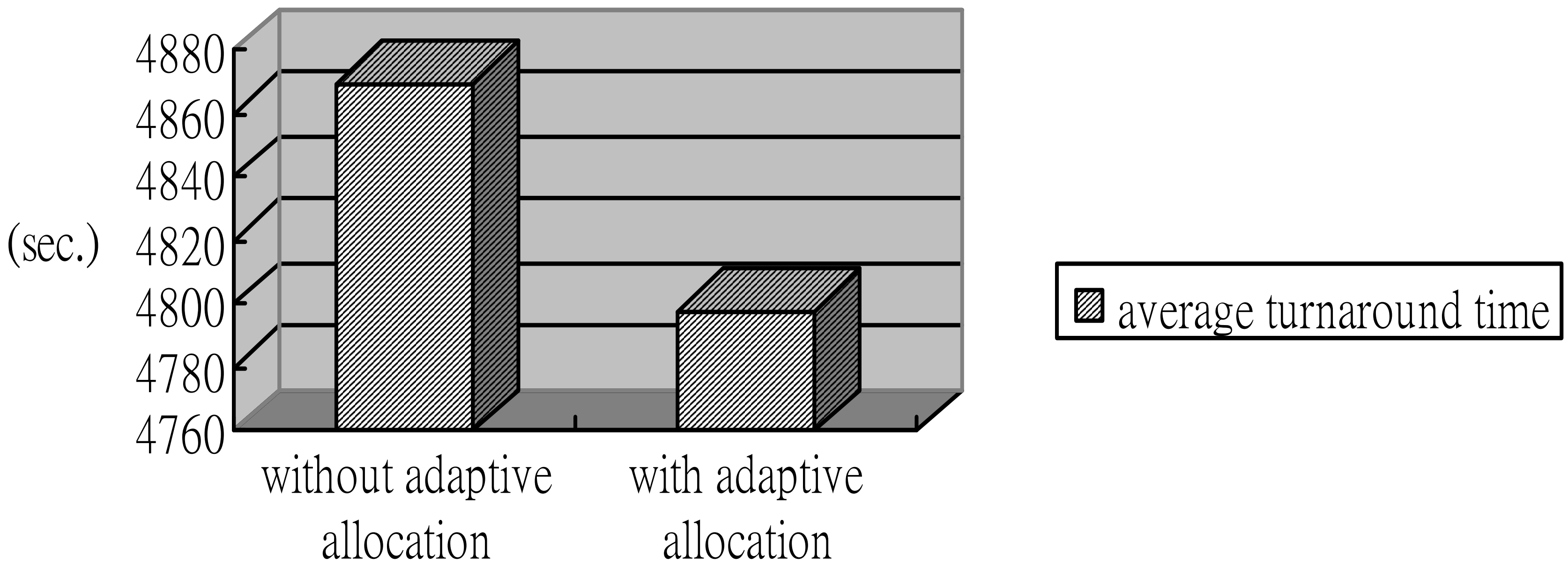
| | Number of jobs | Maximum execution time (sec.) | Average execution time (sec.) | Maximum number of processors per job | Average number of processors per job |
|---|---|---|---|---|---|
| Queue 1 | 4053 | 21922 | 267.13 | 8 | 3 |
| Queue 2 | 6795 | 64411 | 6746.27 | 128 | 16 |
| Queue 3 | 26067 | 118561 | 5657.81 | 128 | 12 |
| Queue 4 | 19398 | 64817 | 5935.92 | 128 | 6 |
| Queue 5 | 177 | 42262 | 462.46 | 50 | 4 |
| Total | 56490 | | | | |

# Configurable Parameters

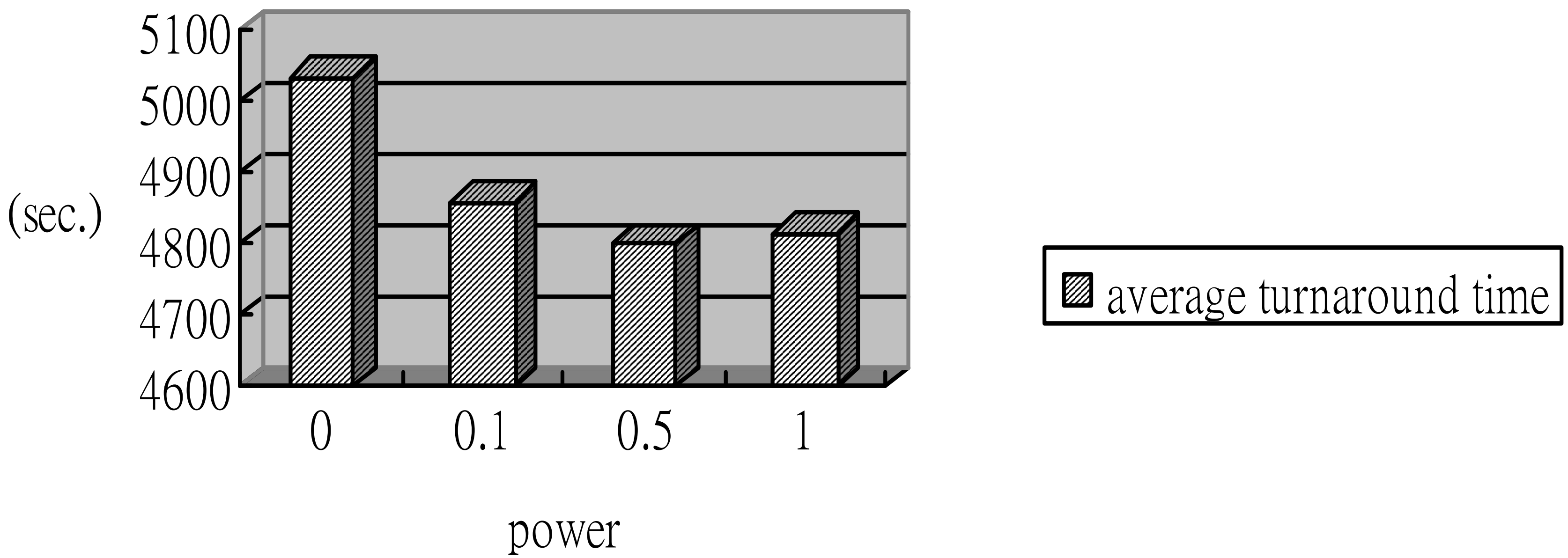
- Speed vector (sp1,sp2,sp3,sp4,sp5)
- Load vector (ld1,ld2,ld3,ld4,ld5)
- Scheduling policy
  - FCFS, SJF
- Single-site allocation policy
  - Best-fit, fastest, adaptive
- Threshold
- power


Check if the waiting queue becomes empty after this job’ s allocation
Yes
No
Pick up the site with the largest free computing power
Pick up the site whose free computing power is best close to the required computing power of the job
Check if the selected site’ s free computing power exceeds the predefined threshold
Yes
No
Perform allocation

speed=(1,3,5,7,9) load=(6.7,6.7,6.7,6.7,6.7) power=0.5

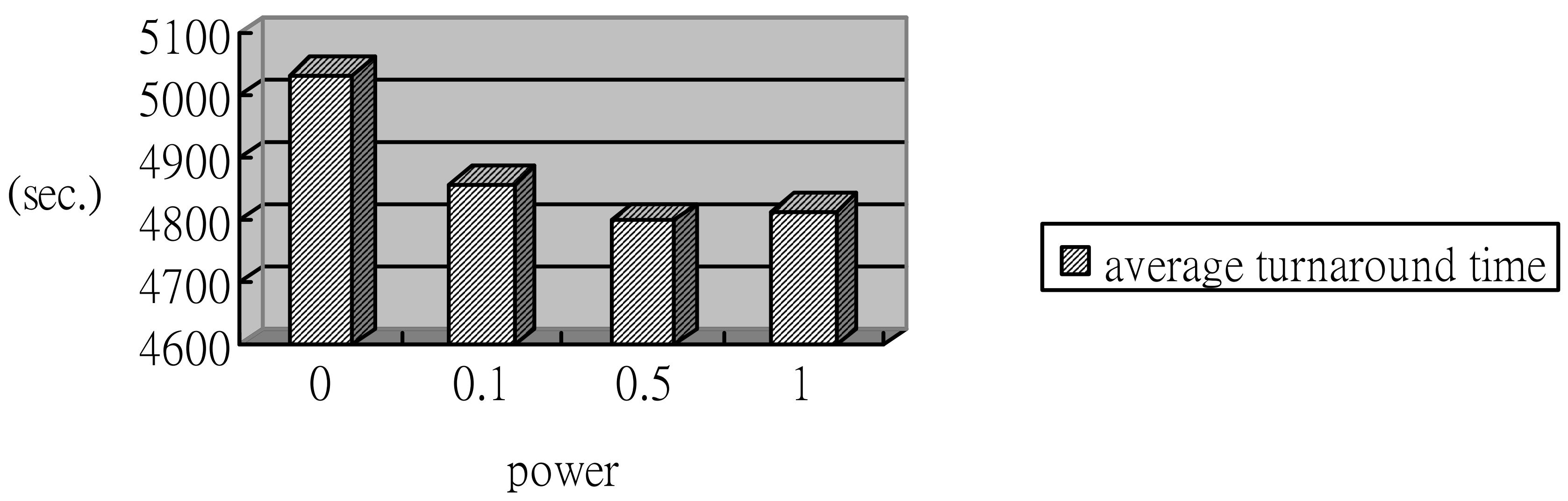


speed=(1,3,5,7,9) load=(6.7,6.7,6.7,6.7,6.7)



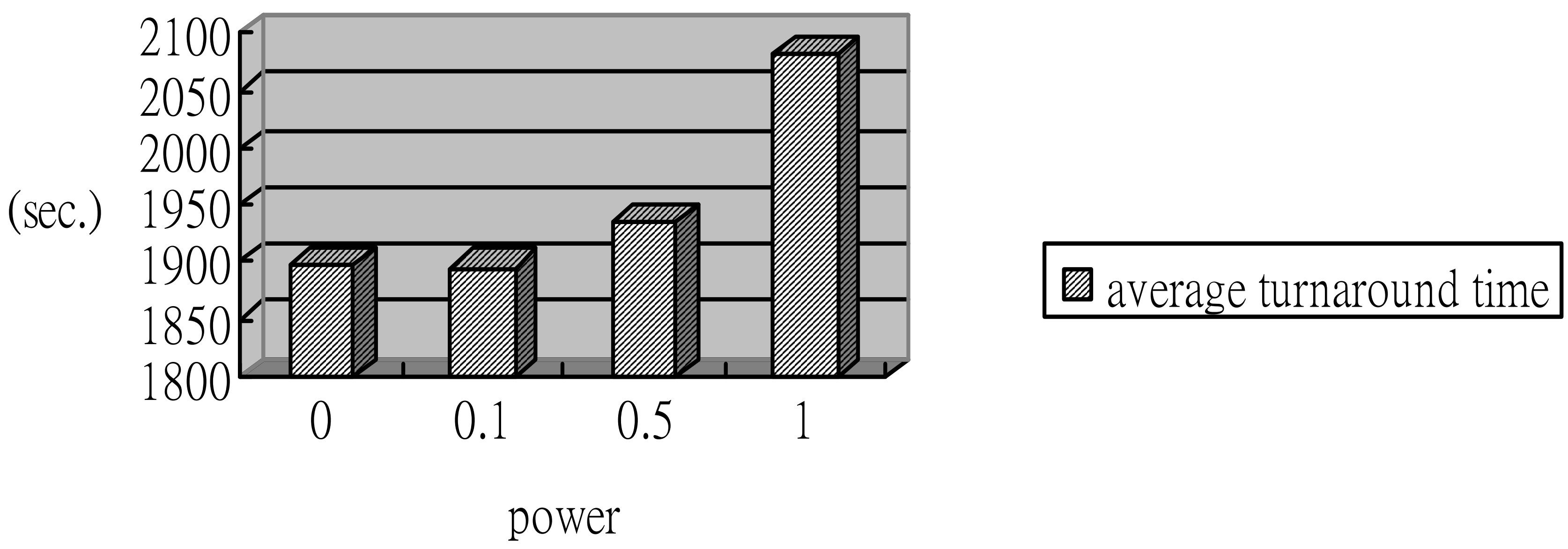
# Adaptive processor allocation with different power values under SDSC's SP2 workload

speed=(1,3,5,7,9) load=(6.7,6.7,6.7,6.7,6.7)

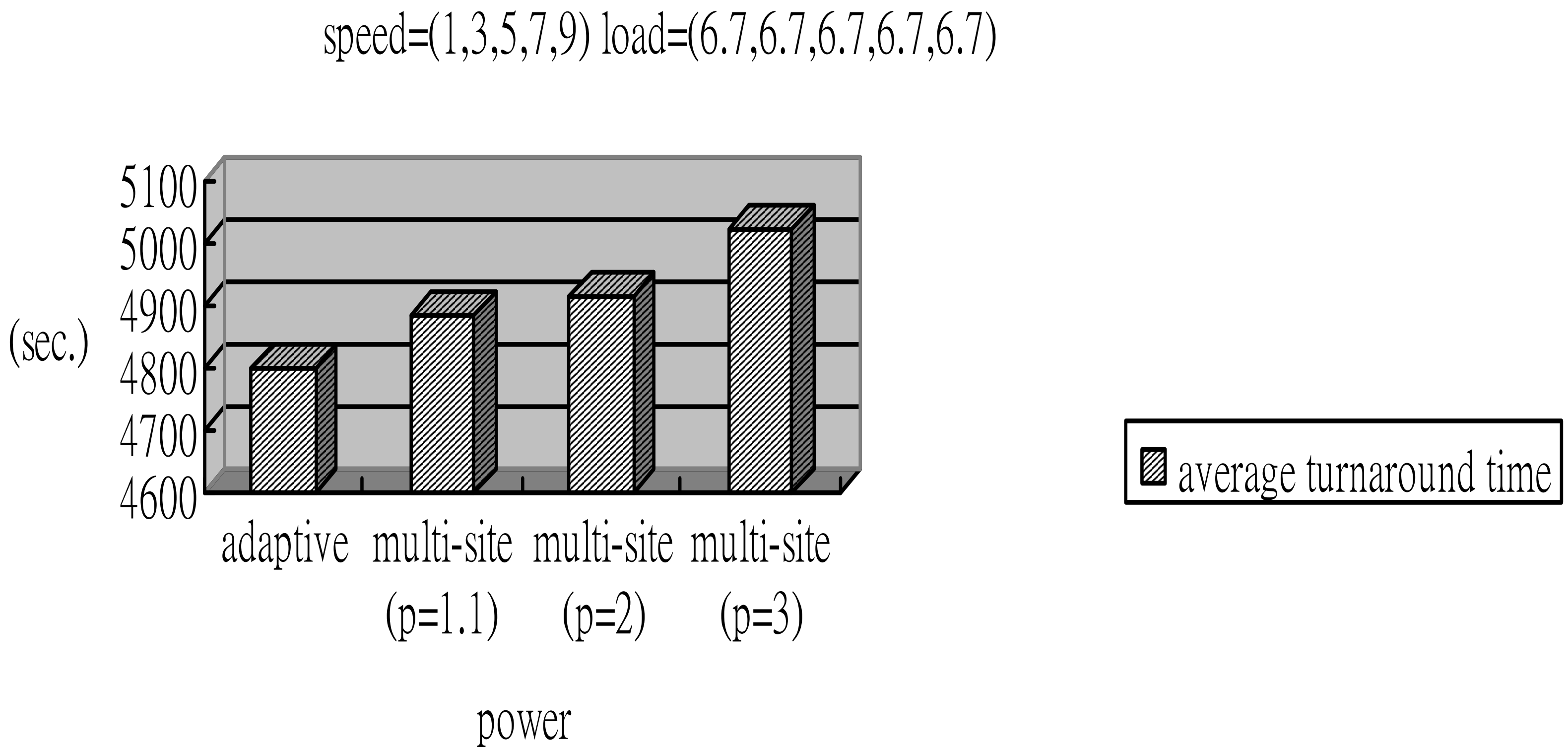


# Adaptive processor allocation with different power values under LANL's CM5 workload

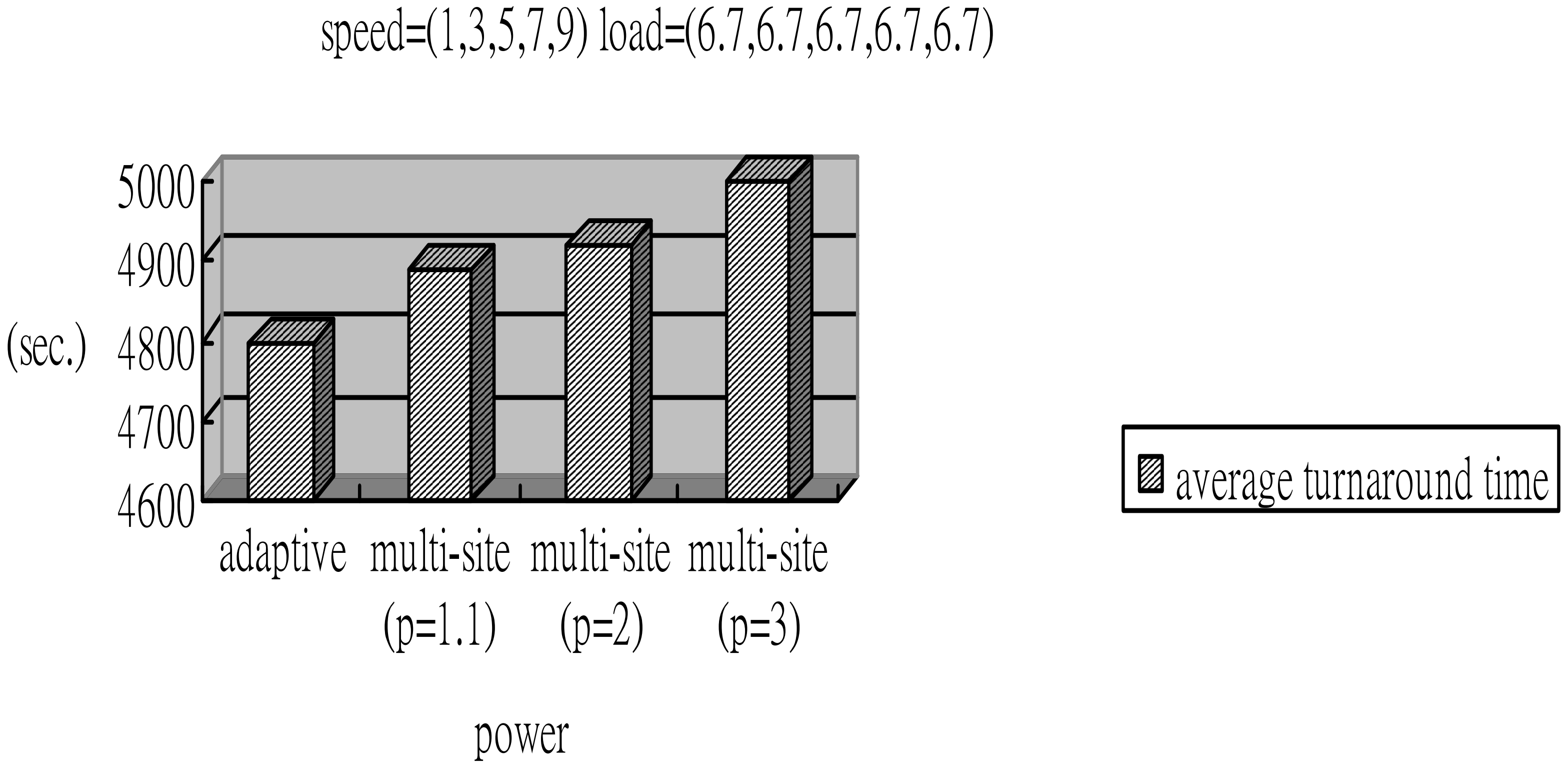
speed=(1,3,5,7) load=(6.7,6.7,6.7,6.7)



# Comparison under SDSC's SP2 workload and uniformly distributed slowdown values

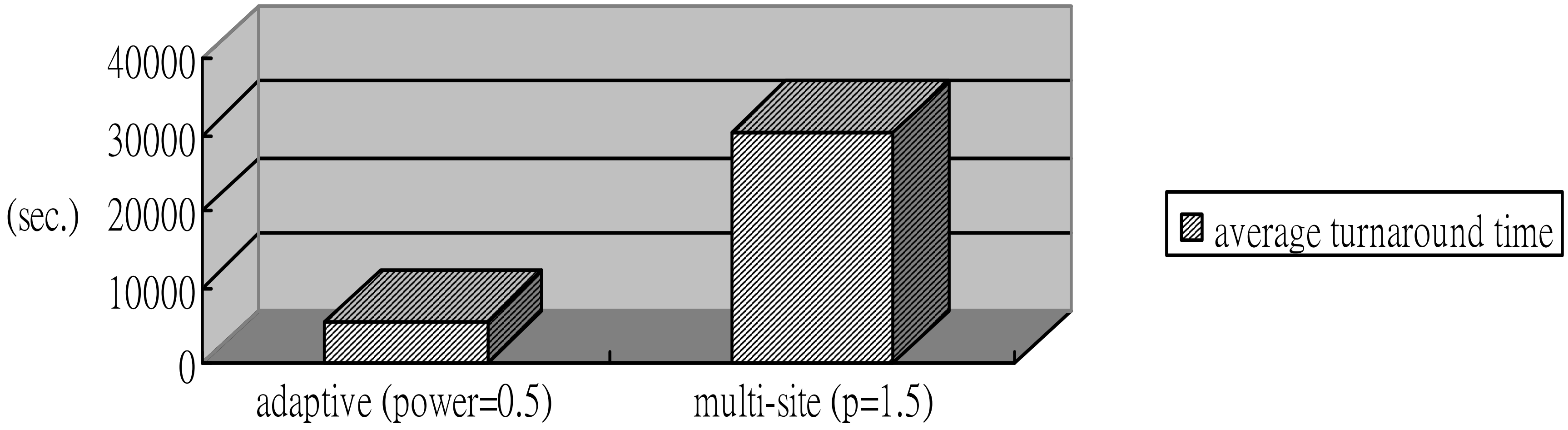


# Comparison under SDSC's SP2 workload and normally distributed slowdown values



# Thorough comparison under SDSC's SP2 workload

average turnaround time for 120 simulation cases corresponding to permutation of speed vector (1,3,5,7,9)

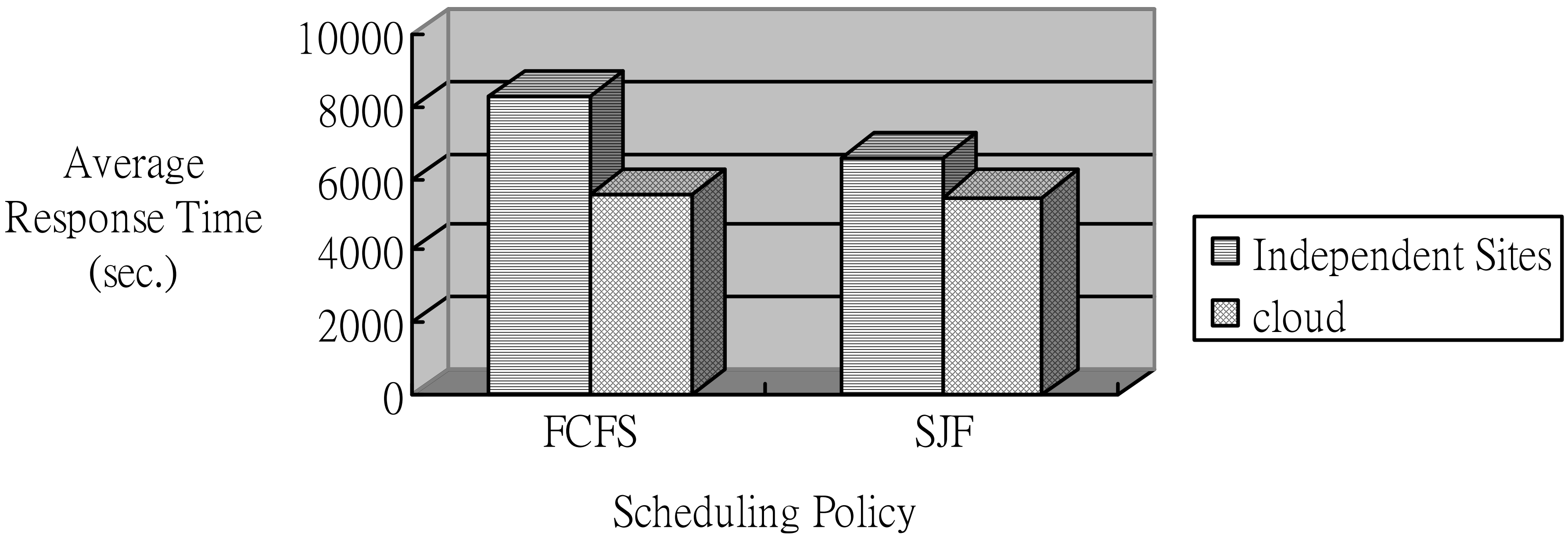


# Summary

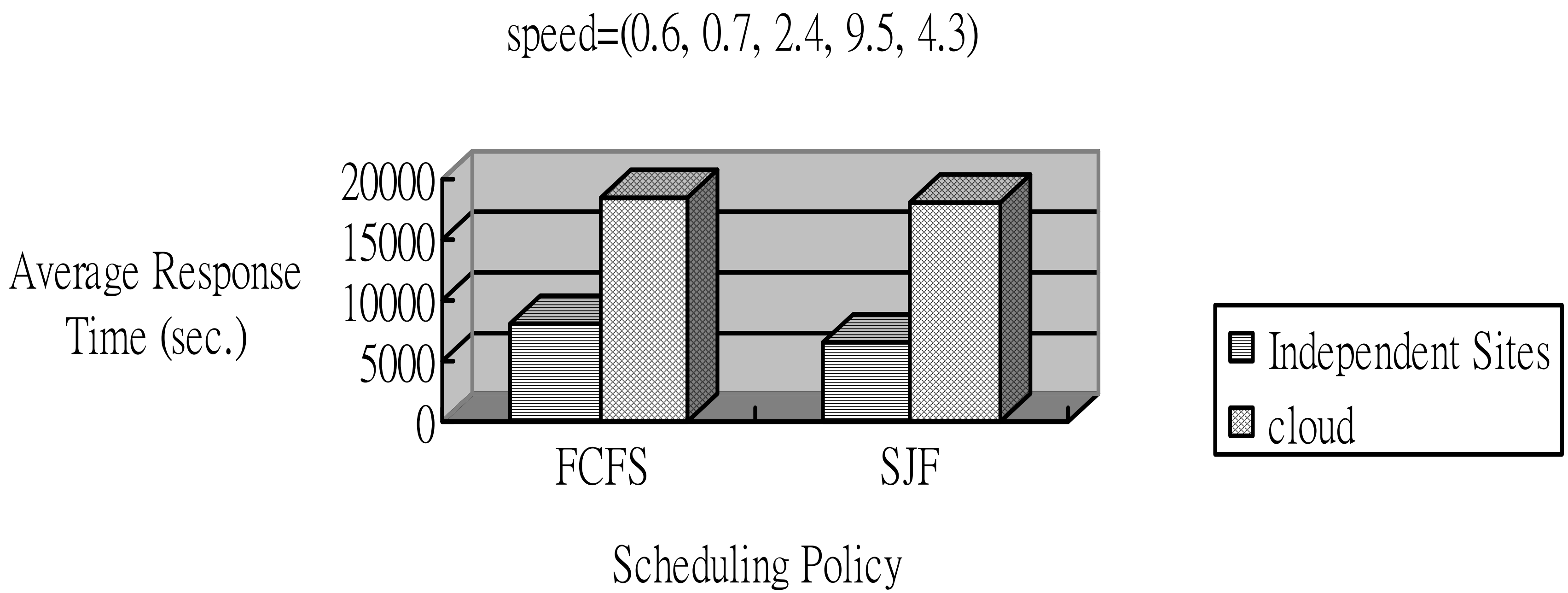
- heterogeneity presents a challenge for effectively arranging load sharing activities in a computational grid.
- adaptive processor allocation is capable of significantly improving the overall system performance in a heterogeneous computational grid environment.

# Job Scheduling in Heterogeneous Cloud

# Load sharing performance in a homogeneous cloud (Best-fit)

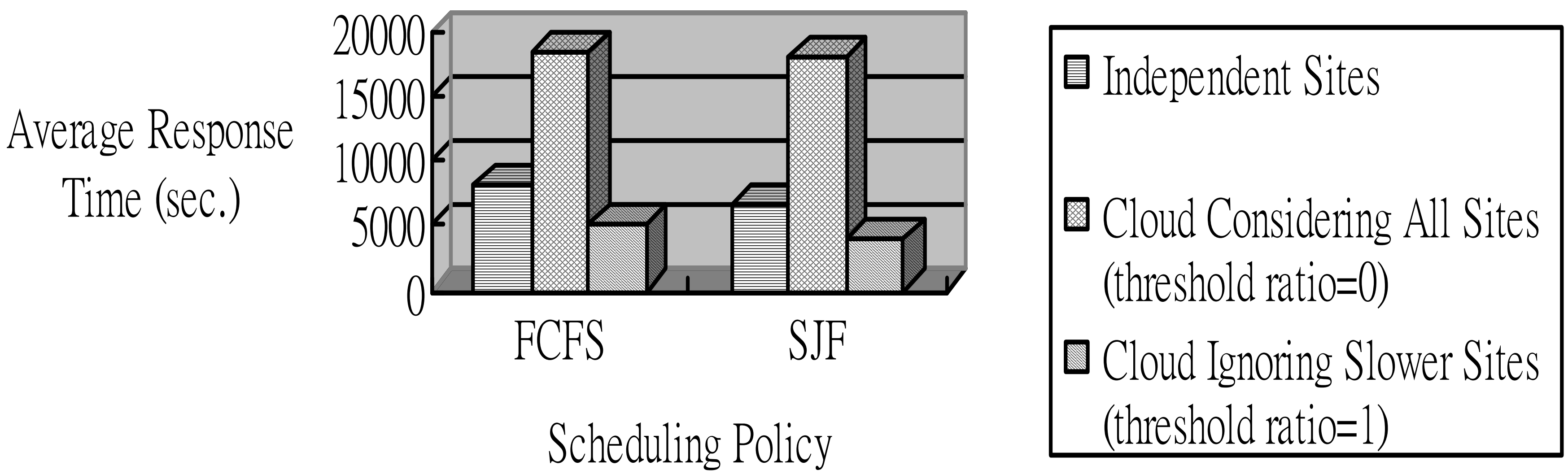


# Load sharing performance in a heterogeneous cloud using best-fit policy

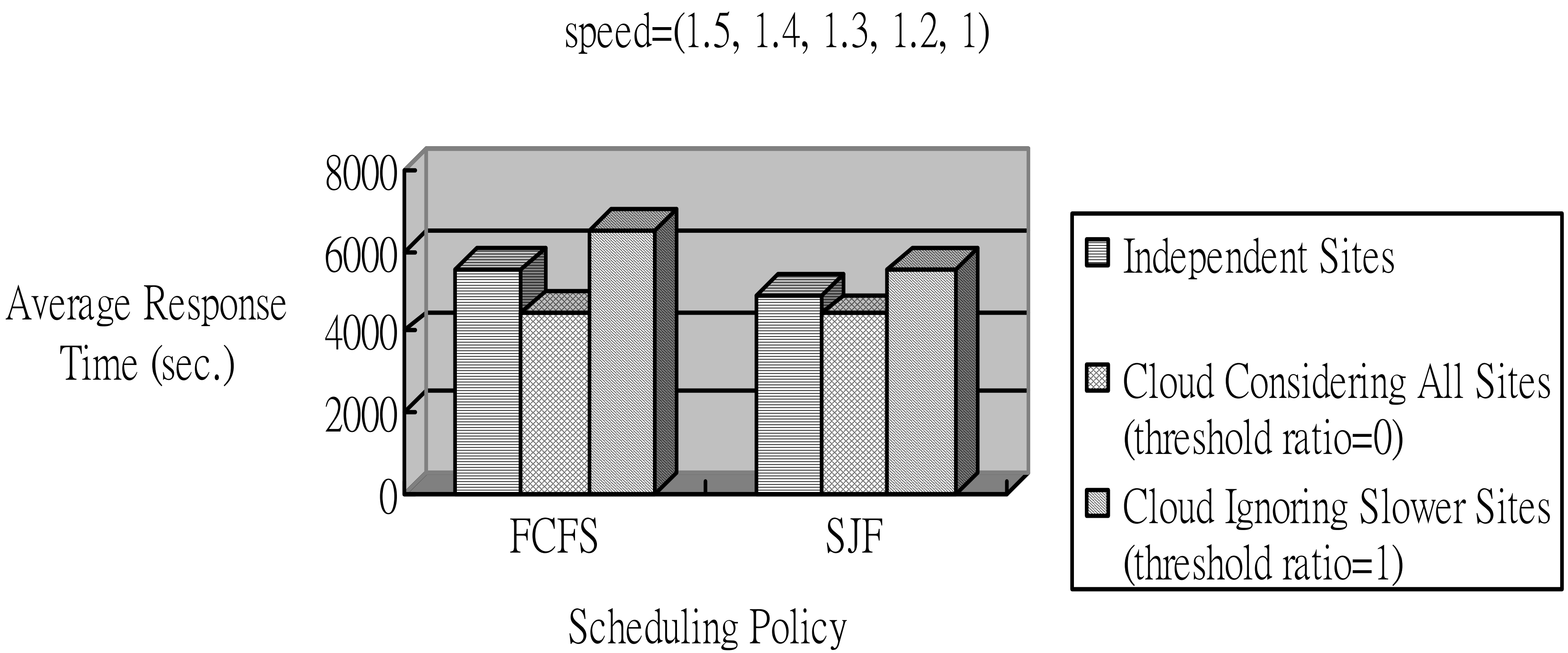


# Performance of best-fit policy with large speed difference among participating sites

speed=(0.6, 0.7, 2.4, 9.5, 4.3)

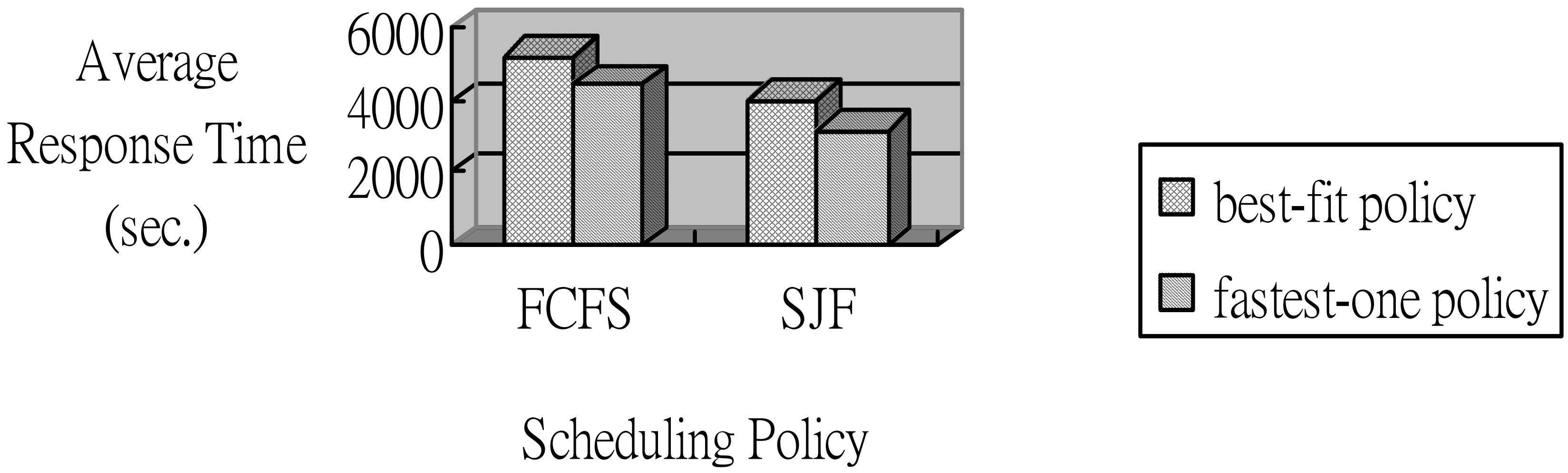


# Performance of best-fit policy with small speed difference among participating sites



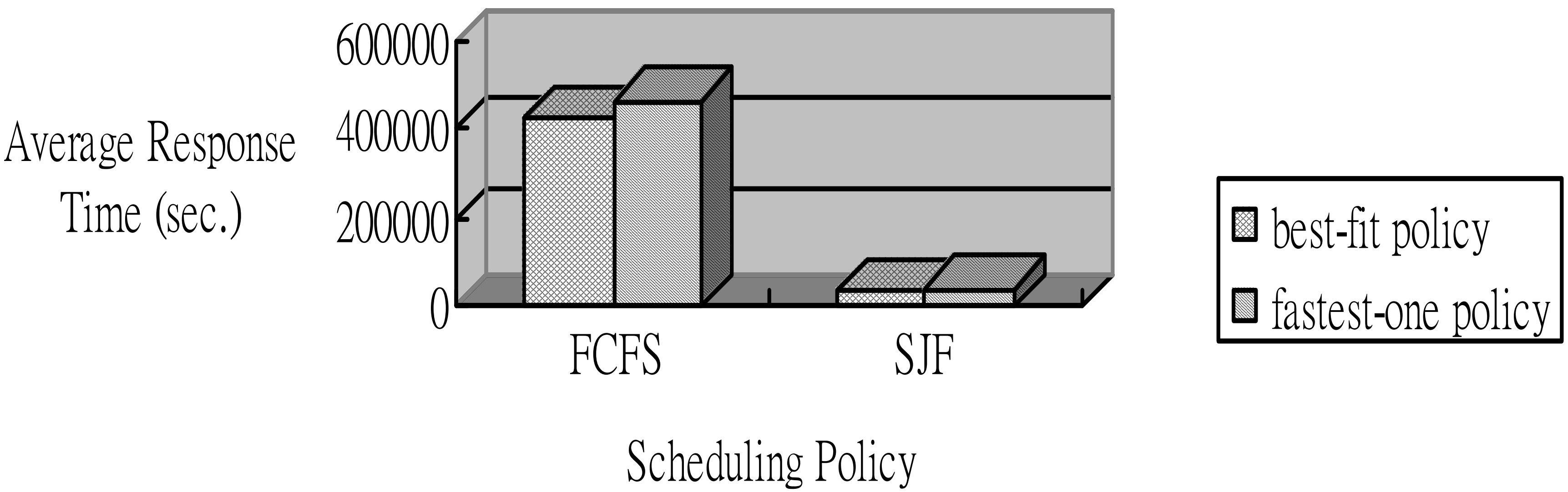
# Comparison of the fastest-one policy and the best-fit policy (I)

speed=(0.6, 0.7, 2.4, 9.5, 4.3) load=(1, 1, 1, 1, 1) Threshold Ratio=1
for Ignoring Slower Sites

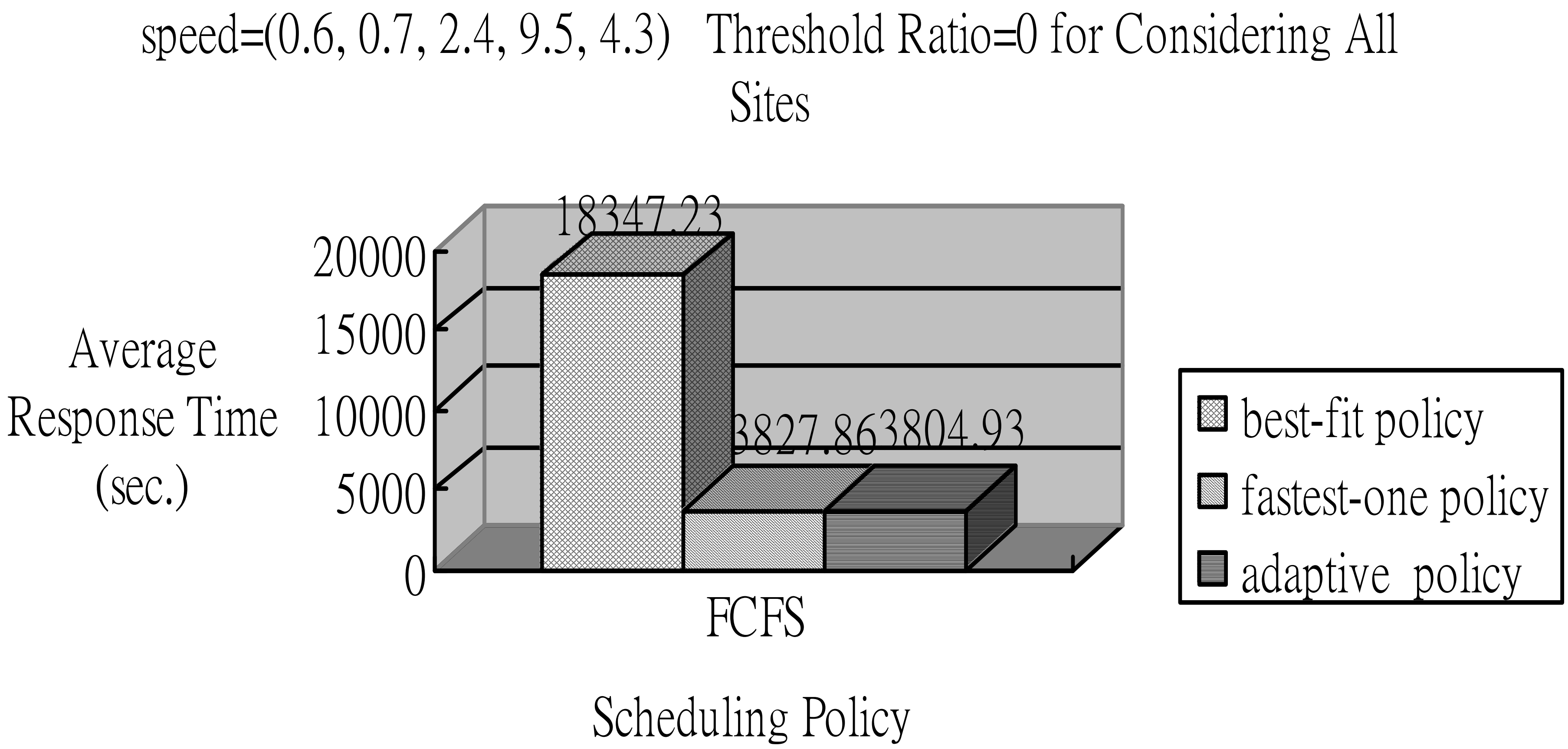


# Comparison of the fastest-one policy and the best-fit policy (II)

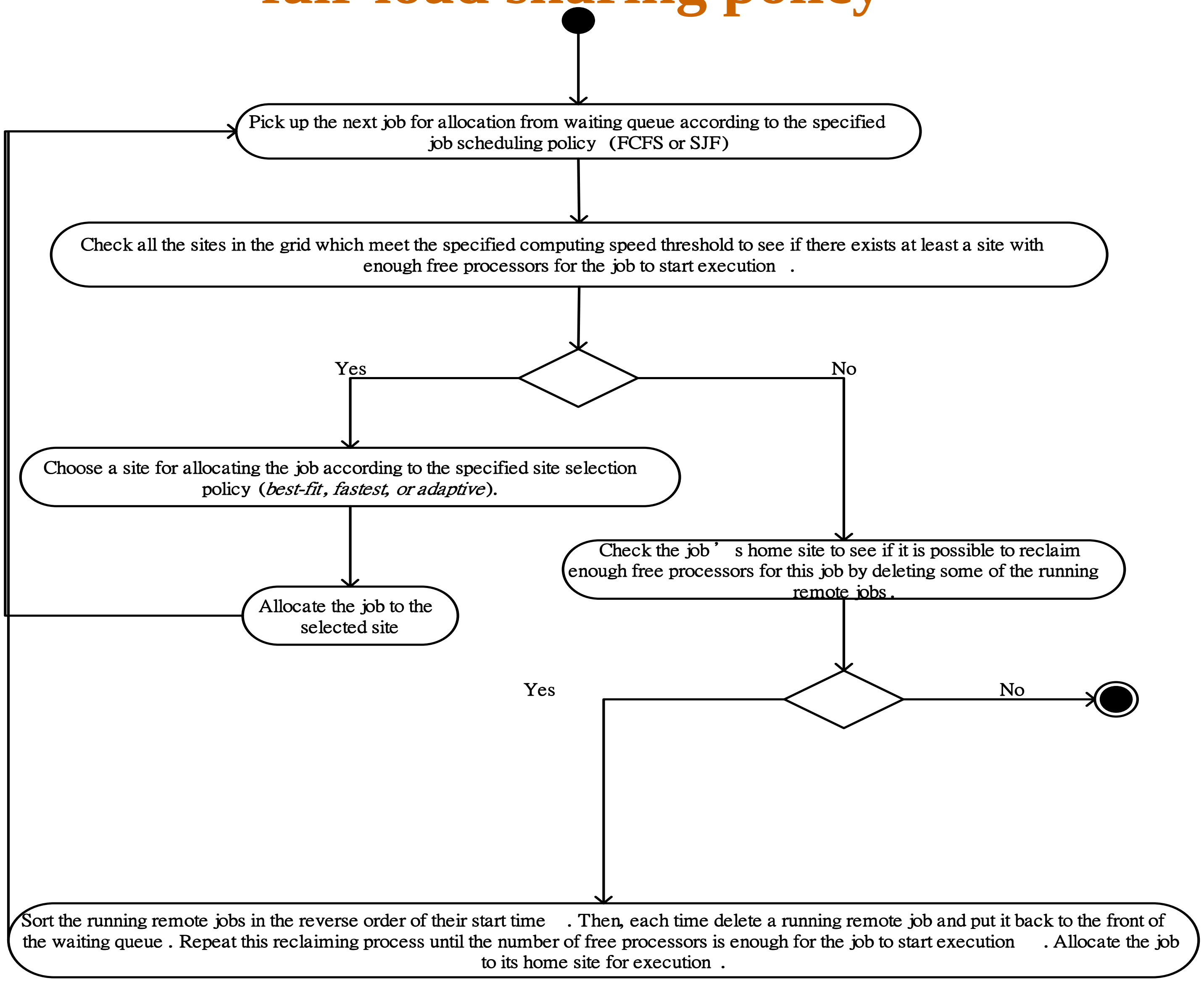
speed=(1.5, 1.4, 1.3, 1.2, 1) load=(5, 5, 5, 5, 1) Threshold Ratio=0 for Considering All Sites



# Performance of the adaptive policy



# fair load sharing policy



# Average job response times (sec.) for different load sharing policies

| | Entire grid | Site 1 | Site 2 | Site 3 | Site 4 | Site 5 |
|---|---|---|---|---|---|---|
| Independent sites | 9260 | 14216 | 10964 | 10199 | 6448 | 57 |
| Ordinary load sharing policy | 4135 | 191 | 4758 | 4799 | 3881 | 559 |
| Fair load sharing policy | 4152 | 193 | 4750 | 4798 | 3939 | 57 |

# Adaptive Processor Allocation with Estimated Job Execution Time in Heterogeneous Cloud

# Introduction

- This part presents an approach, taking advantage of the estimated job execution time, to effectively allocating processors to jobs submitted to a heterogeneous cloud.
  - Many parallel computer systems installed in computing centers worldwide, which adopts backfilling based job scheduling policies, require that users should provide estimated job execution time when submitting a job to the system.

# Introduction (Cont.)

- The proposed adaptive processor allocation approach can effectively improve the overall system performance, in terms of jobs' average turnaround time, from two to four times, compared to currently used methods.

# Introduction (Cont.)

- A cloud usually consists of several parallel or cluster computers located at different sites.
  - Communications between processors within the same site are usually achieved through high-speed networking devices
  - Messages passed across different sites have to go through a much slower wide-area network or Internet.
- A job allocated to a pool of processors within the same site can usually run faster than if it is assigned to processors across different sites.

# Introduction (Cont.)

- Processor allocation deals with the first job in the waiting queue. When the parallel job cannot fit into any single site in a heterogeneous cloud, the system, in general, may have the following choices:
  - simply keep the job waiting until a single site having enough free processors becomes available
  - allow the job to run across several sites
  - for a moldable job the system can even run it with a less number of processors than originally specified.

# Introduction (Cont.)

- The adaptive processor allocation approach dynamically makes the best allocation decision among the above allocation choices.

# The Cloud Model

- There are several independent computing sites with their own local workload. The cloud integrates the sites and shares their incoming jobs.
  - The nodes on each site run at the same speed and are linked with a fast interconnection network.
  - Each site adopts space-sharing and run the jobs in an exclusive fashion.
  - All computing nodes in the cloud are assumed to be binary compatible. The cloud is heterogeneous in the sense that nodes on different sites may differ in computing speed and different sites may have different numbers of nodes.

# The Cloud Model (Cont.)

  - The parallel jobs in this model are assumed to be moldable. We also assume the ability of jobs to run in multi-site mode.
- We used SDSC's SP2 workload logs as the input workload in the simulations.
  - The log contains records collected from May 1998 to April 2000.

| | Number of jobs | Maximum execution time | Average execution time | Maximum number of processors/job | Average number of processors/job |
|---|---|---|---|---|---|
| Queue 1 | 4053 | 21922 | 267.13 | 8 | 3 |
| Queue 2 | 6795 | 64411 | 6746.27 | 128 | 16 |
| Queue 3 | 26067 | 118561 | 5657.81 | 128 | 12 |
| Queue 4 | 19398 | 64817 | 5935.92 | 128 | 6 |
| Queue 5 | 177 | 42262 | 462.46 | 50 | 4 |
| Total | 56490 | | | | |

# Configuration of the Cloud

| | total | site 1 | site 2 | site 3 | site 4 | site 5 |
|---|---|---|---|---|---|---|
| Number of processors | 442 | 8 | 128 | 128 | 128 | 50 |

# The Cloud Model (Cont.)

- we define a speed vector, speed=(sp1,sp2,sp3,sp4,sp5), to describe the relative computing speeds of all the five sites in the cloud.
- We also define a load vector, load=(ld1,ld2,ld3,ld4,ld5), which is used to derive different loading levels from the original workload data by multiplying the load value ld*i* to the execution times of all jobs at site *i*.

# Current Allocation Practices

- Multi-pool configuration
  - Each job must be allocated to exactly one site. If a job cannot fit into any single site in the cloud, it would have to wait.
- Multi-site parallel execution.
  - Run a parallel job across several sites if there is no single site having enough free processors.
- Moldable processor allocation.
  - Instead of keeping the job waiting in queue, the system automatically scales the job’s parallelism down to use exactly the number of free processors.

# Current Allocation Practices (Cont.)

- However, none of the above three approaches can consistently deliver the best performance under different workloads or system configurations.

Table 3. Performance under speed vector (1,5,4,5,2) and load vector (1,1,1,1,1)

| | Average turnaround time (sec.) | Average queue length |
|---|---|---|
| Multi-pool | 1102 | 0.06 |
| Multi-site (slowdown=1.5) | 1103 | 0.05 |
| Multi-site (slowdown=2.0) | 1103 | 0.05 |
| Multi-site (slowdown=2.5) | 1104 | 0.05 |
| Multi-site (slowdown=3.0) | 1105 | 0.05 |
| Moldable | 1109 | 0.03 |

Table 4. Performance under speed vector (4,1,2,1,5) and load vector (1,1,1,1,1)

| | Average turnaround time (sec.) | Average queue length |
|---|---|---|
| Multi-pool | 1576 | 0.16 |
| Multi-site (slowdown=1.5) | 1560 | 0.13 |
| Multi-site (slowdown=2.0) | 1565 | 0.13 |
| Multi-site (slowdown=2.5) | 1571 | 0.13 |

Table 5. Performance under speed vector (1,5,4,5,2) and load vector (5,5,5,5,5)

| | Average turnaround time (sec.) | Average queue length |
|---|---|---|
| Multi-pool | 5708 | 0.44 |
| Multi-site (slowdown=1.5) | 5748 | 0.40 |
| Multi-site (slowdown=2.0) | 5827 | 0.44 |
| Multi-site (slowdown=2.5) | 5895 | 0.45 |
| Multi-site (slowdown=3.0) | 6088 | 0.50 |
| Moldable | 5799 | 0.18 |

Table 6. Performance under speed vector (4,1,2,1,5) and load vector (5,5,5,5,5)

| | Average turnaround time (sec.) | Average queue length |
|---|---|---|
| Multi-pool | 33432 | 19.58 |
| Multi-site (slowdown=1.5) | 1419841 | 594 |
| Multi-site (slowdown=2.0) | 3705479 | 1369 |
| Multi-site (slowdown=2.5) | 10036993 | 5731 |
| Multi-site (slowdown=3.0) | 14508657 | 7307 |
| Moldable | 13525 | 0.63 |

# Adaptive Processor Allocation

Variables:

T0, T1, T2, T3: execution times for processor allocation policies;
S0, S1, S2, S3: set of sites chosen by processor allocation policies;

```
While (job queue is not empty)
  {
    Pick up the first job from the job queue;
    If (any cluster being able to accommodate the job)
      {
        Apply the fastest-one policy to compute and set
        T0 = the execution time to be taken
        S0 = the target site
      }
```

```
Else
   {
     T1 = Min{Ti}, where Ti is (the estimated waiting time + the
            estimated single-site runtime) for site i in the grid sites
     S1 = the site taking the total turnaround time T1

     if (total free processors in the grid is enough for multi-site allocation)
       {
        T2 = the estimated runtime of multi-site allocation
        S2 = the set of sites for the allocation
       }
      else
         T2 = ∞;
```

```
      if (total number of free processors in the grid is not zero)
        {
         T3 = Min{Tj}, where Tj is the estimated runtime when
                 scaling down to site j with non-zero processors
         S3 = the site taking execution time T3
         }
      else
          T3 = ∞
  }
if (T0 is the shortest time)
  Run the job on site S0;
else if (T1 is the shortest time)
        Keep the job waiting in queue until site S1 have enough processors;
       else if (T2 is the shortest time)
                Apply multi-site parallel execution using the sites in S2;
              else
                Scale down parallelism for immediate execution on site S3;
}
```

Table 7. Performance under speed vector (1,5,4,5,2) and load vector(1,1,1,1,1)

| | Average turnaround time (sec.) |
|---|---|
| Multi-pool | 1102 |
| Multi-site (slowdown=1.5) | 1103 |
| Moldable | 1109 |
| Proposed approach (slowdown=1.5) | 254 |

Table 8. Performance under speed vector (4,1,2,1,5) and load vector(1,1,1,1,1)

| | Average turnaround time (sec.) |
|---|---|
| Multi-pool | 1576 |
| Multi-site (slowdown=1.5) | 1560 |
| Moldable | 1547 |
| Proposed approach (slowdown=1.5) | 477 |

Table 9. Performance under speed vector (1,5,4,5,2) and load vector(5,5,5,5,5)

| | Average turnaround time (sec.) |
|---|---|
| Multi-pool | 5708 |
| Multi-site (slowdown=1.5) | 5748 |
| Moldable | 5799 |
| Proposed approach (slowdown=1.5) | 1735 |

Table 10. Performance under speed vector (4,1,2,1,5) and load vector(5,5,5,5,5)

| | Average turnaround time (sec.) |
|---|---|
| Multi-pool | 33432 |
| Multi-site (slowdown=1.5) | 1419841 |
| Moldable | 13525 |
| Proposed approach (slowdown=1.5) | 6162 |

# Summary

- This part investigates the processor allocation issue in heterogeneous cloud environments. An adaptive processor allocation approach is proposed, which takes advantage of the estimated job execution time commonly required by many parallel systems in computing centers.
- The proposed approach can dynamically make the best allocation decision among several allocation choices.
- The simulation results indicate that the proposed approach can consistently outperform the three current practices under different workload and system configurations.

## Summary (Cont.)

- The adaptive approach effectively improves the overall system performance, in terms of jobs’ average turnaround time, from two to four times under different conditions.

# On Effects of Resource Fragmentation on Job Scheduling Performance in a Multi-Cluster Cloud

# Introduction

- This part presents the studies on analysis of job scheduling performance in a multi-cluster cloud from the perspective of resource fragmentation.
- Two parts of job scheduling will impact on resource fragmentation:
  - job selection
  - site selection.
- A series of simulations have been conducted to investigate the effects of resource fragmentation in terms of average waiting time of all jobs.

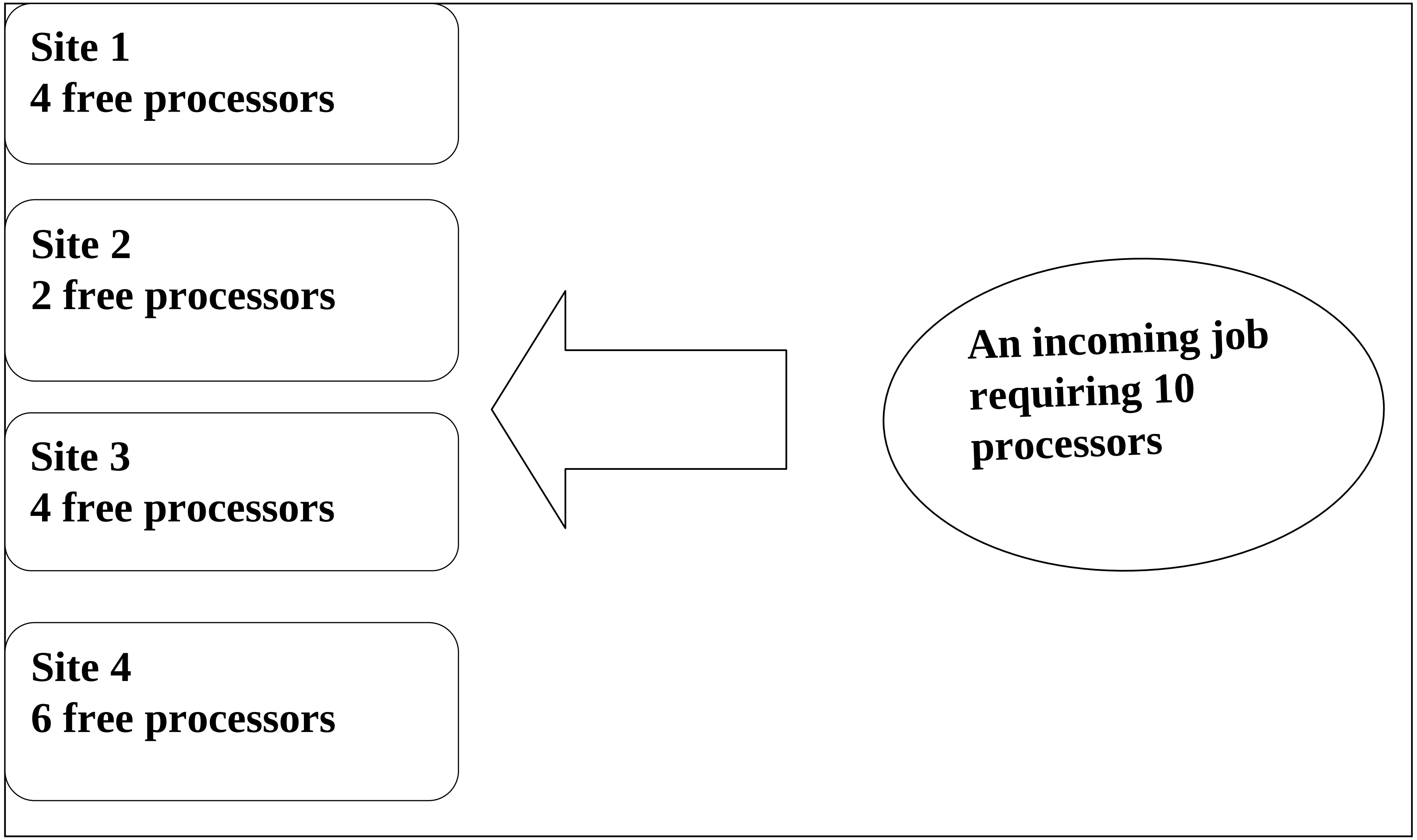
# Resource Fragmentation Issues

- Resource utilization has a significant impact on system performance and thus has been a research topic in many kinds of computer systems.
  - For example, dynamic memory allocation methods were developed to alleviate external fragmentation in memory space.
  - Reduced external fragmentation implies more efficient resource utilization and can lead to improved system performance because of being able to support more applications simultaneously.

# Resource Fragmentation Issues (Cont.)

- On parallel or cluster computer systems, backfilling scheduling methods have been proposed to improve jobs' turnaround time through improving resource utilization.
- This part discusses the resource fragmentation issue in a cloud, consisting of multiple clusters or parallel computers, and investigate its effects on job scheduling performance.

# An Example of  Resource Fragmentation



# Resource Fragmentation Issues (Cont.)

- On parallel or cluster computer systems, job scheduling mainly determines the sequence of starting execution for the jobs waiting in the queue
  - called *job selection* in the following.
- Since a cloud is composed of several sites and intends to allocate a parallel job onto a single site, job scheduling in cloud has to include a second step after job selection.
  - The second step, called *site selection* in the following, chooses an appropriate site with enough free processors for allocating the selected job in the job selection step.

# Resource Fragmentation Issues (Cont.)

- The following studies the resource fragmentation effects under different job selection and site selection methods

# Simulation Model

- Our simulation studies were based on publicly downloadable workload traces.
  - We used the SDSC's SP2 workload logs as the input workload in the simulations.
  - 73496 job records collected on a 128-node IBM SP2 machine from May 1998 to April 2000
  - 56490 job records are used in the simulations after excluding some problematic records based on the *completed* field in the log.

# Job Characteristics in SDSC's SP2 Log

| | Number of jobs | Maximum execution time | Average execution time | Maximum number of processors/job | Average number of processors/job |
|---|---|---|---|---|---|
| Queue 1 | 4053 | 21922 | 267.13 | 8 | 3 |
| Queue 2 | 6795 | 64411 | 6746.27 | 128 | 16 |
| Queue 3 | 26067 | 118561 | 5657.81 | 128 | 12 |
| Queue 4 | 19398 | 64817 | 5935.92 | 128 | 6 |
| Queue 5 | 177 | 42262 | 462.46 | 50 | 4 |
| Total | 56490 | | | | |

# Configuration of the Cloud

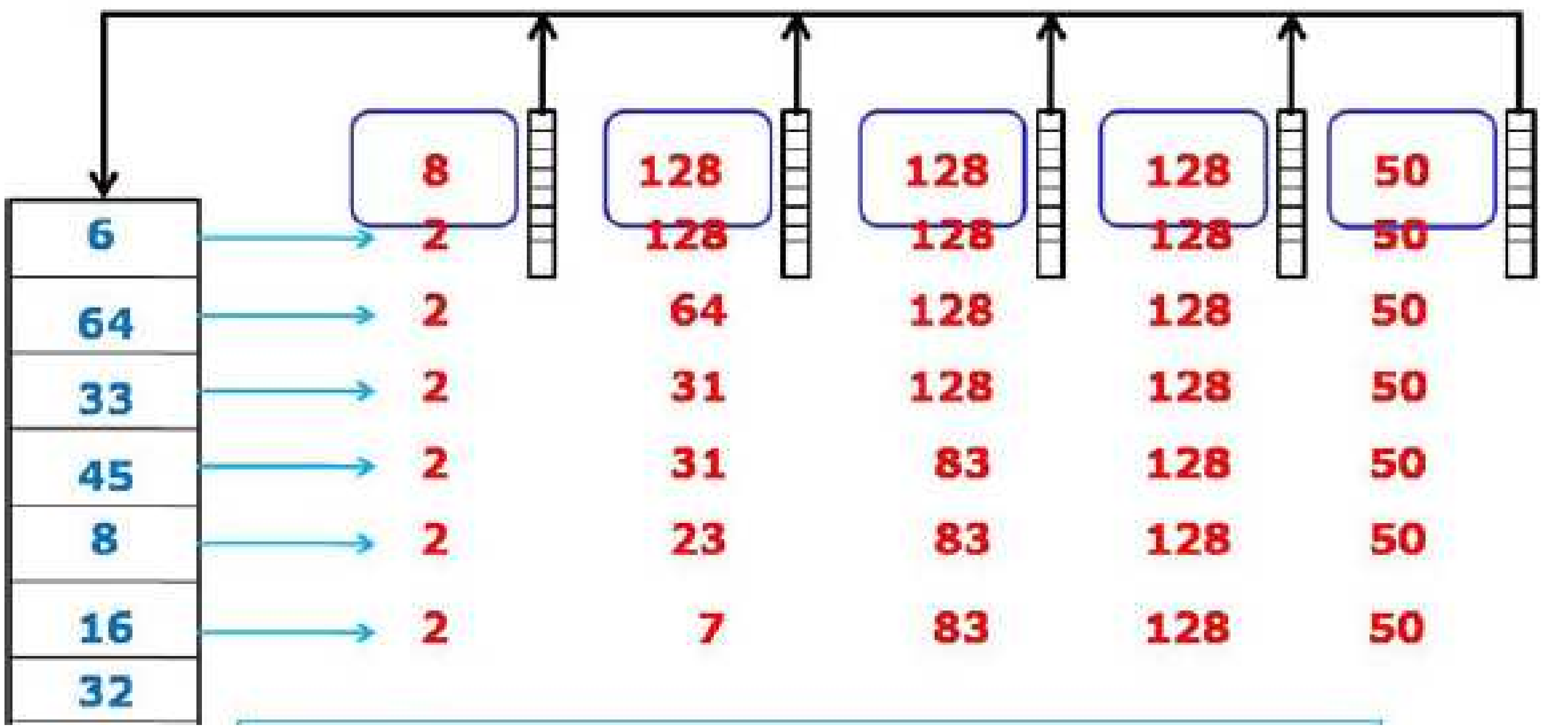
| | total | site 1 | site 2 | site 3 | site 4 | site 5 |
|---|---|---|---|---|---|---|
| Number of processors | 442 | 8 | 128 | 128 | 128 | 50 |

- We define a load vector, load=(ld1,ld2,ld3,ld4,ld5), to study different workload conditions from the original workload data by multiplying the load value ld*i* to the execution times of all jobs at site *i*.

# Site Selection Methods

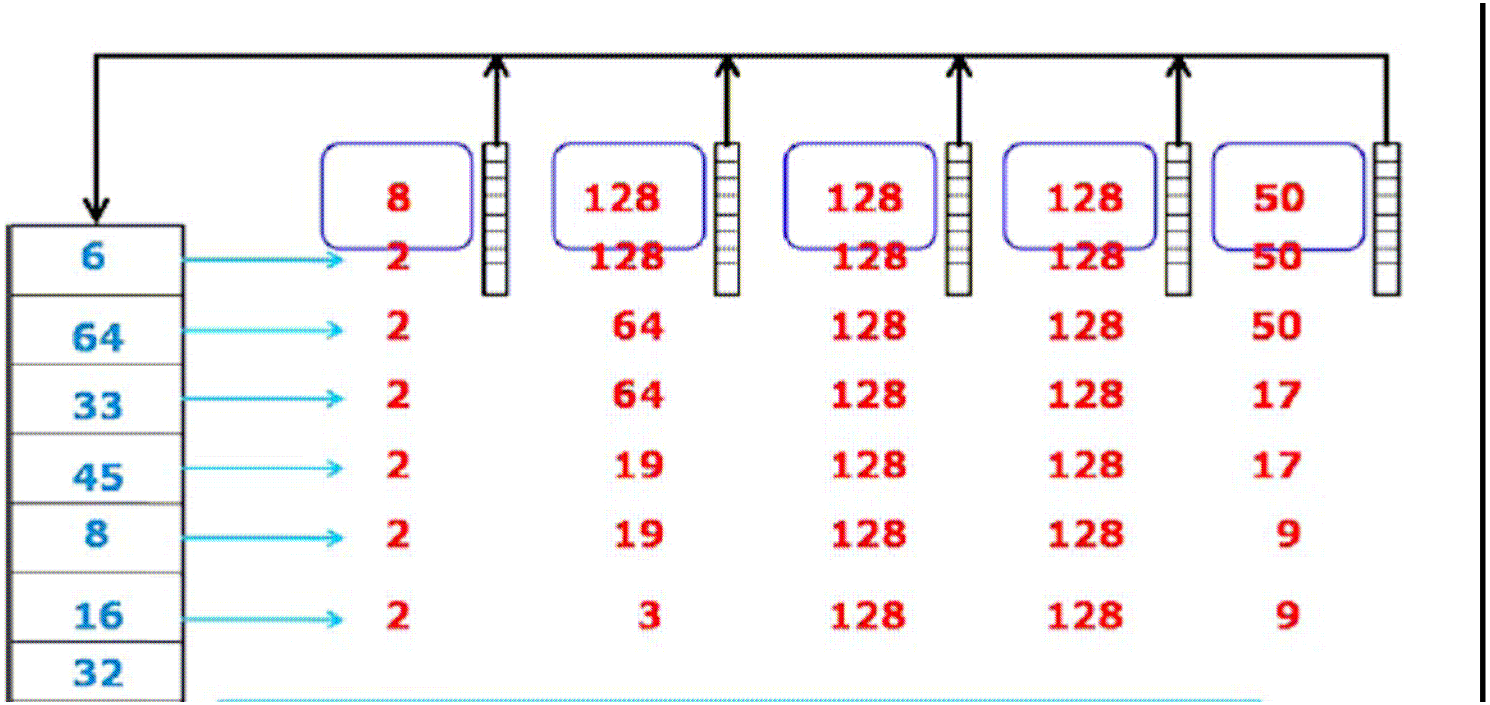
- First-fit
- Best-fit
- Worst-fit
- Median-fit
- Random-fit

# First Fit



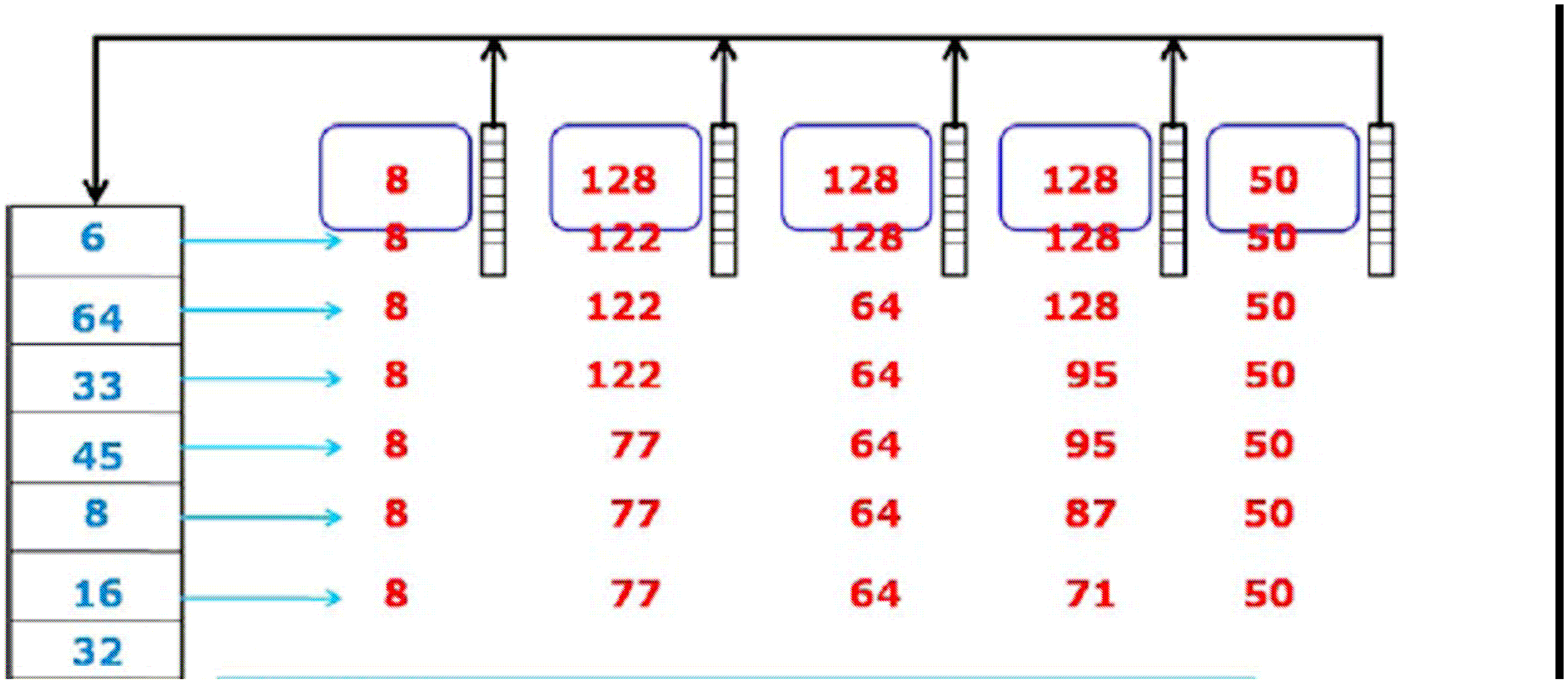
8
128
128
128
50
6
2
128
128
128
50
64
2
64
128
128
50
33
2
31
128
128
50
45
2
31
83
128
50
8
2
23
83
128
50
16
2
7
83
128
50
32

# Best Fit



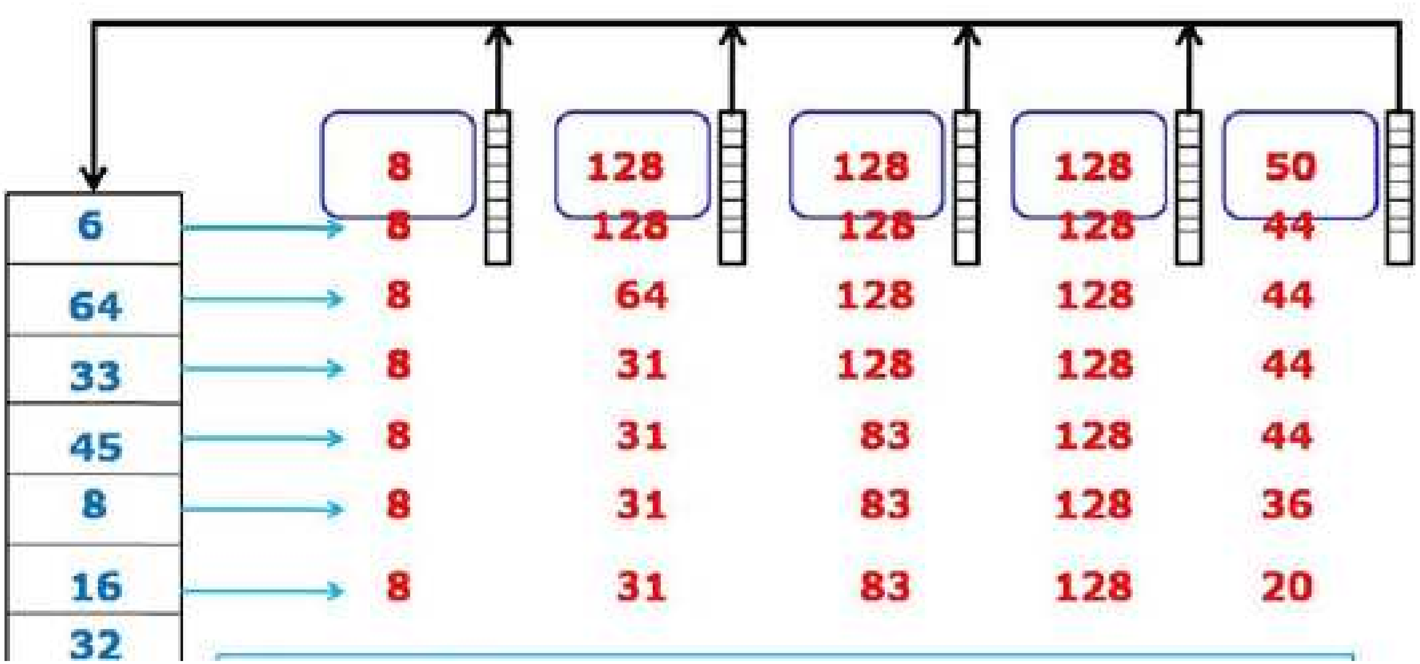
6
64
33
45
8
16
32
8
2
2
2
2
2
2
128
128
64
64
19
19
3
128
128
128
128
128
128
128
128
128
128
128
128
128
128
50
50
50
17
17
9
9

# Worst Fit



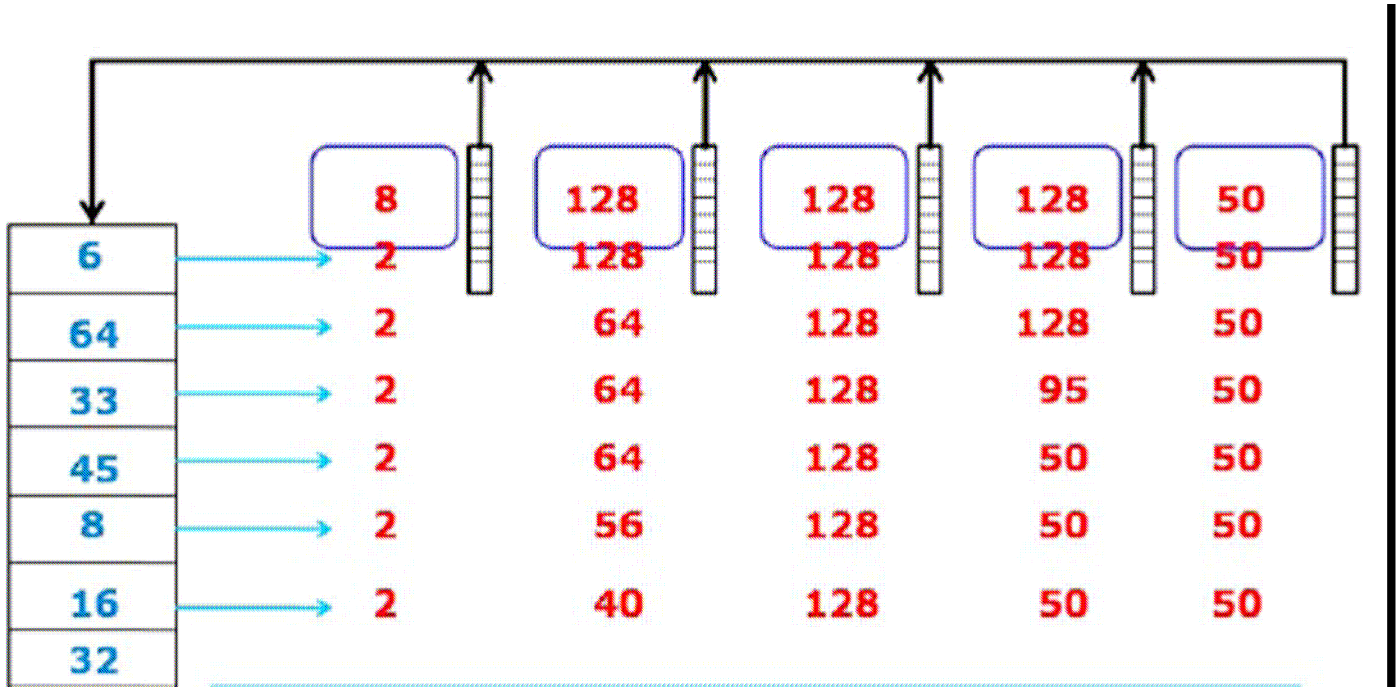
8
128
128
128
50
6
8
122
128
128
50
64
8
122
64
128
50
33
8
122
64
95
50
45
8
77
64
95
50
8
8
77
64
87
50
16
8
77
64
71
50
32

# Median Fit



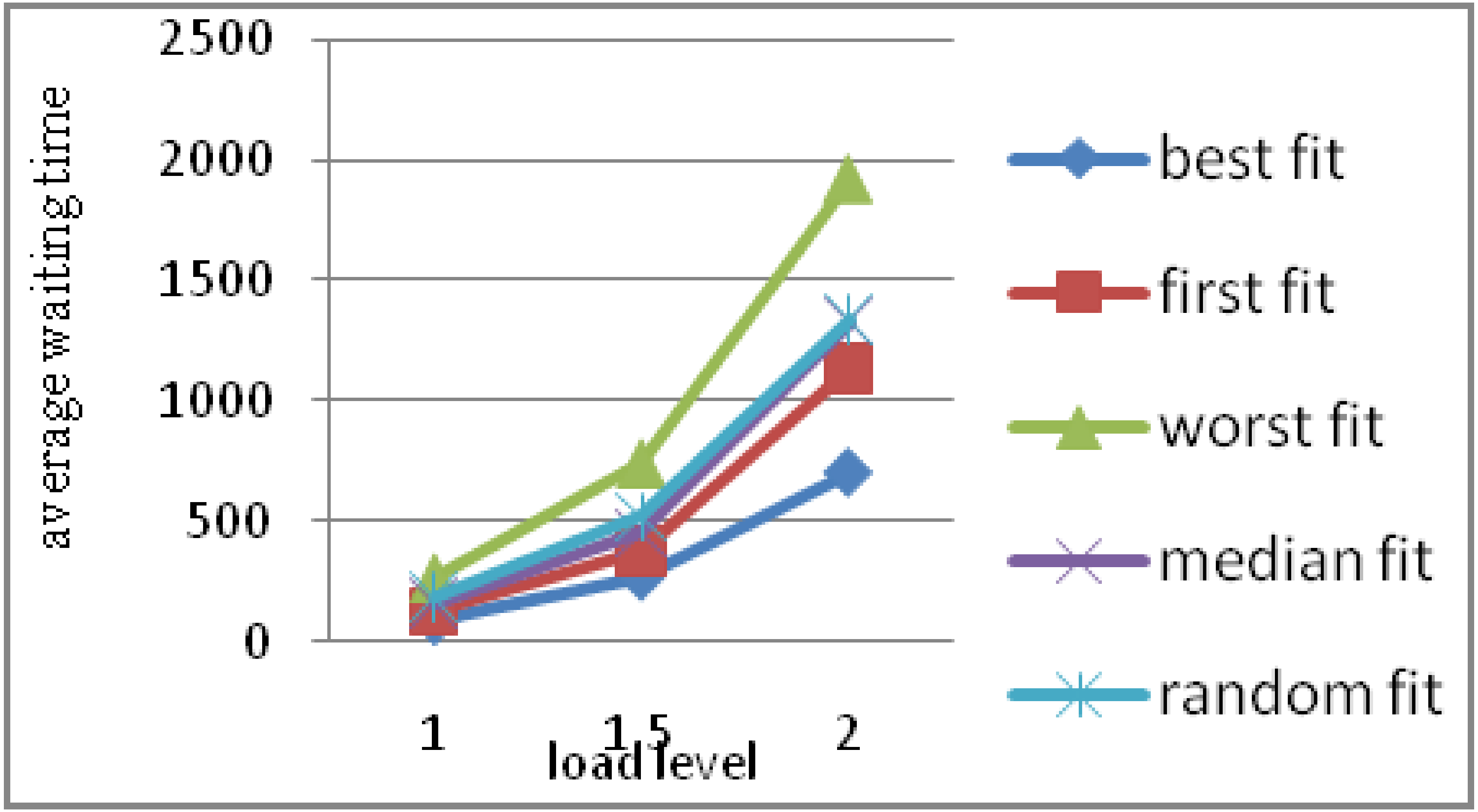
8
128
128
128
50
6
8
128
128
128
44
64
8
64
128
128
44
33
8
31
128
128
44
45
8
31
83
128
44
8
8
31
83
128
36
16
8
31
83
128
20
32

# Random Fit

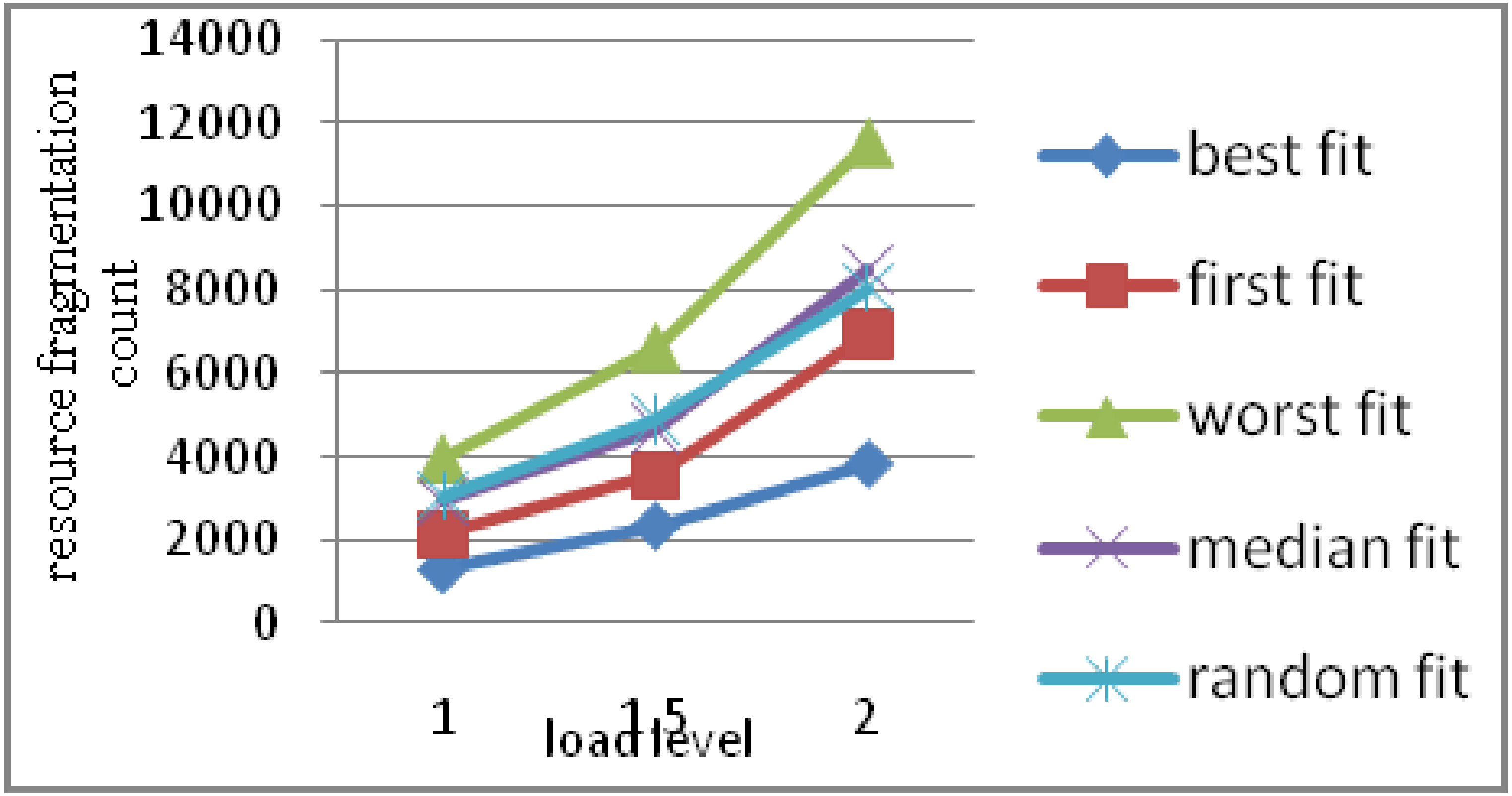


8
128
128
128
50
6
2
128
128
128
50
64
2
64
128
128
50
33
2
64
128
95
50
45
2
64
128
50
50
8
2
56
128
50
50
16
2
40
128
50
50
32

# Performance under FCFS Scheduling



# Resource Fragmentation Counts

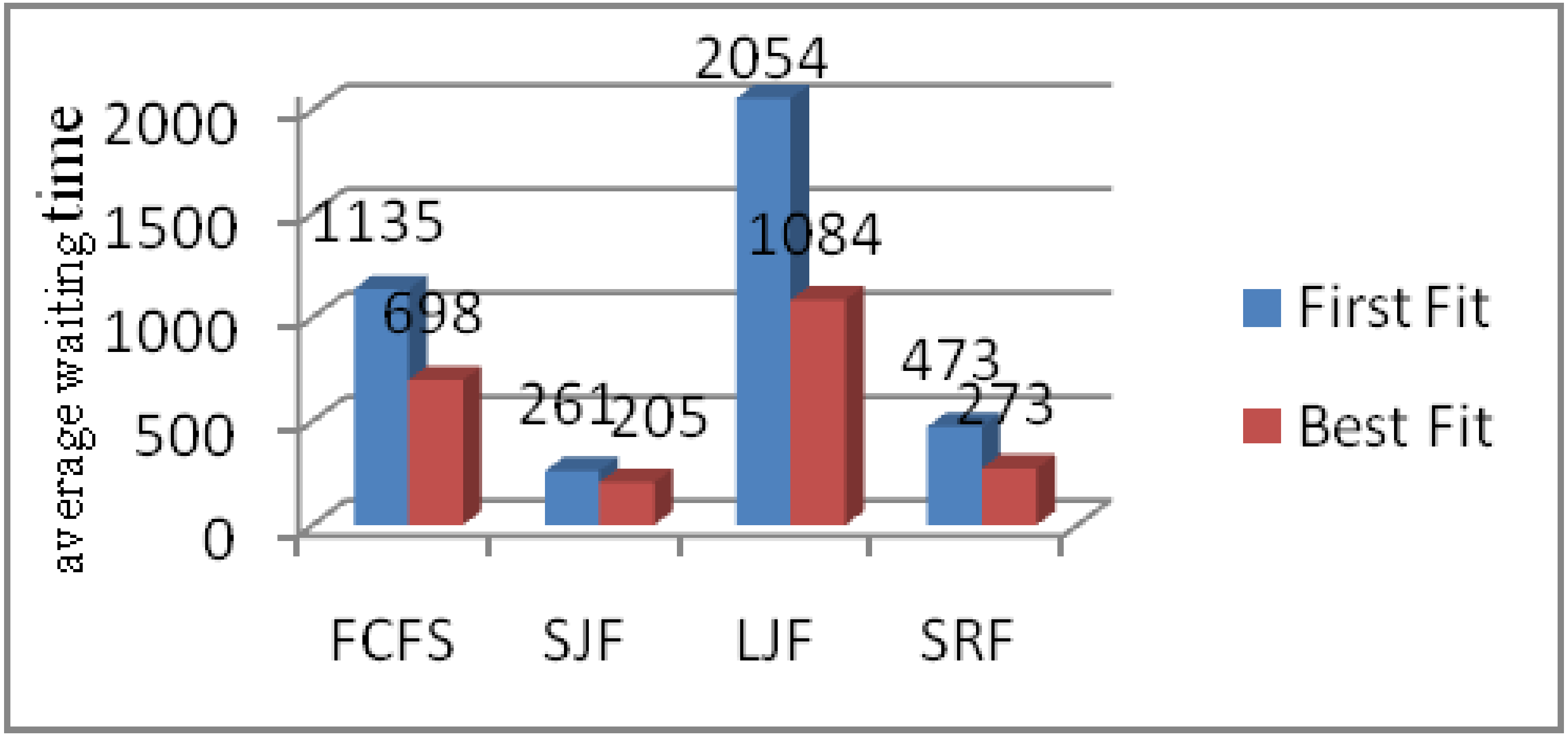



These results indicate that resource fragmentation is the main cause of the performance difference among different site allocation methods.

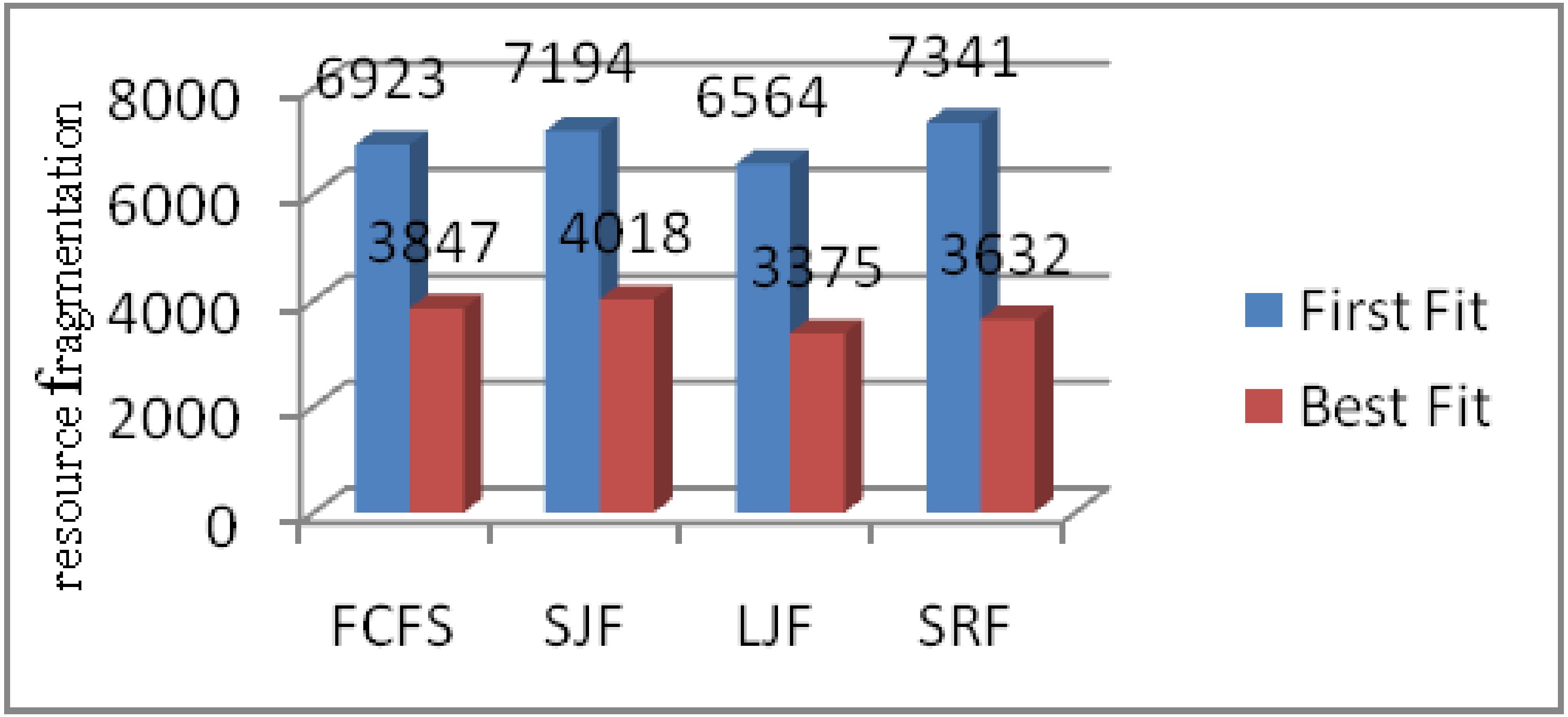
# Job Scheduling Methods

- First-Come, First-Served (FCFS)
- Smallest job first (SJF)
- Largest job first (LJF)
- Shortest runtime first (SRF)

# Performance of Job Scheduling Methods



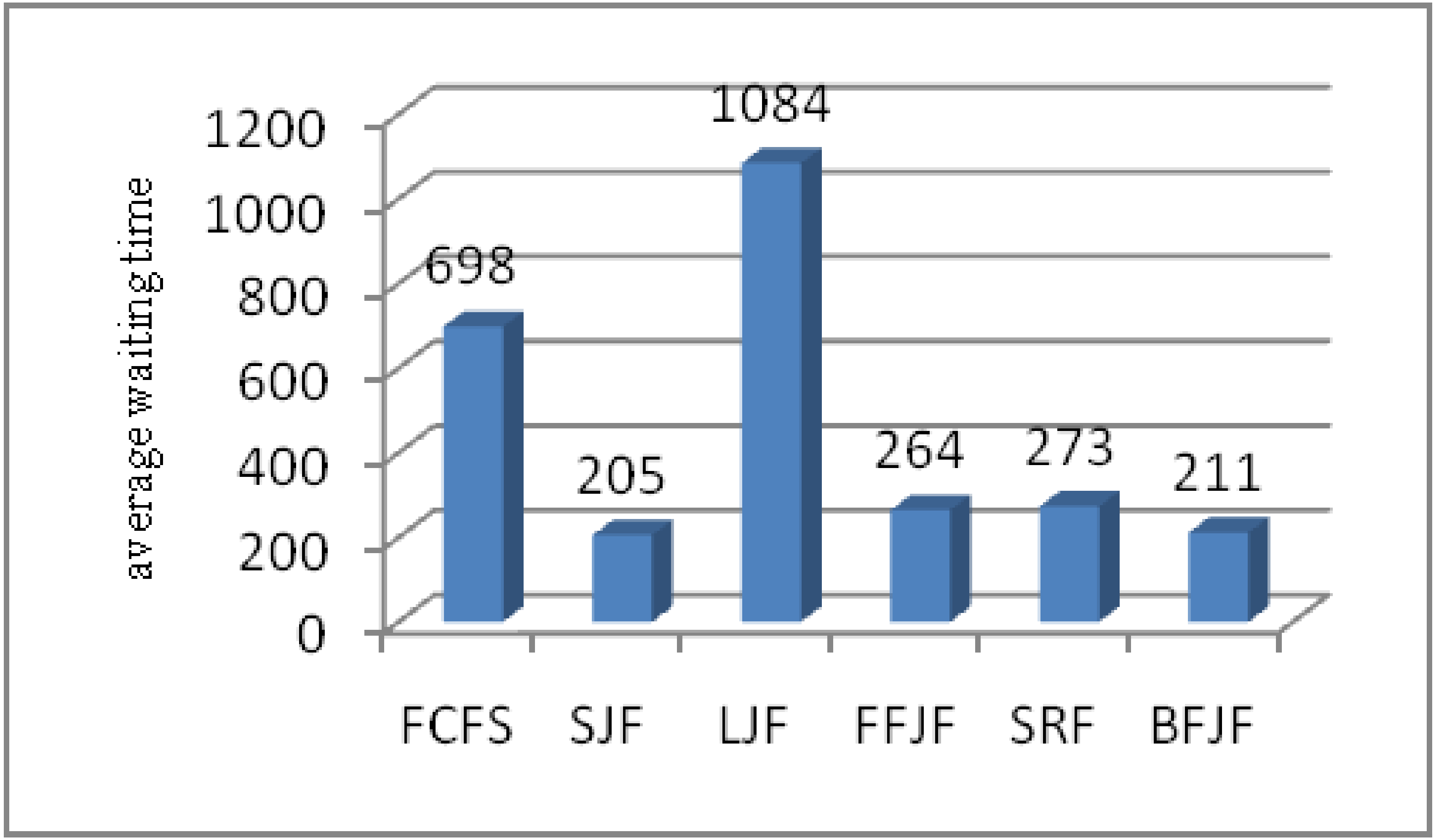
# Resource Fragmentation Counts



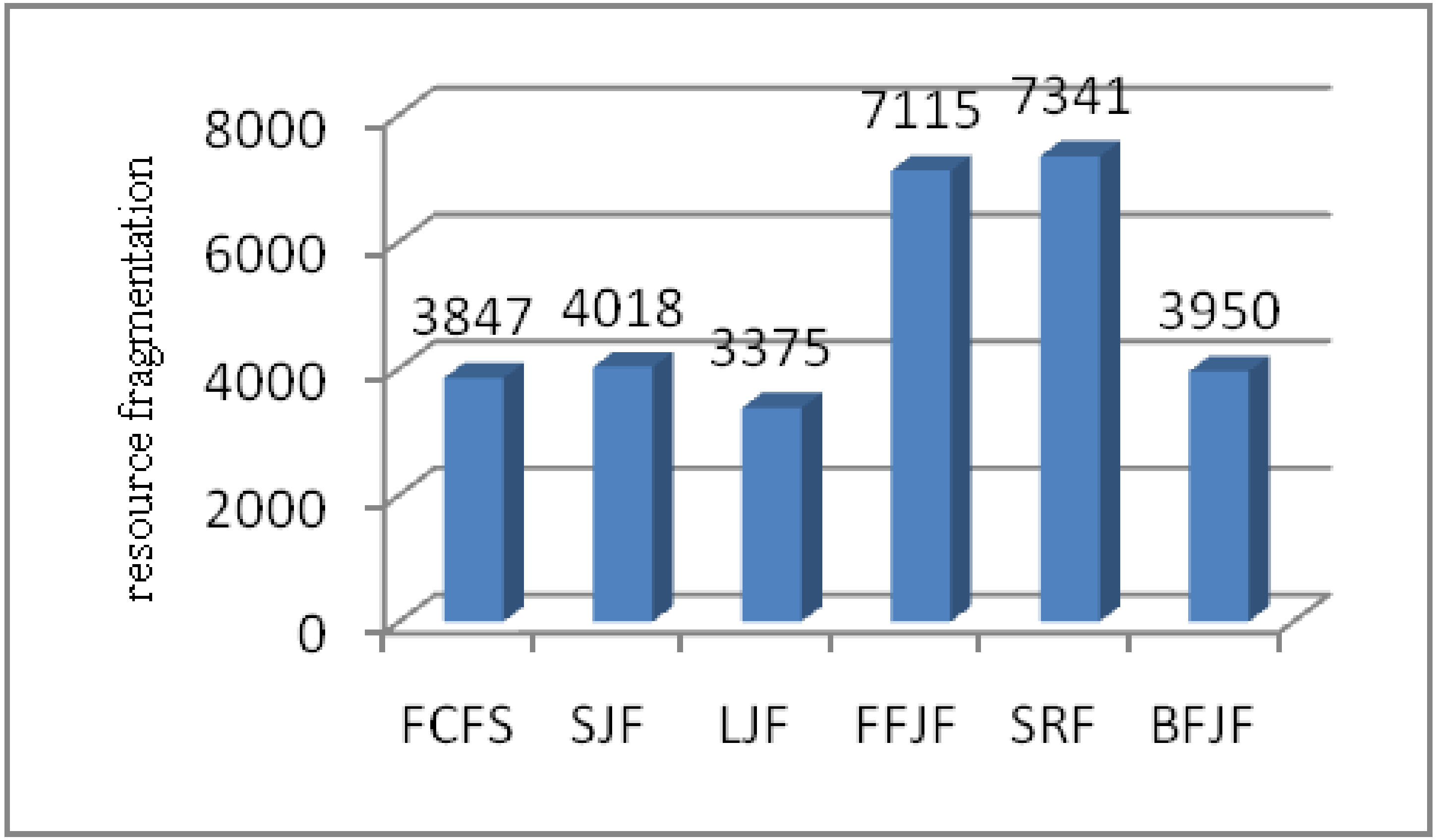
# CO-CONSIDERATION OF JOB SELECTION AND SITE SELECTION

- First-fit job first (FFJF)
- Best-fit job first (BFJF)

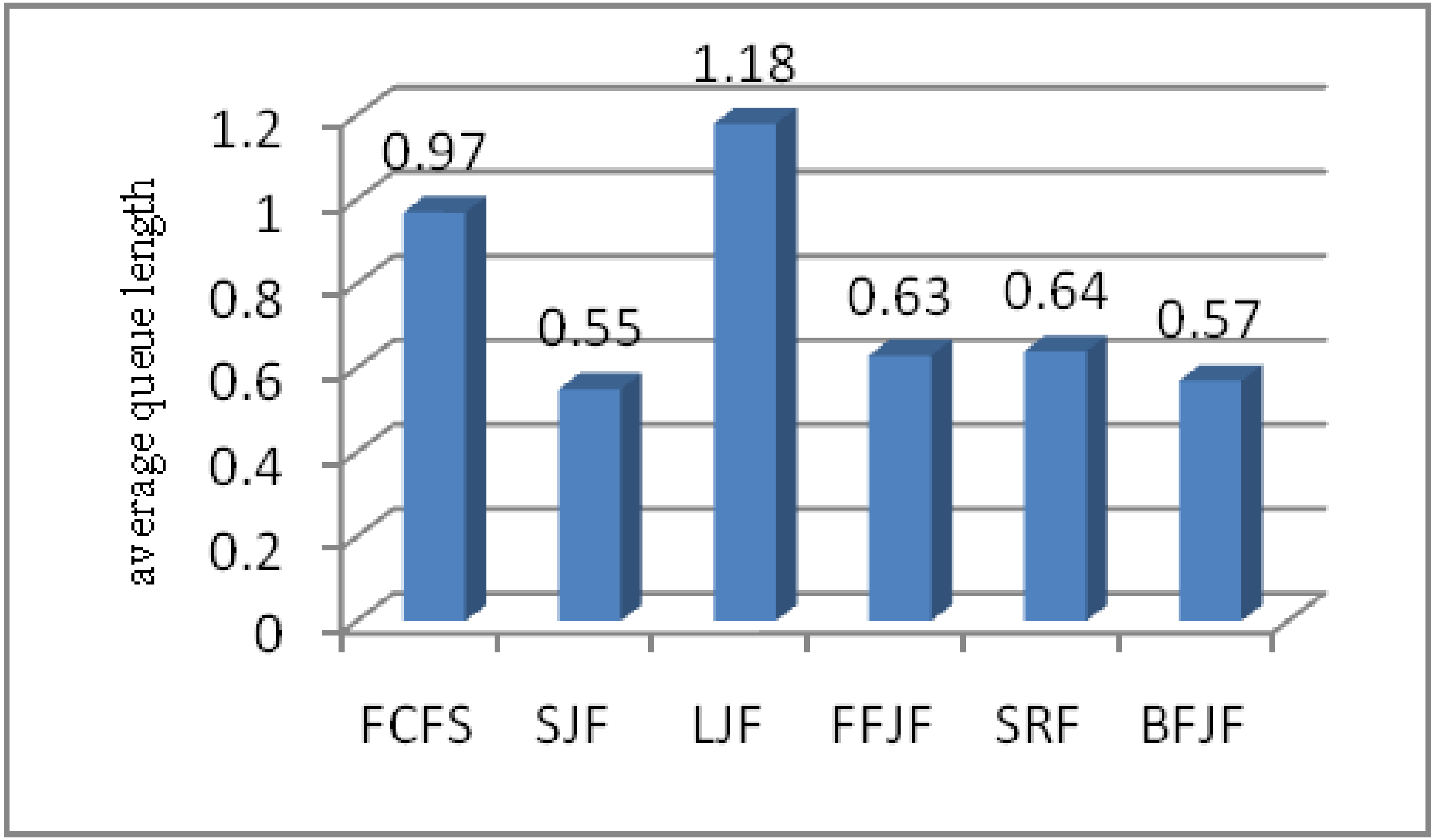
# Performance Results



# Resource Fragmentation Counts



# Average Queue Length



# Discussions

- Methods co-considering job selection and site selection in one single step seem bringing no significant improvement.
- resource fragmentation count is not necessarily proportional to the average waiting time for job selection methods.
  - resource fragmentation may not be the sole effect on job selection performance.

# Discussions (Cont.)

- In addition to resource fragmentation, keeping as many jobs running as possible may be another important performance factor when average waiting time is concerned.
  - More jobs running implies less jobs waiting queue.
- As job selection performance is concerned average queue length may have stronger effects than resource fragmentation.

# Summary

- This paper presents the work in analyzing the underlying causes that lead to the performance difference between different job scheduling methods in multi-cluster cloud environments.
- The performance results in the experiments indicate that resource fragmentation plays an important role on job scheduling performance.

# Summary (Cont.)

- Good job selection and site selection mechanisms are proposed to form an effective job scheduling method which could reduce resource fragmentation and thus improve system performance.
  - Achieving more than five times performance improvement compared to primitive job scheduling methods.

# Scheduling Task-Parallel Jobs in Parallel and Distributed Systems

# Task Graph (DAG)

- A task graph is a directed acyclic graph G=(V,E,w,c) representing a job J.
  - The nodes in V represent the tasks of J.
  - The edges in E representing the communications between the tasks.
  - The positive weight w(n) associated with node n represents its computation cost.
  - The nonnegative weight $c(e_{ij})$ associated with edge $e_{ij}$ represents its communication cost.

# Task Graph

- A task graph can be used to represent the structure of a program, where a task is a statement, or the structure of a large job, where each task may itself be the execution of a specific program.
- Limitations
  - It does not provide any mechanism to efficiently represent an iterative computation.
  - It does not exhibit conditional execution; that is, there is no branching.

# Task Graph

- Exercise:
  - Construct a task graph for the code below. Each line shall be represented by one task, named by its line number, and the costs shall be assumed as follows:
    - Computation. Assignment alone: 1 unit; add/subtract operation: 2 units; multiply operation: 3 units; divide operation: 4 units.
    - Communication. Communicating a variable with a small letter and with a capital letter costs 1 unit and 2 units, respectively.

```
1:  a = 56
2:  b = a * 10 + 2
3:  C = (b – 2) / 3
4:  D = 91.125
5:  E = D * a
6:  F = D * b + 1
7:  g = 11 + a
8:  H = (E + F) * g
```

# Computer Representation of Graphs

- There are two standard ways to represent a graph G = (V,E):
  - as a collection of adjacency lists
  - as an adjacency matrix
- Adjacency list representation
  - A graph can be represented as a array of |V| adjacency lists, one for each vertex in V. the adjacency list belonging to vertex u contains pointers to all vertex v that are adjacent to u.
  - It has the disadvantage that there is no quicker way to determine of an edge $e_{uv}$ is part of a graph G than to search in u’s adjacency list.
  - suitable for sparse graph.

# Computer Representation of Graphs

- Adjacency matrix representation
  - A graph is represented by a |V|×|V| matrix A. each element $a_{ij}$ of the matrix A has one of two possible values: 1 if the edge eij exists and 0 otherwise.
  - It uses more memory space than adjacency list representation.
  - suitable for dense graphs, or when the fast determination of the existence of an edge is crucial.

# Computer Representation of Graphs

- Exercise:
  - Give an adjacency matrix representation and an adjacency list representation of the task graph for the previous exercise.

# Topological Order

- A topological order of a directed acyclic graph G=(P,E) is a linear ordering of all its vertices such that if E contains an edge $e_{uv}$, then u appears before v in the ordering.
- A directed graph G=(P,E) is acyclic if and only if there exists a topological order of its vertices.
- Algorithm Topological-Sort(G)

Execute DFS(G), Depth First Search, with the following addition:

Insert each vertex of G onto the front of a list L as soon as it is marked finished

Return L

# Topological Order

- Algorithm DFS(G)

```
for each vertex v in G do
    if v not discovered then
        DFS-Visit(v)
    end if
End for
```

- Algorithm DFS-Visit(u)

```
for each adjacent vertex v of u do
    if v not discovered then
        Mark v as discovered
        DFS-Visit(v)
    end if
end for
Mark u as finished
```

# Topological Order

- Topological order of a task graph is useful when scheduling a task graph onto a single CPU, but is not enough when scheduling a task graph onto a parallel system.
- Exercise:
  - Find a topological order for the task graph in the previous execrise.

# Task Scheduling

- Here, static task scheduling is addressed. Static scheduling usually refers to the scheduling before job execution, as opposed to dynamic scheduling, where tasks are scheduled during job execution at runtime.
- Static scheduling is suitable for compilers to schedule the machine instructions in a program into parallel execution since the computation and communication cost can be calculated.

# Task Scheduling

- The scheduling problem was introduced as the spatial and temporal assignment of tasks to processors.
- The spatial assignment, or mapping, is the allocation of tasks to the processors.
  - A processor allocation A of the task graph G = (V,E,w,c) on a finite set P of processors is the processor allocation function proc: V => P of the nodes of G to the processors of P.
- The temporal assignment is the attribution of a start time to each task. However, it presupposes the allocation of the tasks to processors and therefore commonly both are defined by a schedule.

# Task Scheduling

- A schedule S of the task graph G = (V,E,w,c) on a finite set P of processors is the function pair ($t_s$,proc), where
  - $t_s$: is the start time function of the nodes in G.
  - proc: is the processor allocation of the nodes of G to the processors of P.

# With Communication Costs

- Target parallel system– classic model
  - A target parallel system P consists of a set of identical processors connected by a communication network.
  - Dedicated system. The parallel system is dedicated to the execution of the scheduled task graph. No other program or task is executed on the system while the scheduled task graph is executed.
  - Dedicated processor. A processor can execute only one task at a time and the execution is not preemptive.
  - Cost-free local communication. The cost of communication between tasks executed on the same processor is negligible and therefore considered zero.
  - Communication subsystem. Interprocessor communication is performed by a dedicated communication subsystem. The processors are not involved in communication.

# With Communication Costs

- Concurrent communication. Interprocessor communication in the system is performed concurrently; there is no contention for communication resources.
- Fully connected. The communication network is fully connected.

- Node finish time.
  - The finish time of a node is the node's start time plus its execution time (computation cost).
  - $t_f(n) = t_s(n) + w(n)$
- Edge finish time.
  - The time at which a communication arrives at the destination processor.
  - $t_f(e_{ij}, P_{src}, P_{dst}) = t_f(n_i, P_{src}) +$
    - $0$ if $P_{src} = P_{dst}$
    - $c(c_{ij})$ otherwise

# With Communication Costs

- Condition 1: exclusive processor allocation
  - $proc(n_i) = proc(n_j) \rightarrow$
    - $t_s(n_i) < t_f(n_i) <= t_s(n_j) < t_f(n_j)$
    - or $t_s(n_j) < t_f(n_j) <= t_s(n_i) < t_f(n_i)$
- Condition 2: precedence constraint
  - $t_s(n_j, P) >= t_f(e_{ij}, proc(n_i), P)$
- Feasible schedule
  - A schedule S is feasible if and only if all nodes n and edges e in the graph comply with conditions 1 and 2.

# With Communication Costs

- Data ready time
  - $t_{dr}(n_j, P) = \max \{t_f(e_{ij}, proc(n_i), P)\}$ for all $e_{*j}$
- Data ready time constraint
  - $t_s(n, P) >= t_{dr}(n,P)$
- Processor finish time
  - $t_f(P) = \max \{t_f(n)\}$ for all n where proc(n) = P
- Schedule length
  - $sl(S) = \max\{t_f(n)\}$ for all n in G
- Used processors
  - $Q = \cup proc(n)$ for all n in G
  - For any schedule S, |Q| <= |P|
- Sequential time
  - $Seq(G) = \sum w(n)$ for all n in G
  - G's execution time on one processor only.

# With Communication Costs

- Exercise:
  - A schedule example for the task graph in the previous exercise.

# Scheduling Complexity

- Scheduling problem
  - Let G = (V,E,w,c) be a task graph and P a parallel system. The scheduling problem is to determine a feasible schedule S of minimal length sl for G on P.
- The decision problem SCHED(G,P) associated with the scheduling problem is as follows.
  - Is there a schedule S for G on P with length sl(S) <= T
- SCHED(G,P) is NP-complete, even when |P| >= |V|
- $sl(S_{opt}(P_{+1})) <= sl(S_{opt}(P))$
  - on systems with P processors, but may using less processors in the schedule

# Without Communication Costs

- Target parallel system – cost-free communication
  - A target parallel system $P_{c0}$ consists of a set of identical processors connected by a cost-free communication network.
- Edge finish time
  - $t_f(e_{ij}, P_{src}, P_{dst}) = t_f(n_i)$
- Data ready time
  - $t_{dr}(n_j) = \max \{t_f(n_i)\}$ for all $n_i$ connecting to $n_j$
- Exercise:
  - A schedule example for the task graph in exercise 1.

# Scheduling Complexity

- SCHED-C0(G,$P_{c0}$) is NP-complete.
- While in general the scheduling problem without communication costs is NP-complete, it is solvable in polynomial time for an unlimited number of processors.
- A simple algorithm to find an optimal schedule is based on two ideas:
  - Each node is assigned to a distinct processor.
  - Each node starts execution as soon as possible.

# Scheduling Complexity

- Optimal scheduling algorithm:

Insert all n in G in topological order into sequential list L

for each $n_i$ in L do

    DRT = 0

    for each $n_j$ belonging to pred($n_i$) do

        DRT = max{DRT,$t_f(n_j)$}

    end for

    $t_s(n_i)$ = DRT

    $t_f(n_i) = t_s(n_i) + w(n_i)$

    proc($n_i$) = $P_i$

end for

# Scheduling Complexity

- $sl(S^{q+1}_{opt}) <= sl(S^{q}_{opt})$, schedules using exactly q+1 or q processors.
- The above relation is not valid when considering communication costs.
  - An example of chain structure can illustrate this.

# Task Graph Properties

- Path length, len(p)
  - The length of a path p in G is the sum of the weights of its nodes and edges.
- Computation length, $len_w(p)$
  - The sum of the weights of the nodes in a path
- Allocated path length, len(p,A)
  - The path length determined for a given processor allocation A.
- $Len(p) >= len(p,A) >= len_w(p)$
- Critical path
  - A critical path cp of a task graph G is a longest path in G.
- The critical path gains its importance for scheduling from the fact that its length is a lower bound for the schedule length.
  - $sl >= len_w(cp_w)$

# Task Graph Properties

- For cost-free communication and unlimited processors, $sl(S_{opt}) = len_w(cp_w)$

# Node Levels

- Let G=(V,E,w,c) be a task graph and n belong to V.
  - Bottom level bl(n) of n is the length of the longest path starting with n.
  - A path starting with n of length bl(n) is called a bottom path of n and denoted by $p_{bl(n)}$.
  - Top level tl(n) of n is the longest length path ending in n, excluding w(n).
  - A path ending in n of length tl(n) + w(n) is called a top path of n and denoted by $p_{tl(n)}$.
  - Computation bottom level $bl_w(n)$
  - Computation top level $tl_w(n)$
  - $p_{bl(n)} \neq p_{blw(n)}$ and $p_{tl(n)} \neq p_{tlw(n)}$

# Level Bounds on Start Time

- Let S be a schedule for task graph G=(V,E,w,c) on system P. For each n belonging to V,
  - $sl \geq t_s(n) + bl_w(n)$
  - $t_s(n) \geq tl_w(n)$
- $p_{tb(n)} = p_{tl(n)} \cup p_{bl(n)}$
- $len(p_{tb(n)}) = tl(n) + bl(n)$

# Critical Path Length and Node Levels

- Let G=(V,E,w,c) be a task graph. For any node $n_{cp,i}$ of a critical path cp
  - $len(cp) = tl(n_{cp,i}) + bl(n_{cp,i})$
- $bl(n_{src}) = len(p_{tb}(n_{src}))$
- $bl(n_{cp,1}) = len(cp) >= bl(n_i)$ for each $n_i$ in V
  - Consequently, a source node with the highest bottom level of all nodes is the first node $n_{cp,1}$ of a critical path cp of G.

# As-Soon/Late-as-Possible Start Times

- $ASAP(n) = tl(n)$
- $ALAP(n) = len(cp) – bl(n)$
- $ASAP_w(n) = tl_w(n)$
- $ALAP_w(n) = len_w(n) – bl_w(n)$

# Computing Levels and Critical Path

- To compute node levels, the following recursive definition of the levels is convenient. For a task graph G=(V,E,w,c) and $n_i$ belonging to V,

$$bl(n_i) = w(n_i) + \max_{n_j \in succ(n_i)} \{c(e_{ij}) + bl(n_j)\}$$

$$tl(n_i) = \max_{n_j \in pred(n_i)} \{tl(n_j) + w(n_j) + c(e_{ji})\}$$

# Computing Levels and Critical Path

- Algorithm: compute bottom levels

Insert n of V in inverse topological order into sequential list L.

for each $n_i$ in L do

max ← 0; $nbl_{succ}(n_i)$ ← NULL

for each $n_j$ in $succ(n_i)$ do

if $c(e_{ij})+bl(n_j)$ > max then

max ← $c(e_{ij})+bl(n_j)$; $nbl_{succ}(n_i)$ ← $n_j$

end if

$bl(n_i)$ ← $w(n_i)$ + max

end for

end for

# Computing Levels and Critical Path

- Algorithm: compute top levels

Insert n of V in topological order into sequential list L.

for each $n_i$ in L do

  max $\leftarrow$ 0; $ntl_{pred}(n_i) \leftarrow$ NULL

    for each $n_j$ in pred($n_i$) do

      if $tl(n_j)+w(n_j)+c(e_{ji}) >$ max then

        max $\leftarrow tl(n_j)+w(n_j)+c(e_{ji})$ ; $ntl_{pred}(n_i) \leftarrow n_j$

      end if

      $tl(n_i) \leftarrow$ max

    end for

end for

## Computing Levels and Critical Path

- Observe that the top and bottom paths are also computed with the presented algorithms.
- Moreover, a critical path and its length are also computed by the algorithm of computing bottom levels.
  - It suffices to store a node with the highest bottom level during the run of the algorithm.
- The paths of node levels and the critical path are in general not unique.

# Granularity

- Task graph granularity
  - Let G=(V,E,w,c) be a task graph. G's granularity is $g(G) = \frac{\min_{n \in V} w(n)}{\max_{e \in E} c(e)}$
  - A task graph is said to be coarse grained if g(G) >= 1
  - Coarse granularity is a desirable property of a task graph.
- One objective of task scheduling is always to minimize the cost of communication.
  - This is achieved by having as much local communication as possible.
  - Unfortunately, this objective conflicts with the other objective of scheduling, namely, the distribution of the tasks among the processors.
- Grain
  - Let G=(V,E,w,c) be a task graph. The grain of node $n_i$ in V is

$$grain(n_i) = \min\left\{ \frac{\min_{n_j \in pred(n_i)} w(n_j)}{\max_{e_{ji} \in E, n_j \in pred(n_i)} c(e_{ji})}, \frac{\min_{n_j \in succ(n_i)} w(n_j)}{\max_{e_{ji} \in E, n_j \in succ(n_i)} c(e_{ji})} \right\}$$

# Granularity

- Task graph weak granularity
  - Let G=(V,E,w,c) be a task graph. G's weak granularity is

$$g_{weak}(G) = \min_{n \in V,\, pred(n) \cup succ(n) \neq \Phi} grain(n)$$

  - This definition of granularity is called weak granularity because $g(G) <= g_{weak}(G)$

# Granularity and Critical Paths

$$g_{weak}(G) \leq grain(n_j) \leq \frac{w(n_i)}{c(e_{ij})}$$

- Relation between critical path and computation critical path
  - Let G=(V,E,w,c) be a task graph, cp its critical path, and $cp_w$ its computation critical path. The nodes of cp are denoted by $V_{cp}$, where $n_{last}$ is the last node of cp, and its edges by $E_{cp}$. It holds that

$$len(cp) \leq (1 + \frac{1}{g_{weak}(G)}) len_w(cp_w)$$

# Granularity and Critical Paths

- proof

$$len(cp) = \sum_{n \in V_{cp}} w(n) + \sum_{e_{ij} \in E_{cp}} c(e_{ij})$$

$$\le \sum_{n \in V_{cp}} w(n) + \sum_{e_{ij} \in E_{cp}} \frac{w(n_i)}{g_{weak}(G)} = \sum_{n \in V_{cp}} w(n) + \sum_{n \in V_{cp} - n_{last}} \frac{w(n_i)}{g_{weak}(G)}$$

$$\le \sum_{n \in V_{cp}} w(n) + \frac{1}{g_{weak}(G)} \sum_{n \in V_{cp}} w(n) = (1 + \frac{1}{g_{weak}(G)}) \sum_{n \in V_{cp}} w(n)$$

$$= (1 + \frac{1}{g_{weak}(G)}) len_w(cp) \le (1 + \frac{1}{g_{weak}(G)}) len_w(cp_w)$$

# Communication to Computation Ratio

- The measure of granularity considers extreme values and consequently guarantees certain properties of a task graph. However, the general scheduling behavior of a task graph is not necessarily related to the granularity of the graph.
- Let G=(V,E,w,c) be a task graph. G's communication to computation ratio is

$$CCR(G) = \frac{\sum_{e \in E} c(e)}{\sum_{n \in V} w(n)}$$

- Usually, a task graph is said to have high, medium, and low communication for CCRs of about 10, 1, and 0.1, respectively.

# Exercise

- For the task graph in exercise 1, determine the following:
  1. Granularity g(G)
  2. Weak granularity $g_{weak}(G)$
  3. Communication to computation ratio CCR(G)

# Fundamental Heuristics for Scheduling Task-Parallel Jobs

# Two Fundamental Heuristics

- List scheduling
- Clustering
- These two heuristics are classes or categories rather than simple algorithms. Most of the algorithms that have been proposed for task scheduling fall into one of these two classes.

## List Scheduling

- In its simplest form, the first part of list scheduling sorts the nodes of the task graph to be scheduled according to a priority scheme, while respecting the precedence constraints of the nodes—that is, the resulting node list is in topological order.
- In the second part, each node of the list is successively scheduled to a processor chosen for the node.
  - Usually, the chosen processor is the one that allows the earliest start time of the node.

# List Scheduling

- Algorithm: simple list scheduling—static priorities (G=(V,E,w,c), P)

  1 Part:

  Sort nodes in V into list L, according to priority scheme and precedence constraints.

  2. Parts:

  for each n in L do

  Choose a processor in P for n

  Schedule n on P

  end for

- Each node is only scheduled once, that is, the start time and the allocated processor are never changed in a latter step of the algorithm. The partial schedules must be feasible in order to achieve a feasible final schedule.

# List Scheduling

- Free node
  - Let G=(V,E,w,c) be a task graph, P a parallel system, and $S_{cur}$ a partial feasible schedule for a subset of nodes $V_{cur}$, included in V, on P. A node n in V is said to be free if n is not in $V_{cur}$ and ance(n) is included in $V_{cur}$.
- In list scheduling, every node to be scheduled is free, because the nodes are processed in precedence order. Hence, by definition, at the time a node is scheduled all ancestor nodes have already been processed.

# List Scheduling

- End technique
  - Let G=(V,E,w,c) be a task graph, P a parallel system, and $S_{cur}$ a partial feasible schedule for a subset of nodes $V_{cur}$, included in V, on P. The start time of the free node n in V, on a given processor P, is determined by

    $$t_s(n,P) = \max\{t_{dr}(n,P), t_f(P)\}$$

- This determination of the start time is here called “end technique”, as node n is scheduled at the end of all other nodes scheduled on processor P.

# Start Time Minimization

- Algorithm: schedule free node n on Earliest-Start-Time Processor

  Require: n is a free node

  $t_{min} \leftarrow$ infinity; $P_{min} \leftarrow$ NULL

  for each $P$ in P do

      if $t_{min} > \max\{t_{dr}(n,P), t_f(P)\}$ then

          $t_{min} \leftarrow \max\{t_{dr}(n,P), t_f(P)\}$; $P_{min} \leftarrow P$

      end if

  end for

  $t_s(n) \leftarrow t_{min}$; proc(n)$\leftarrow P_{min}$

- In the literature, list scheduling usually implies the above start time minimization method.
- An Example

# Online Scheduling of Workflow Applications in Cloud Environment

# Introduction

- Cloud environments are an important platform for running high-performance and distributed applications. Many large-scale scientific applications are usually constructed as workflows due to large amounts of interrelated computation and communication,
  - e.g., Montage and EMAN.
- Scheduling workflow applications in parallel systems is a great challenge.
  - It is an NP-complete problem.

# Introduction (Cont.)

- Many heuristic methods have been proposed in the literature
  - Most of them deal with a single workflow at a time.
- In recent years, there are several heuristic methods proposed to deal with concurrent workflows or online workflows
  - They do not work with workflows composed of data-parallel tasks.
- In the following, we present an online scheduling approach for mixed-parallel workflows in cloud environments.

# Introduction (Cont.)

- The proposed approach was evaluated with a series of simulation experiments.
  - We developed a simulator using discrete-event based techniques for experiments.
  - A workflow is represented by direct acyclic graph (DAG).
  - The cloud environment is assumed to consist of several dispersed clusters, each containing a specific amount of processors.
  - The results show that the proposed approach delivers good performance under various workloads.

# Related Work

- In the past years, most works dealing with workflow scheduling were restricted to single workflow application.
- Recently, Zhao et al. in their work envisaged a scenario that need to schedule multiple workflow applications at the same time. They proposed two approaches:
  - The composition approach merges multiple workflows into a single workflow first. Then, list scheduling heuristic methods, such as HEFT, can be used to schedule the merged workflow.
  - The main idea of the fairness approach is that when a task completes, it will re-calculate the slowdown value of each workflow against other workflows and make a decision on which workflow should be considered next.

# Related Work (Cont.)

- The composition and the fairness approaches are static algorithms and not feasible to deal with online workflow applications,
  - *i.e.* multiple workflows come at different time instants.
- Later, RANK_HYBD is proposed to deal with online workflow applications submitted by different users at different times. The task scheduling approach of RANK_HYBD sorts the tasks in *waiting queue* using the following rules repeatedly.

# Related Work (Cont.)

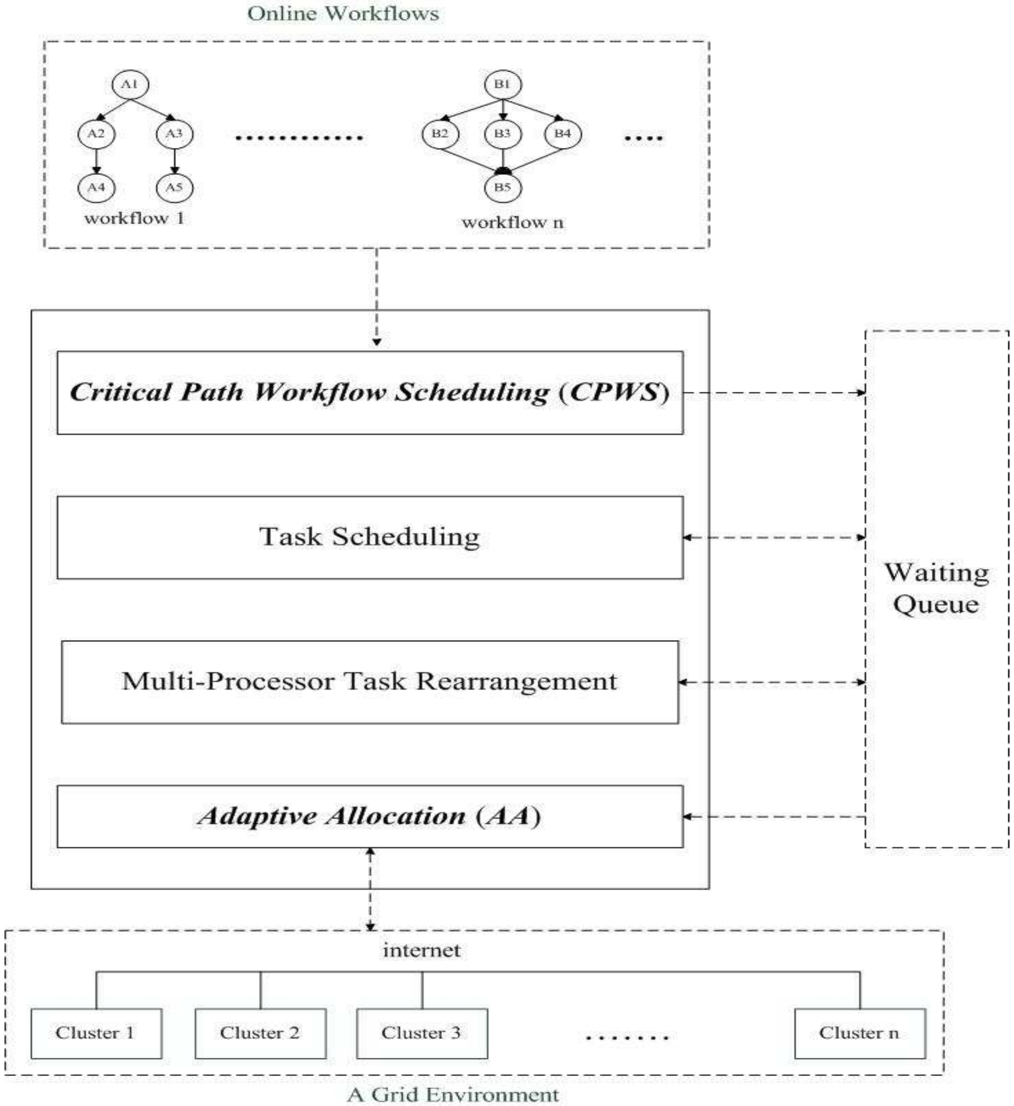
  - If tasks in *waiting queue* come from multiple workflows, the tasks are sorted in ascending order of their rank value ($rank_u$) where $rank_u$ has the same definition as in HEFT;
  - If all tasks belong to the same workflow, the tasks are sorted in descending order of their rank value ($rank_u$).

- However, in the above approaches, the number of processors to be used by each task is limited to a single processor. It is not feasible to deal with workflows composed of data-parallel tasks.

# Related Work (Cont.)

- N'takpe' et al. proposed a scheduling approach for mixed parallel applications on Heterogeneous platforms.
  - However, their approach is restricted to concurrent workflows submitted at the same time. It is infeasible to deal with online workflows submitted at different time instants.
- The OWM proposed in the following is designed to deal with multiple online mixed-parallel workflows that previous methods cannot handle well.

# Online Workflow Management

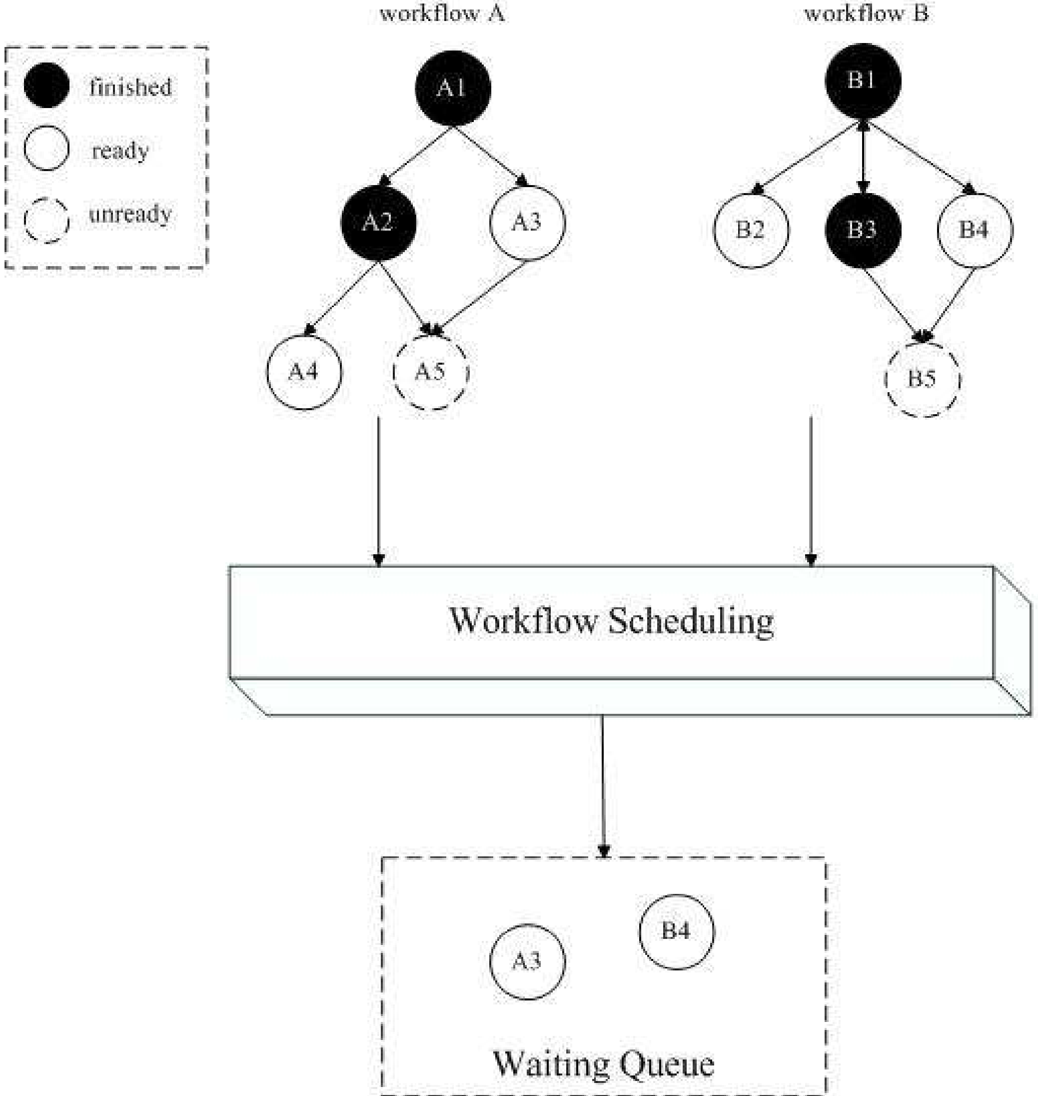


# Online Workflow Management (Cont.)

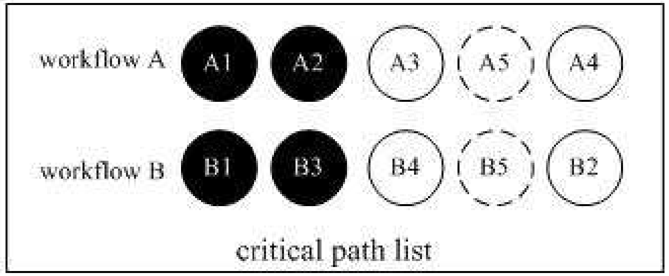
- In OWM, there are four steps:
  - Critical Path Workflow Scheduling (CPWS). When a new workflow arrives, *CPWS* is adopted to calculate $rank_u$ of each task in the workflow and sort the tasks in descending order of $rank_u$ into a list. During the workflow's execution, according to the order in each critical path list, *CPWS* continuously submits the ready tasks in the list into the waiting queue until running into an unready task.

$$rank_u(t_i) = \overline{w_i} + \max_{t_j \in succ(t_i)}(\overline{c_{i,j}} + rank_u(t_j))$$

# An example of CPWS




| workflow | Rank | | | |
|---|---|---|---|---|
| workflow A | A1 | 80 | A4 | 15 |
| | A2 | 72 | A5 | 25 |
| | A3 | 43 | | |
| workflow B | B1 | 120 | B4 | 65 |
| | B2 | 20 | B5 | 47 |
| | B3 | 98 | | |



# Online Workflow Management (Cont.)

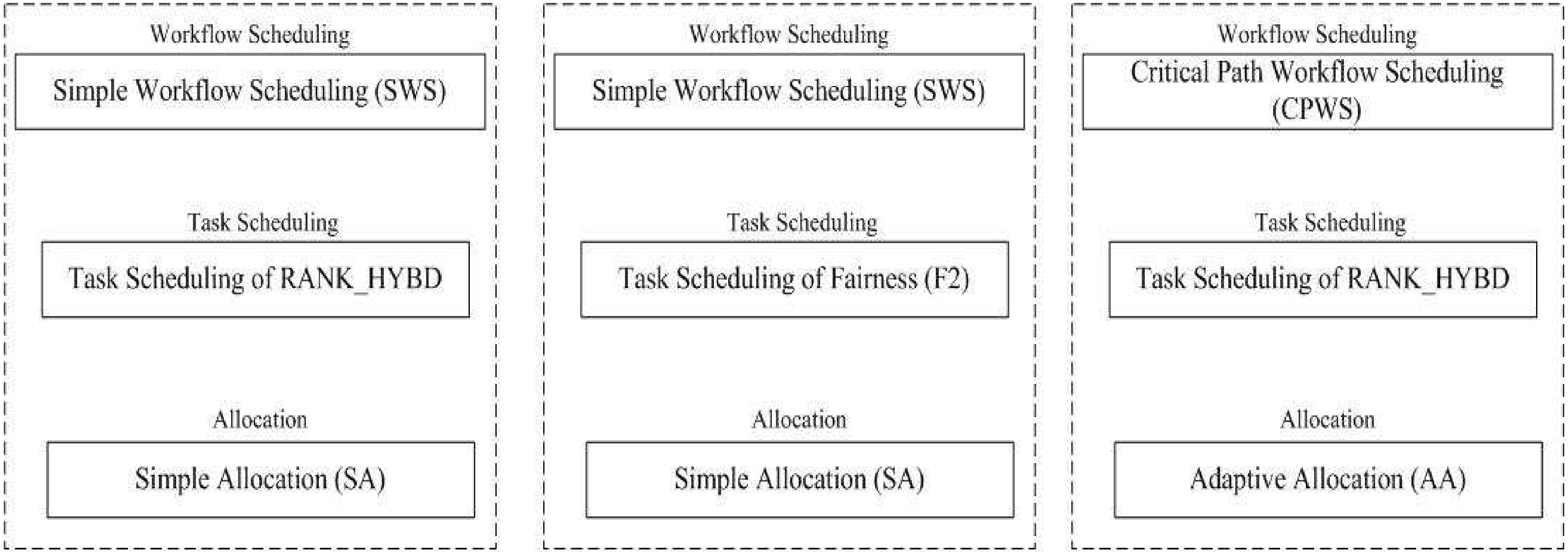
- Task Scheduling. This step adopts the RANK_HYBD method.
- Multi-processor task rearrangement. It improves processor utilization by applying techniques such as first fit, easy backfilling, and conservative backfilling scheduling approaches.

# Online Workflow Management (Cont.)

- Adaptive Allocation (AA). When the number of clusters that can accommodate the first task in queue is 1, it first finds the cluster with the earliest estimated available time among other clusters. If the estimated finish time of the first task on that cluster is earlier than that on the free cluster, the task will be kept in the waiting queue. Otherwise, the system allocates the task to the free cluster right away.

# Experimental Results

- The performance metrics used in the following experiments include:
  - Makespan. The time between submission and completion of a workflow.
  - Schedule Length Ratio (SLR). $SLR = \frac{makespan}{CPL}$
  - win (%). The win value of an algorithm means the percentage of the workflows that have the shortest makespan when applying this algorithm.
- In the following experiments, we compare OWM with two other approaches: *RANK_HYBD* and *Fairness_Dynamic*.



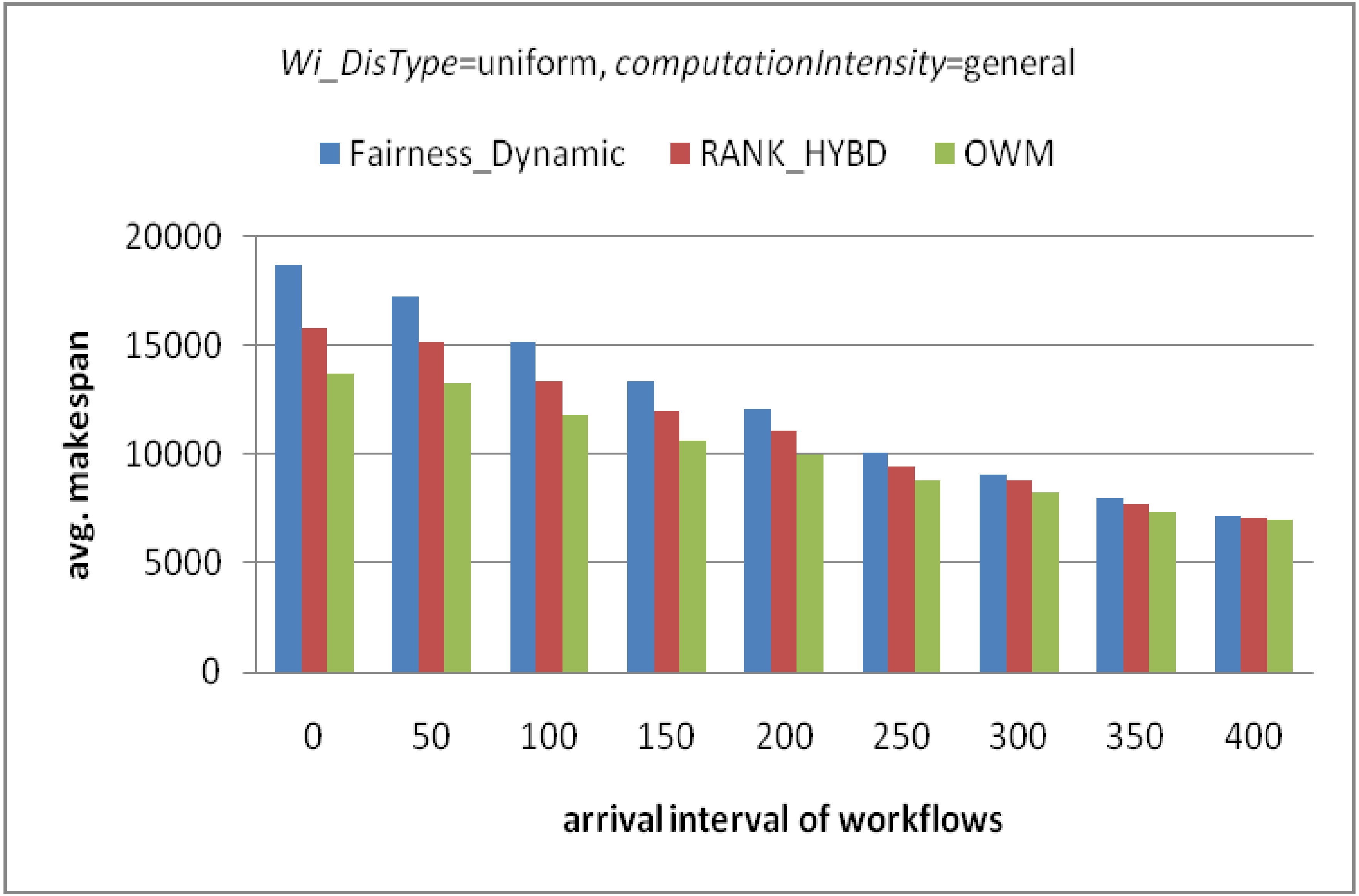

(a) RANK_HYBD (b) Fairness_Dynamic (c) OWM

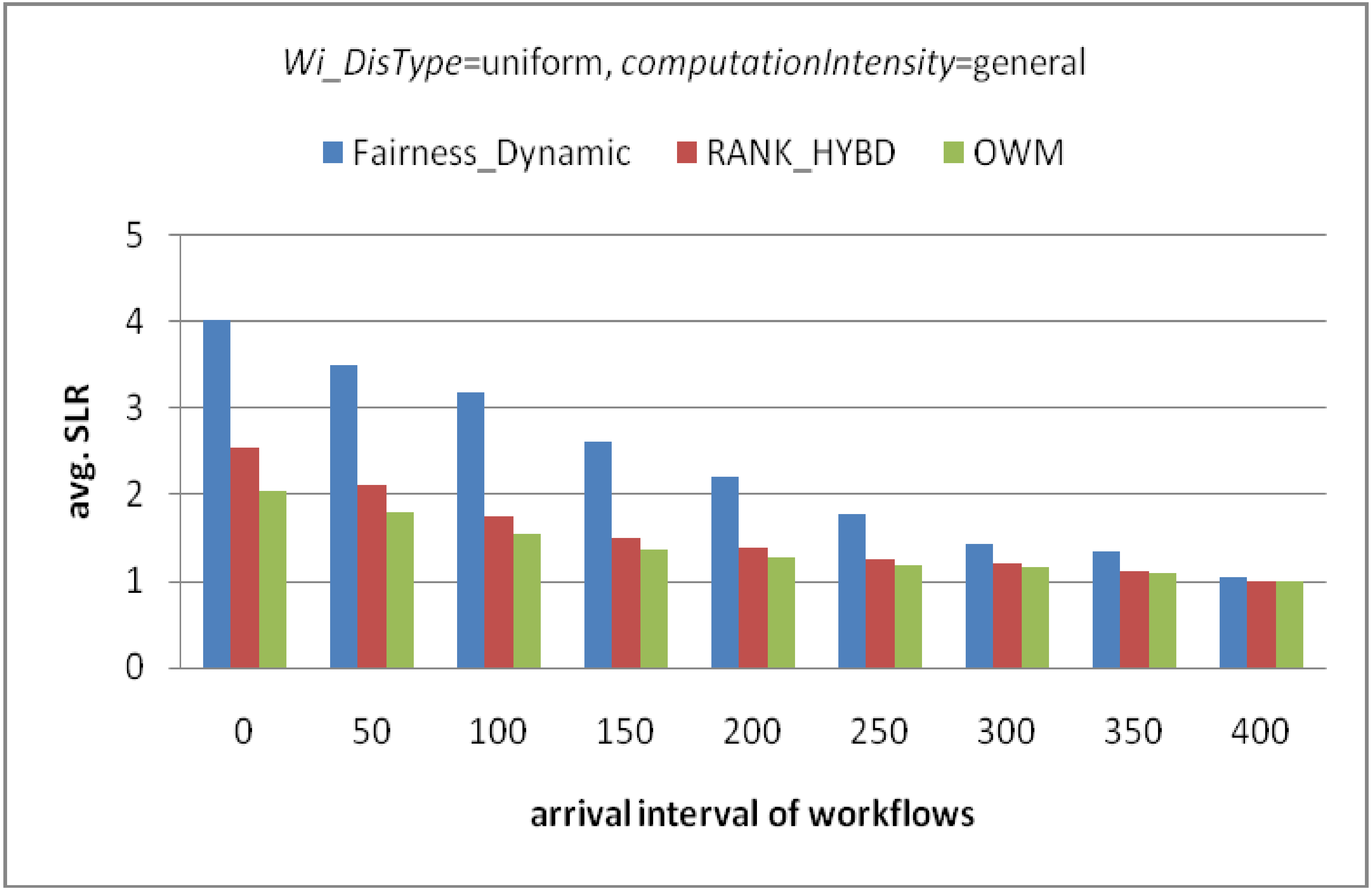
# Experimental Results (Cont.)

- To experiment with different workload characteristics, we use the following parameters to generate different types of workflows. A workflow is represented as a Directed Acyclic Graph (DAG).
  - Node={20, 40, 60, 80, 100}
  - Shape={0.5, 1.0, 2.0}
  - OutDegree={1, 2, 3, 4, 5}
  - CCR={0.1, 0.5, 1.0, 1.5, 2.0}
  - BRange={0.1, 0.25, 0.5, 0.75, 1.0}
  - WDAG=100~1000
  - [7] Topcuoglu, H., Hariri, S., and Wu, M. Y., "*Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing*". IEEE Transactions on Parallel and Distributed Systems, 2(13):260-247, 2002.

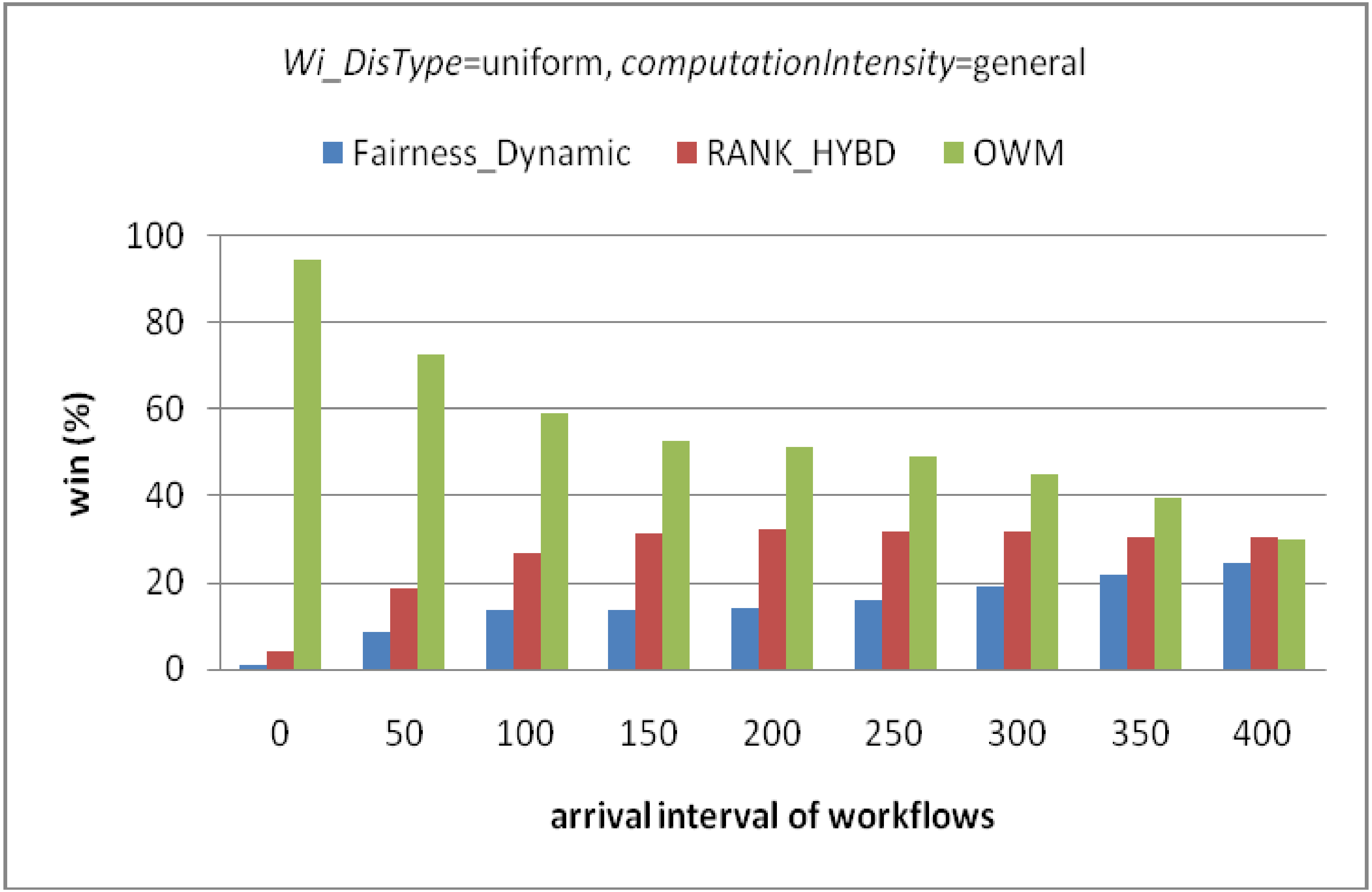
# Experimental Results (Cont.)

- The values of the parameters are randomly selected from the corresponding sets given above for each DAG. The arrival interval value between DAGs is set based on Poisson distribution. Each experiment involves 20 runs, and each run has 100 unique DAGs in a grid environment that contains 3 clusters each containing 30~50 processors respectively.
- We experimented with both a uniform distribution and an exponential distribution for tasks' computation cost.


Wi_DisType=uniform, computationIntensity=general
Fairness_Dynamic
RANK_HYBD
OWM
20000
15000
10000
5000
0
avg. makespan
0
50
100
150
200
250
300
350
400
arrival interval of workflows



Wi_DisType=uniform, computationIntensity=general
Fairness_Dynamic
RANK_HYBD
OWM
avg. SLR
0
1
2
3
4
5
0
50
100
150
200
250
300
350
400
arrival interval of workflows



Wi_DisType=uniform, computationIntensity=general
Fairness_Dynamic
RANK_HYBD
OWM
100
80
60
40
20
0
win (%)
0
50
100
150
200
250
300
350
400
arrival interval of workflows

# Summary

- Most existing workflow scheduling algorithms are restricted to handle only one single workflow. There are few researches for scheduling multiple or online workflows. In the above, we propose an online workflow management (***OWM)*** approach for scheduling multiple online mixed-parallel workflows in a grid environment.
- Our experiments show that ***OWM*** outperforms other methods in terms of average makesapn, average SLR and win (%) under different workloads.

# Thank You!