Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Individualized precise scheduling strategy based on program's runtime characteristic for workload consolidation

Lin Wang
Shandong Normal University
Tianyuan Huang
Shandong Normal University
Shichao Geng (gengsc@hotmail.com)
Shandong Normal University

Article

Keywords: Runtime Characteristic, Individualized Precise Scheduling, Workload Consolidation, Clusters or Data Centers

Posted Date: July 27th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-1867897/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Abstract

In data centers, workload consolidation is the common method to improve resource utilization. However, efficient workload consolidation faces challenges from two aspects: the first is the wide variety of program characteristics; the second is the multiple performance indexes in clusters or data centers, such as the program performance requirement from custom and the system utilization from the cluster provider. In this paper, multiple experiments are executed about program performance with restricted resource usage, and four conclusions are summarized. Then based on the conclusions, two individualized precise scheduling strategies are proposed for efficient consolidation. Sche-index2 schedule programs by considering the resource sensitivity hyper-plane on five resource dimensions, and Sche-utility scheduling programs by considering the sensitivity curve of five resource dimensions. The experiment results show that compared with the common scheduling strategy, Sche-index2 can improve the average performance of the program by 36%, and Sche-utility can improve the average performance of the program by 27.6%. Meanwhile, both of the scheduling strategies can improve the resource utilization of CPU, disk read/write bandwidth, memory, and network bandwidth.

1. Introduction

Workload consolidation is the common method to improve resource utilization in clusters or data centers [1, 2, 3]. In order to consolidate programs efficiently, scheduling programs to appropriate execution environment plays an important role [4]. There are some requirements for this kind of scheduling strategy: (1) the performance of programs should be guaranteed, without dramatic performance dropdown [5]; (2) server utilization should be improved, at the same time the resource congestion should be avoided [6]; (3) either over-provision or under-provision is the suboptimal plan, proper provision should be the best plan [7].

Meanwhile, programs characteristic in cluster and data centers are varied [8, 9, 10]. For example, programs accessing the database are disk I/O-intensive programs; programs caching data in memory are memory-intensive programs; programs analyzing images by data mining are CPU-intensive programs; programs playing as a streaming server are network-intensive programs, and many programs are the mix of these resource-intensive types. Moreover, when the used resource is occupied by other programs, the degree of performance dropdown of different programs is different, the performance dropdown of some programs is big, and the performance dropdown of other programs is small [11]. So, when consolidating programs, in order to map all kinds of programs to an appropriate execution environment, all the resource usage of programs should be considered, even more, the degree of program's performance dropdown when the used resource is occupied by other programs should be considered.

In our past work, we propose a novel index system to support efficient consolidation [12]. The novel index system classifies the program by co-located program runtime characteristics, then labels the programs for scheduling. By the index system, the program can be divided by the following method: (1) dependence on multi-dimensional resources including CPU, disk I/O, memory and network I/O; and (2) impact and

vulnerability to resource sharing which can be embodied by resource usage and resource sensitivity. These labels can help in efficient workload consolidation.

In this paper, with the aim of efficient workload consolidation, we do multiple experiments about program performance with restricted resource usage, and get multiple conclusions. Some of the conclusions are in accordance with expectations, such as different resource does different contribute to a program. But some of the conclusions are unexpected, such as the resource usage of a program is different from the resource sensitivity of the program. Then based on our observation, we propose two individualized precise scheduling strategies to support efficient consolidation. The first strategy scheduling programs by considering the resource sensitivity hyper-plane on five resource dimensions, and the second strategy scheduling programs by considering the sensitivity curve of five resource dimensions. Both of the two scheduling strategies can improve program performance, resource utilization, and overall system throughput. The experiment results show that compared with the common scheduling strategy, Sche-index2 can improve the average performance of the program by 36%, and Sche-utility can improve the average performance of the program by 27.6%. Meanwhile, both of the two scheduling strategies can improve the resource utilization of CPU, disk read bandwidth, disk write bandwidth, memory and network bandwidth.

The rest of the paper is organized as follows. Section 2 introduces the background knowledge of the multi-dimension utility combination scheduling strategy. Multi-dimension utility combination. Multiple experiments are executed and several conclusions are summarized in Section 3, and two individualized precise scheduling strategies are proposed in Section 4. The experiment method is described in Section 5 and the experiment results are analyzed in Section 6. Related work is discussed in Section 7, and conclusions are summarized in Section 8.

2. Background

Utility: in economics, utility refers to goods or services to meet people's desires or needs [13]. It is similar to our observation in section 3, for different programs, the same resource can do different contributions to performance. If the results of the feasible solution have a variety of possible results value o, then $\mathrm{u} = \mathrm{u}\left(\mathrm{o}\right)$ is the value of the result for the decision-maker, we call it utility. In a decision problem, in general, the maximum return value of utility is defined as 1, and the smallest return value of utility is defined as 0, utility values are between $\left[0,1\right]$. In a plane rectangular coordinate system, if the abscissa represents return value; the ordinate represents utility value, the attitude of decision-maker to return value can be plotted as a curve, this curve called the utility curve of decision-maker, as is shown in Fig. 1. Figure 1 describes the utility curve of two different programs for the memory, when increasing the size of the occupied memory, the program’s performance is improved; when the program’s performance reached the highest, the utility is 1. Different programs have different utility curve, program Data Caching almost reach its highest performance when memory size is 4G, while ft.C.8 reach its highest performance when memory size is 8G.

Multi-dimension utility combination: A remarkable feature of the multi-objectives problem is commensurability between the objectives. The multi-dimension utility combination [14] can solve this problem. Assumption that multi-objectives problem has s number of criterions for evaluation, and has m feasible scheme, then the utility function of criterion for evaluation are $u_1, u_2, \cdots, u_s$, and the sub-utility value of m feasible scheme under criterion evaluation are

$$u_1(a_i), u_2(a_i), \cdots, u_s(a_i), (i = 1,2, \cdots, m)$$

In order to describe the utility of feasible scheme a_i on the whole, a certain method should be used to combine sub-utility to the utility. Then reorder all feasible schemes based on utility value. The feasible scheme with the biggest utility value is the appropriate scheme, which is called multi-dimension utility combination.

3. Observation

In this section, we introduce our observations based on a large number of experiments about program performance under different resource restrictions. These observations can help individualized precise scheduling. The experiment machine is equipped with two Intel Xeon E5506 (Westmere) processors and 8G of memory. These processors include four physical out-of-order cores, and the disk read bandwidth is 137 MB/s, the disk write bandwidth is 120 MB/s, the network bandwidth is 100 MBit/s. The server runs CentOS 6.5 with Linux kernel version 2.6.32. The programs are selected from NAS Parallel Benchmark [15], PARSEC [16], SysBench [17] and Cloudsuite [18], and the resources are restricted by Cgroups [19]. The following observations are obtained:

Observation 1: Different resources do different contribute to one program. Some resource is important for the program, but the other resource maybe not. An example is in Fig. 1, defining the program's performance of running alone as the baseline, when increasing the CPU capacity from 2 to 8, the performance of Data Caching only increases 7.5%; when increasing memory capacity from 2G to 8G, the performance of Data Caching increase almost 100%; when increasing network capacity from 20Mb to 100Mb, the performance of Data Caching increase almost 100%. Moreover, when memory usage of Data Caching is 4G, Data Caching's performance is 91.2% of the baseline, after that go on increasing memory capacity, the performance improves just a little. It can be concluded that CPU capacity does just a little contribution to the performance of Data Caching; memory capacity is the important resource for the performance of Data Caching, but when the memory capacity increasing to a threshold, the performance of Data Caching is no longer improved; Network capacity is the important resource for the performance of Data Caching, and with network capacity increasing, the performance of Data Caching is always improved.

Observation 2: The resource contribution to different program is different, some resource is important for program A, but the resource is not working for program B. Figure 2 describes resource's (CPU, memory, network) contribution to program mg.C.8, facesim and Data Caching. Among them, increasing CPU

capacity can largely improve the performance of mg.C.8, but cannot improve the performance of facesim and Data Caching; increasing memory capacity can largely improve the performance of mg.C.8 and Data Caching, but cannot improve the performance of facesim; increasing network capacity can largely improve the performance Data Caching, but cannot improve the performance of mg.C.8 and facesim.

Observation 3: The resource usage of the program is different from the resource sensitivity of the program. Defining the resource usage of the program as the resource consumed when the program running alone on a server, and defining the resource sensitivity of the program as the performance degradation of the program when the required resources are being occupied by other programs. The resource sensitivity of a program under novel index system is a sensitivity hyper-plane, and the resource sensitivity of program on every resource dimension is a sensitivity curve, such as the sensitivity curve in Fig. 1, the sensitivity curve can be described as a vector. However, the resource usage of the program on every dimension is a value. In order to compare the resource sensitivity with the resource usage of the program, a suitable point from the sensitivity vector needs to choose to delegate the sensitivity on this resources dimension. The lower boundary point is selected to delegate the sensitivity vector. The reason is that the sensitivity characteristic is varied with different programs, and the sensitivity value on the upper boundary point of the sensitivity vector is 1. Suppose reducing the restricted resource, if the sensitivity value is always close to 1, indicating that the program is not sensitive to the resource. If the resource usage of the program is cut down, the performance of the program would drop down a little. Conversely, if the sensitivity value decreases to 0 when reducing the restricted resource, indicating that the program is sensitive to the resource. Otherwise, if the resource usage of program is cut down, the performance of the program would drop down dramatically. Therefore, the sensitivity value on the lower boundary point can be used to delegate the range of performance dropdown when the resource is reduced. Figure 3 shows, in accordance with expectations, the value of resource usage inversely with the value of resource sensitivity, which means when the resource usage of the program is big, the program is sensitive; when the resource usage of the program is small, the program is insensitive, such as the situation of CPU in (a), (b), (d), (e). But conflict with common expectation, the value of resource usage does not have a complete correlation with the value of resource sensitivity, such as in (c) and (f), two programs have the same CPU usage, but they have different CPU sensitivity. Another example is in (d), network bandwidth usage of Data Caching is little, but the network bandwidth sensitivity of Data Caching is also little.

Observation 4: Commonly reducing the memory capacity for a mem-sensitive program is fatal to program performance, but reduce the CPU capacity for a CPU-sensitive program is impressionable to the program performance, and reduce the network bandwidth capacity for a net-sensitive program is impressionable to the program performance. In Fig. 4 (a) and (c), when reducing CPU capacity or network bandwidth capacity of programs, the performance of CPU-intensive or net-intensive programs are drop-down gradually; but when reducing memory capacity of programs, the performance of memory-intensive programs drops down dramatically from one restriction node to next restriction node. It gives the following revelation: the idea of resource consolidation is to make a trade-off between program performance and resource utilization, which is running more programs in the cluster with the pay of

pulling down the performance of the programs. When consolidating programs, reducing the resource usage of CPU, disk I/O and network I/O commonly results in the program's performance decrease; but reducing the resource usage of memory commonly results in performance dropdown dramatically.

4. Individualized Precise Scheduling Strategy

We propose two individualized precise scheduling strategies to support efficient consolidation. When a new program needs to be scheduled, the schedulers find an appropriate node for the program according to the future knowledge of programs. The future knowledge of these scheduling strategies are all from the label of the novel index system. The first scheduling strategy takes the resource sensitivity hyper-plane as the future knowledge, and we call it Sche-index2 for short; The second scheduling strategy takes the sensitivity curve of five dimensions as future knowledge, and we call it Sche-utility for short. At last, the program can be mapped to the most appropriate node for running. Both methods can improve program performance, resource utilization, and throughput.

4.1. Scheduling Strategy Based on Resource Sensitivity Hyper-plane

Sche-index2 should not only guarantee the performance of the mapped program, but also guarantee the performance of the programs already running on servers. With this goal, Sche-index2 should consist of three components below:

1) Profiler. The mission of the profiler is to collect the information of idle resource on five dimensions, and report them to the predictor periodically. By making use of common profiling tools such as collectl, the resource usage of CPU, Disk r/w bandwidth, memory and network bandwidth can be easily collected, the idle resource can be calculated by the overall resource minor the resource has been used.

2) Predictor. The primary objective of the predictor is to predict the program performance if the program was mapped onto the server. Meanwhile, it has to predict the performance of programs already running on the server. For the new arrival program, the performance can be calculated by Eq. 1, Serveridle[s][C], Serveridle[s][Dr], Serveridle[s][Dw], Serveridle[s][M], Serveridle[s][N] are the idle resource of CPU, Disk r/w bandwidth, memory and network bandwidth, and they are collected by profiler. resource sensitivity is a function to calculate the program performance under the assumption of these idle resources. Under the novel index system, the sensitivity labeled to the program is a sensitivity hyper-plane on five resource dimensions. The mechanism of the resource sensitivity function is to search the nearest point to the idle resource and take the sensitivity value of the point as Prog_perf. Prog_perf is the predicted performance. For the programs already running on the server, the performance can also be calculated by Eq. 1, but the idle resource is the overall resource of the server minor the resource usage of other programs (including the new arrival program).

$$Prog\_perf = resource\_sensitivity[Serveridle[s][C], Serveridle[s][Dr], Serveridle[s][Dw], Serveridle[s][M], Serveridle[s][N]] \quad (1)$$

3) Scheduler. The scheduler maps programs to an appropriate server. If the performance value of the program in server s predicted is lower than a threshold, the server s is dropped and the next server is to be checked; If the performance value of all programs in server s predicted is bigger than the threshold, the server s is the object server. Then the scheduler maps the new arrival program to server s.

The pseudo-code of this scheduling strategy is shown in Algorithm 1. For a program to be scheduled, the server selection process is carried out iteratively over each server. For the first checked server s, the available resource is collected, and the sensitivity of a program on the checked server is obtained by accessing the 5-dimensional array in the program label using the available resource levels of server s as the indices. If the sensitivity value is bigger than a threshold, then check the sensitivity of other programs running on server s. If the sensitivity value of all the programs running on server s is bigger than a threshold, then take server s as the target server, and then the program is scheduled to that server for execution.

Algorithm 1. Mapping programs based on resource sensitivity hyper-plane

```
1: //The resource label is a two-tuple (resource usage, resource sensitivity)
2: //Program resource sensitivity id 5-dimensional array (defined as in previous paper) with each element a sensitivity value at a specific resources combination
3: //Threshold_sens is the threshold of sensitivity
4: //Available_resource is a 5-element vector keeping the available resource of the current server
5:   for s=1 to max_server do
6:       Available_resource=the available resources on server[s];
7:       Prog_sens=resource sensitivity [Serveridle[s][C], Serverid-le[s][Dr], Serveridle[s][Dw],Serveridle[s][M],Serveridle[s][N]];
8:        if Prog_sens>Threshold_sens then
9:           find all the programs running on server s and calculate the resource sensitivity of these programs
10:          if all the sensitivity value > Threshold_sens then
11:              take server s as the object server;
12:              update the idle resource of server s;
13:          end if
14:       end if
15:       break;
16:  end for
```

Actually, for different workloads, they have their own Equation 1. We had tried to synthetic the resource_sensitivity into a formula, but at last we gave up because of the relatively big inaccuracy. We used the following method to get the data of Prog_perf: we had a database about the program's resource sensitivity hyper-plane, the sensitivity data from the database were from our multiple experiment record. when we got the Available_resource of C, Dr, Dw, M and N, we found the nearest point of the Available_resource in the database. Then the sensitivity data could be used as Prog_perf.

4.2. Scheduling Strategy Based on multi-dimension utility combination

We choose the multi-dimension utility combination to realize our second scheduling strategy because they have very similar characteristics. Every resource dimension has an impact on the program's performance, and the whole resource dimensions have an impact on the program's performance. This fits perfectly with the multi-dimension utility combination. However, there is indeed an inaccuracy between the sub-utility combination with the utility. Sche-utility is consist of three components:

1) Profiler. the profiler of the Sche-utility is the same as the profiler of the Sche-index2.

2) Predictor. Sche-utility uses multi-dimension utility combination to predict the program performance if the program was mapped onto the server. For the new arrival program, the performance can be calculated by Eq. 2, UC, UDr, UDw, UM, UN are the sub-utility function of five resources, and u is the utility value of these available resources, it is also the performance value predicted by multi-dimension utility combination.

$$u = U_c(\text{Serveridle}[s][C]) * U_{Dr}(\text{Serveridle}[s][Dr]) * U_{Dw}(\text{Serveridle}[s][Dw]) * U_M(\text{Serveridle}[s][M]) * U_N(\text{Serveridle}[s][N]) \quad (2)$$

The formula is valid in theory because it directly makes use of the multidimensional utility merge model. The performance of the program on one resource dimension can be taken as a sub-utility, then there are five sub-utility. In order to describe the utility of program's performance as a whole, we combine sub-utility to the utility, then the utility is the performance value of the program with the five-resource restriction.

3) Scheduler. The scheduler of the Sche-utility is the same as the scheduler of the Sche-index2.

The pseudo-code of this scheduling strategy is shown in Algorithm 2. For the first checked server s, the available resource is collected, by which the sub utility value of CPU, disk read bandwidth, disk write bandwidth, memory and network bandwidth can be calculated by the sub utility function. The utility value is the product of the five sub utility values. If the utility value is bigger than the utility threshold, then take server s as the target server.

Algorithm 2. Mapping programs based on multi-dimension utility

```
1: //Threshold_utility is the threshold of utility value
2: //Available_resource is a 5-element vector keeping the available resource of the current server
3: for s=1 to max_server do
4:     Available_resource=the available resources on server[s];
5:     utility[s]=1;
6:     for i=1 to 5 do
7:         sub_utility[i]=sub-utility-funi [Available_resource];
8:           utility[s]=utility[s]*sub_utility[i];
9:     end for
10:    if utility[s]>Threshold_utility then
11:        take server s as the object server;
12:        update the idle resource of server s;
13:    end if
14:    break;
15: end for
```

The advantage of these two scheduling strategies is that they are more intelligent than the common least load scheduling strategies. In section 3, we delved into the resource sensitivity of the program and got four observations. The two scheduling strategies proposed in this section take full use of the program's resources sensitivity. That is, the two scheduling strategies use the conclusions from the previous observations to allocate resources. So, the two scheduling strategies will get better performance in theory. Meanwhile, Sche-index2 takes the program's resource sensitivity hyper-plane into consideration and Sche-utility takes the impact of every resource into consideration. Because there is an inaccuracy between the sub-utility combination with the utility, the performance of Sche-utility wouldn't catch up with Sche-index2 in theory.

5. Methodology

5.1. Hardware & Software Setup

We use a cluster composed of 4 servers. The server for experiments is equipped with two Intel Xeon E5506 (Westmere) processors. Our server runs CentOS 6.5 with Linux kernel version 2.6.32.

5.2. Programs Selection and Setup

Cluster or data centers have a great variety of programs, in order to guarantee authenticity and diversity, we choose programs from multiple benchmarks. We choose Data Caching, Data Serving, Media Streaming from Cloudsuite, these scale-out programs are different from other parallel programs such as PARSEC. Programs from Cloudsuite represent th typical workload of the data centers and have more complicated resource demand patterns [20], therefore they are of urgent need for our index system to promote program's performance, resource utilization and system throughput. We also increase the heterogeneity of the workload to include the programs from the NAS Parallel Benchmark and SysBench. Different programs pressure different kind of resource in node, and for different programs, the performance measurement method is different, for program Data Caching, the performance standard is average latency; for program Data Serving, the performance standard is operation per second; for programs in NAS Parallel Benchmark and PARSEC, the performance standard is instruction per cycle.

The purpose of consolidation is to run programs on fewer node, so the experiment scene in our paper needs a relatively large number of programs. We run 14 programs on 4 nodes, and in order to obtain program performance all the time with interference, we restart programs after they accomplish. Limited to the experiment resource, we use only 4 nodes. But the purpose of our algorithm is to find the execution environment that satisfies the performance requirements of the program. For the clusters in the data centers, our algorithms show good expansibility.

5.3. Compared Scheduling Strategy

we compare three scheduling strategies: first, the common least load scheduling strategy, which scheduling programs by considering CPU usage and memory usage, people call it least loaded scheduling (ll-2 for short). ll-2 scheduling strategy is a classical scheduling policy, and many researchers [21] compare their own scheduling policy with it. So ll-2 can be used as a baseline for new proposed scheduling; the second scheduling strategy not only consider the resource usage of CPU, disk read bandwidth, disk write bandwidth, memory and network bandwidth, but also consider resource sensitivity of them, that is Sche-index2; the third scheduling strategy use multi-dimension utility combination to find out the appropriate execution environment, that is Sche-utility. The experiment first discusses the effect of the different threshold values to program performance with scheduling strategies Sche-index2 and Sche-utility. Then set the threshold as 0.8 with Sche-index2 and Sche-utility, and compare with scheduling strategy ll-2. In order to guarantee the accurate, our experiments execute multiple times and calculate the mean value.

6. Evaluation

Figure 5 shows program performance with different threshold ε with scheduling strategy Sche-index2. The performance is represented as the relative performance compared to the one achieved when the program is running alone on a server. We define the programs' performance when running alone as 1. When the value of ε is equal to 0.6, 0.7, 0.8, the program combination is the same, hence the program performance is the same. It is because when ε is equal to 0.6, 0.7, 0.8, the sensitivity value calculated

with Sche-index2 are all without the range of [0.6,0.8]. When the value of ε is equal to 0.9, the last two programs facesim and x264 in the task queue cannot be scheduled into the cluster, thus the performance of facesim and x264 are 0. The general trend is the greater the value of ε, the better the program performance, and the programs may employ more nodes.

Figure 6 shows program performance with different threshold ε with scheduling strategy Sche-utility. When the value of ε is equal to 0.6, 0.7, the strategy scheduling 14 programs onto 3 nodes, correspondingly more programs running on a single node, leading to relatively low program performance, that are 57% and 66% respectively. However, the average performance value is close to the ε value, which can demonstrate our scheduling strategy can ensure program performance according to ε value. When the value of ε is equal to 0.8, 0.9, the strategy scheduling 14 programs onto 4 nodes, and the average performance is higher when ε is equal to 0.9.

Figure 7 shows program performance with different scheduling strategies. The ε value in scheduling strategy Sche-index2 and Sche-utility are set as 0.8. We can see compared with ll-2, when scheduled by Sche-index2, mg.C.4's performance declines by 25.2%, and the performance of other programs improves. The average performance improves by36%. when scheduled by Sche-utility, cg.C.4's performance declines by 35.6%, mg.C.4's performance declines by 53.5%, and the performance of other programs improve. Overall, the average performance improves by 27.6%. It is because both of the two strategies consider the resource sensitivity of five resource dimensions. They sacrifice the performance of a small portion of CPU-intensive programs to improve the performance of the majority of programs with disk-intensive or network-intensive natures, which can prove that when making the scheduling decision, considering the resource sensitivity can improve the overall performance of the entire workload set. Meanwhile, the difference between the two strategies is, Sche-index2 consider the sensitivity on five resource dimensions at the same time, but Sche-utility considers the sensitivity on one resource dimension at one time, and multiply the five sensitivity value as the evaluation criterion. The experiment results conclude that consider the sensitivity on five resource dimensions at the same time is better than multiply the five sensitivity values.

Figure 8 demonstrates the system resource utilization under different scheduling strategies. Both Sche-index2 and Sche-utility achieve the higher utilization of the disk bandwidth and network bandwidth. It is because compared with ll-2, the other two scheduling strategies considers not only CPU and memory, but also disk r/w bandwidth and network bandwidth. In general, compared with ll-2, by a reasonable combination of programs, Sche-index2 can improve the utilization of CPU, disk read bandwidth, disk write bandwidth, memory and network bandwidth by 33%, 77.7%, 21.9%, 50.7% and 20.9%, and Sche-utility can improve the utilization of CPU, disk read bandwidth, disk write bandwidth, memory and network bandwidth by 6%, 90.5%, 23%, 50.1% and 18%.

7. Related Work

In data centers or clusters, in order to improve the efficiency of workload consolidation, there are some works about scheduling strategy aiming at program's performance and program's QoS.

Scheduling based on the program's characteristics. Paragon [21] and Quasar [22] used collaborative filtering to identify the similarity of the profiling program with the known workloads, then classify the unknown workloads. Based on the above classified information, programs can be scheduled to the optimal execution environment with minimal interference and maximize server utilization. They compared two resource provision strategies on the Google computing engine [23]: resource reservation and on-demand resource provision, and find that both of the two strategies cannot achieve optimal performance and cost. Then they explored the mixed strategy of resource reservation and on-demand resource provision. Tarcil [24] is the high quality and high-speed scheduler for short and long running jobs, with the prior information of resource preference and interference sensitivity. Han et al. [25] investigated how interference in a subset of nodes affects the execution times of the applications, and predicted program's performance under interference in a consolidated cluster. Li et al. [26] proposed adaptive work-stealing strategy tail-control, parallelizing requests by status, system load and target delay. David et al. [27] used an isolation mechanism to parallelize batch tasks and latency-sensitive tasks. But they only studied the program sensitivity to the single resources, lacking the program sensitivity to multiple resources. Tang et al. [28] Improving Application Performance and Cluster Throughput with Resource-Aware Job Placement. They map the programs to a different server according to the program's characteristic. MAGI [29] uses neural networks to monitor and further locate the root cause of performance interference, and to ensure the QoS of the LC application by adjusting the resource share of the corresponding application. Runtime system Laius [30] carefully allocates computing resources to co-located applications to maximize the throughput of batch applications while ensuring the QoS required for user-oriented services. Tang [31] maps programs to a different nodes based on the program characteristic.

Scheduling for the program's QoS. Leverich et al. [32] pointed out that queue latency, scheduling latency and load unbalance result in the QoS violation. Then they use an interference aware manner to resolve the QoS violation. Delimitrou et al. [33] proposed ARQ, a multi-class admission control protocol, dividing applications into classes based on the quality of resources they need, to maintain the QoS of applications. Rajiv et al. [34] proposed Twig, a scalable task manager to ensure the performance of latency-sensitive program. Sage [35] uses machine learning to predict RPC to ensure the QoS. Avalon [36] allocates the "just" core and shared cache space for each query in order to allocate the remaining resources to execute best-effort applications.

Conclusion

In order to improve the efficiency of workload consolidation, this paper first executed multiple experiments about program performance with restricted resource usage, and got multiple conclusions. Then based on the conclusions, two individualized precise scheduling strategies are proposed for efficient consolidation. Sche-index2 schedule programs by considering the resource sensitivity hyper-

plane on five resource dimensions, and Sche-utility schedule programs by considering the sensitivity curve of five resource dimensions. The experiment results showed that compared with ll-2, Sche-index2 could improve the average performance of program by 36%, and Sche-utility could improve the average performance of program by 27.6%. Meanwhile, both of the two scheduling strategies could improve the resource utilization of CPU, disk read bandwidth, disk write bandwidth, memory and network bandwidth.

Declarations

Availability of Data and Materials

The datasets generated and/or analysed during the current study are not publicly available, but are available from the corresponding author on reasonable request.

References

1. R. W. Ahmad, A. Gani, S. H. A. Hamid, M. Shiraz, A. Yousafzai, and F. Xia.: A survey on virtual machine migration and server consolidation frameworks for cloud data centers. J. Netw. Comput. Appl, vol. 52, pp. 11–25 (2015)
2. T Patel, and D Tiwari.: Clite: Efficient and qos-aware co-location of multiple latency-critical jobs for warehouse scale computers. In: 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA). IEEE, pp. 193–206 (2020)
3. L.Zhang, Y.Yang and K.Zhang.: Rhythm: component-distinguishable workload deployment in datacenters. In: Proceedings of the Fifteenth European Conference on Computer Systems. pp. 1–17 (2020)
4. M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes.: Omega: flexible, scalable schedulers for large compute clusters. In: Proceedings of the 8th ACM European Conference on Computer Systems, pp. 351–364 (2013)
5. C. Shuang, C. Delimitrou, and J. F. Martínez.: PARTIES: QoS-aware resource partitioning for multiple interactive services. In: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems. pp. 107–120 (2019)
6. H. Chen, X. Zhu, G. Liu.: Uncertainty-aware online scheduling for real-time workflows in cloud service environment. IEEE Transactions on Services Computing (2018)
7. M. Artemiy, S. Gupta, R. Gonzalez-Alberquilla, and B.Grot.: Stretch: Balancing qos and throughput for colocated server workloads on smt cores. In: 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA), IEEE, pp. 15–27 (2019)
8. J. Zhan, L. Zhang, N. Sun, L. Wang, Z. Jia, and C. Luo.: High volume throughput computing: Identifying and characterizing throughput oriented workloads in data centers. in Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International, pp. 1712–1721 (2012)

9. L. A. Barroso, J. Clidaras, and U. Hölzle.: The datacenter as a computer: An introduction to the design of warehouse-scale machines. Synthesis lectures on computer architecture, vol. 8, no. 3, pp. 1-154 (2013)
10. S. Chen, S. Galon, C. Delimitrou, S. Manne, and J. F. Martinez.: Workload Characterization of Interactive Cloud Services on Big and Small Server Platforms. in Proc. of IISWC, IEEE, pp. 125–134 (2017)
11. J. Mars and L. Tang.: Whare-map: heterogeneity in homogeneous warehouse-scale computers. In: ACM SIGARCH Computer Architecture News, vol. 41, no. 3, pp. 619–630 (2013)
12. L. Wang, D. Qian, Z. Luan, G. Wei, R. Wang, and H. Yang.: Speeding up profiling program's runtime characteristics for workload consolidation. PloS one, vol. 12, no. 4, p. e0175861 (2017)
13. P. C. Fishburn.: Nonlinear preference and utility theory. Johns Hopkins University Press Baltimore (1988)
14. G. W. Fischer.: Multidimensional utility models for risky and riskless choice. Organizational Behavior and Human Performance, vol. 17, no. 1, pp. 127–146 (1976)
15. NAS Parallel Benchmark. http://www.nas.nasa.gov/publications/npb.html.
16. PARSEC. http://parsec.cs.princeton.edu/.
17. SysBench. https://github.com/akopytov/sysbench.
18. CloudSuite. http://parsa.epfl.ch/CloudSuite/CloudSuite.html.
19. Cgroups. https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt.
20. S. Di, D. Kondo, and F. Cappello.: Characterizing cloud applications on a Google data center. in Parallel Processing (ICPP), In: 2013 42nd International Conference on, pp. 468–473 (2013)
21. C. Delimitrou and C. Kozyrakis.: Paragon: QoS-aware scheduling for heterogeneous datacenters. in ACM SIGPLAN Notices, vol. 48, no. 4, pp. 77–88 (2013)
22. C. Delimitrou and C. Kozyrakis.: Quasar: resource-efficient and QoS-aware cluster management. ACM SIGPLAN Notices, vol. 49, no. 4, pp. 127–144 (2014)
23. C. C. Delimitro. Optimizing resource provisioning in shared cloud systems. Tech. Rep (2014)
24. C. Delimitrou, D. Sanchez, and C. Kozyrakis.: Tarcil: reconciling scheduling speed and quality in large shared clusters. in Proceedings of the Sixth ACM Symposium on Cloud Computing, pp. 97–110 (2015)
25. J. Han, S. Jeon, Y.-r. Choi, and J. Huh.: Interference management for distributed parallel applications in consolidated clusters. ACM SIGPLAN Notices, vol. 51, no. 4, pp. 443–456 (2016)
26. J. Li et al.: Work stealing for interactive services to meet target latency. in ACM SIGPLAN Notices, vol. 51, no. 8, pp. 1–13 (2016)
27. D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis.: Heracles: improving resource efficiency at scale. In: ACM SIGARCH Computer Architecture News, vol. 43, no. 3, pp. 450–462 (2015)
28. X. Tang, H. Wang, X. Ma, N. El-Sayed, J. Zhai, W. Chen, and A. Aboulnaga.: Spread-n-share: improving application performance and cluster throughput with resource-aware job placement. In: Proceedings

of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–15 (2019)

29. W.Sa, Z.Yanhai and C.Shanpei.: A Case for Adaptive Resource Management in Alibaba Datacenter Using Neural Networks. Journal of Computer Science and Technology 35.1, pp. 209–220 (2020)
30. W. Zhang, W. Cui, K. Fu, Q. Chen, D. E. Mawhirter, B. Wu, C. Li, and M. Guo.: Laius: Towards Latency Awareness and Improved Utilization of Spatial Multitasking Accelerators in Datacenters. in ACM Conf. on Supercomputing, pp. 58–68 (2019)
31. X Tang, H Wang and X Ma.: Spread-n-share: improving application performance and cluster throughput with resource-aware job placement. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 1–15 (2019)
32. J. Leverich and C. Kozyrakis.: Reconciling high server utilization and sub-millisecond quality-of-service. In: Proceedings of the Ninth European Conference on Computer Systems, pp. 1–14 (2014)
33. C. Delimitrou, N. Bambos, and C. Kozyrakis.: QoS-Aware Admission Control in Heterogeneous Datacenters. in ICAC, pp. 291–296 (2013)
34. N. Rajiv, V. Petrucci, P. Carpenter, and M. Själander.: Twig: Multiagent task management for colocated latency-critical cloud services. in Proceedings of the International Symposium High Performance Computer Architecture. ACM. pp. 167–179 (2020)
35. Y Gan, S Dev, D Lo and C Delimitrou.: Sage: Leveraging ML To Diagnose Unpredictable Performance in Cloud Microservices. ML for Computer Architecture and Systems, (2020)
36. Q. Chen, Z. Wang, J. Leng, C. Li, W. Zheng, and M. Guo.: Avalon: Towards QoS Awareness and Improved Utilization Through Multi-Resource Management in Datacenters. in ACM Conf. on Supercomputing, pp. 272–283 (2019)

Figures

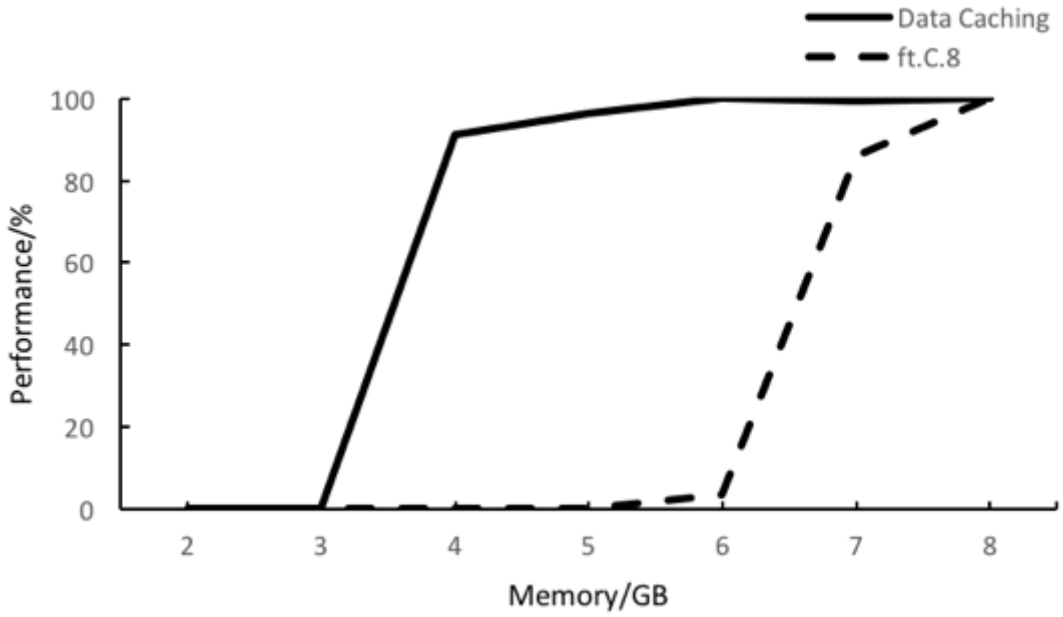

Figure 1

Program performance relative to the running-alone performance under different scheduling strategies.

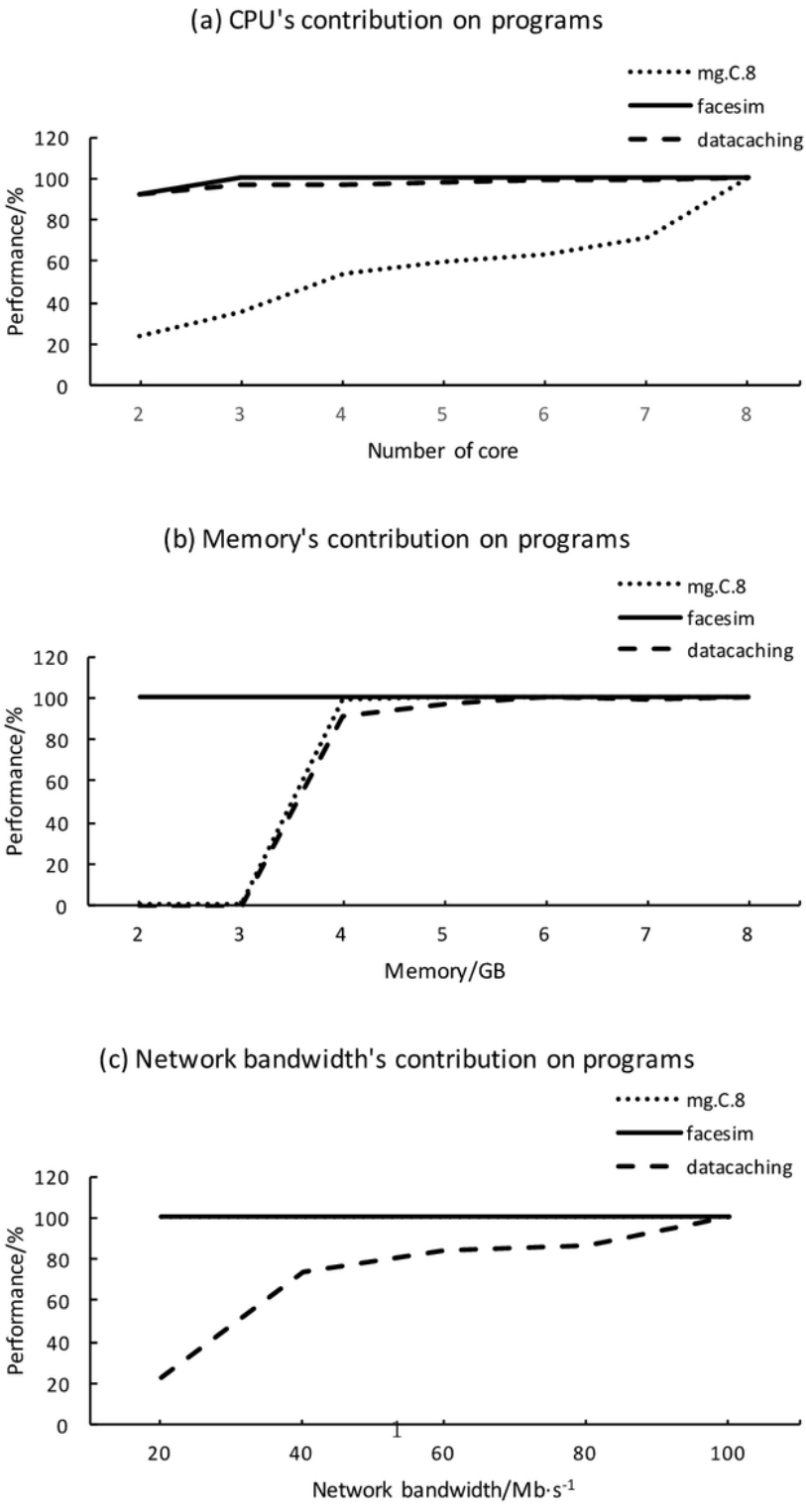

Figure 2

Resource contribution to program mg.C.8, facesim and Data Caching.

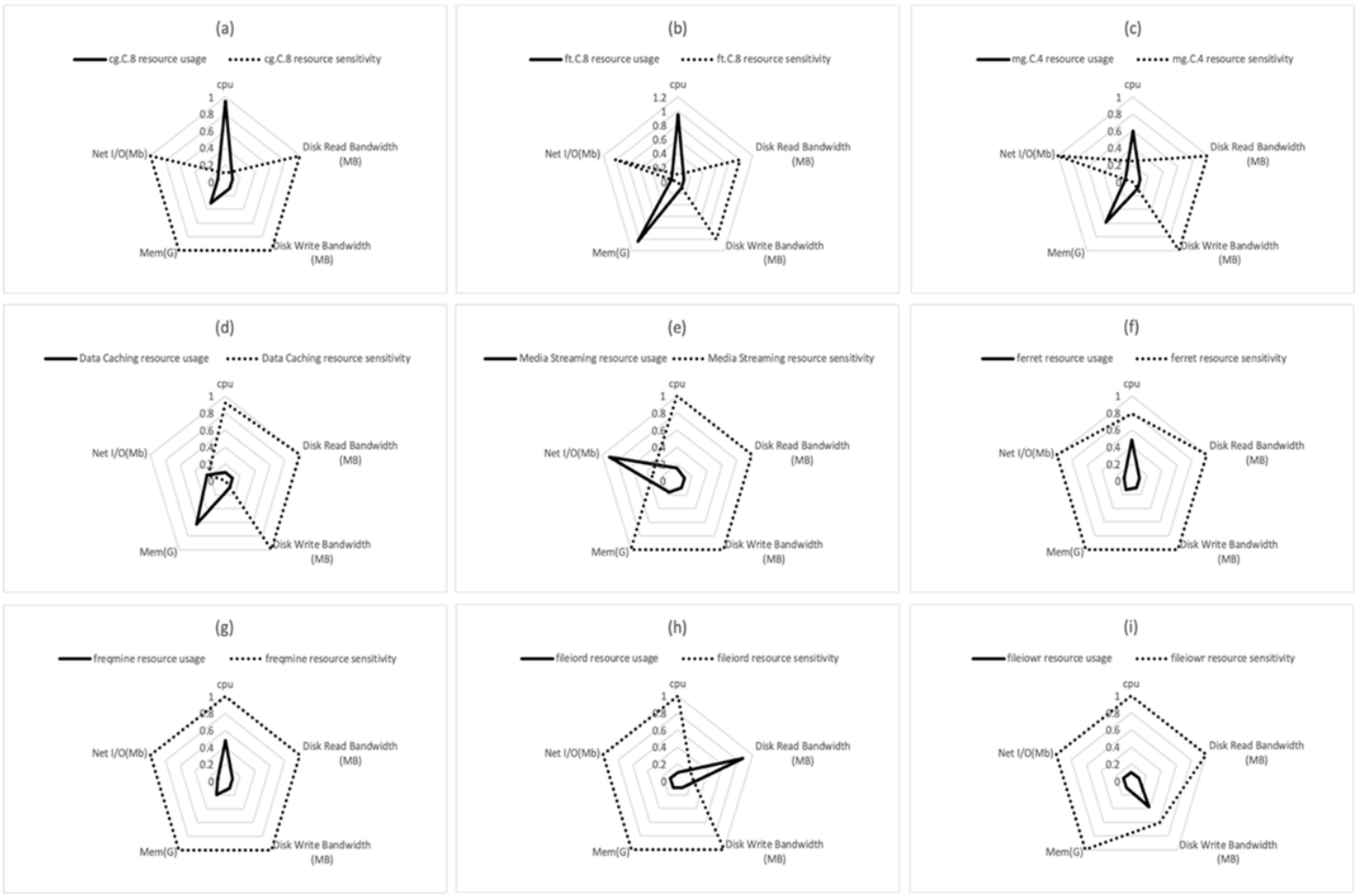

Figure 3

Resource usage and resource sensitivity of programs

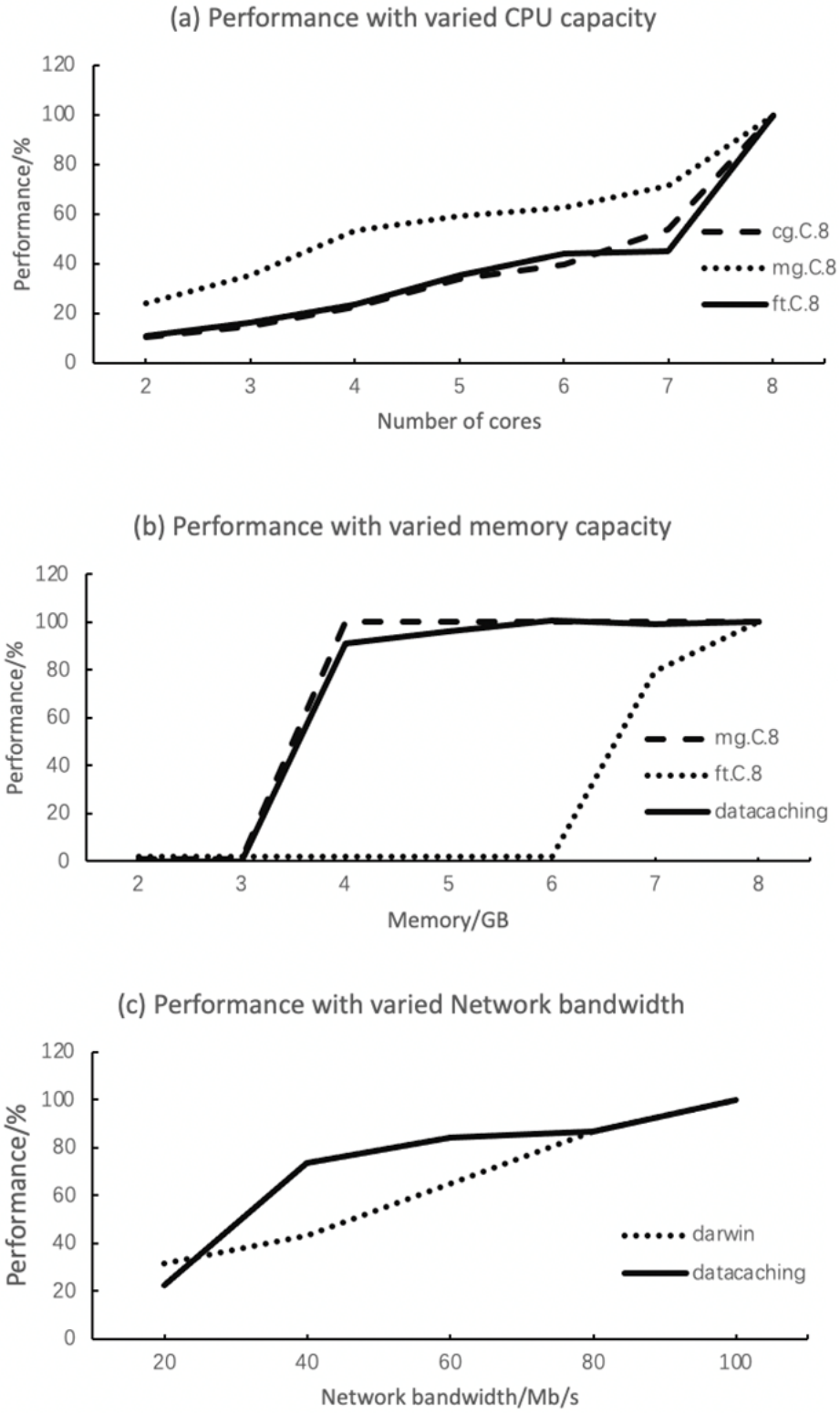

Figure 4

Program's performance varied with differ-ent resource usage

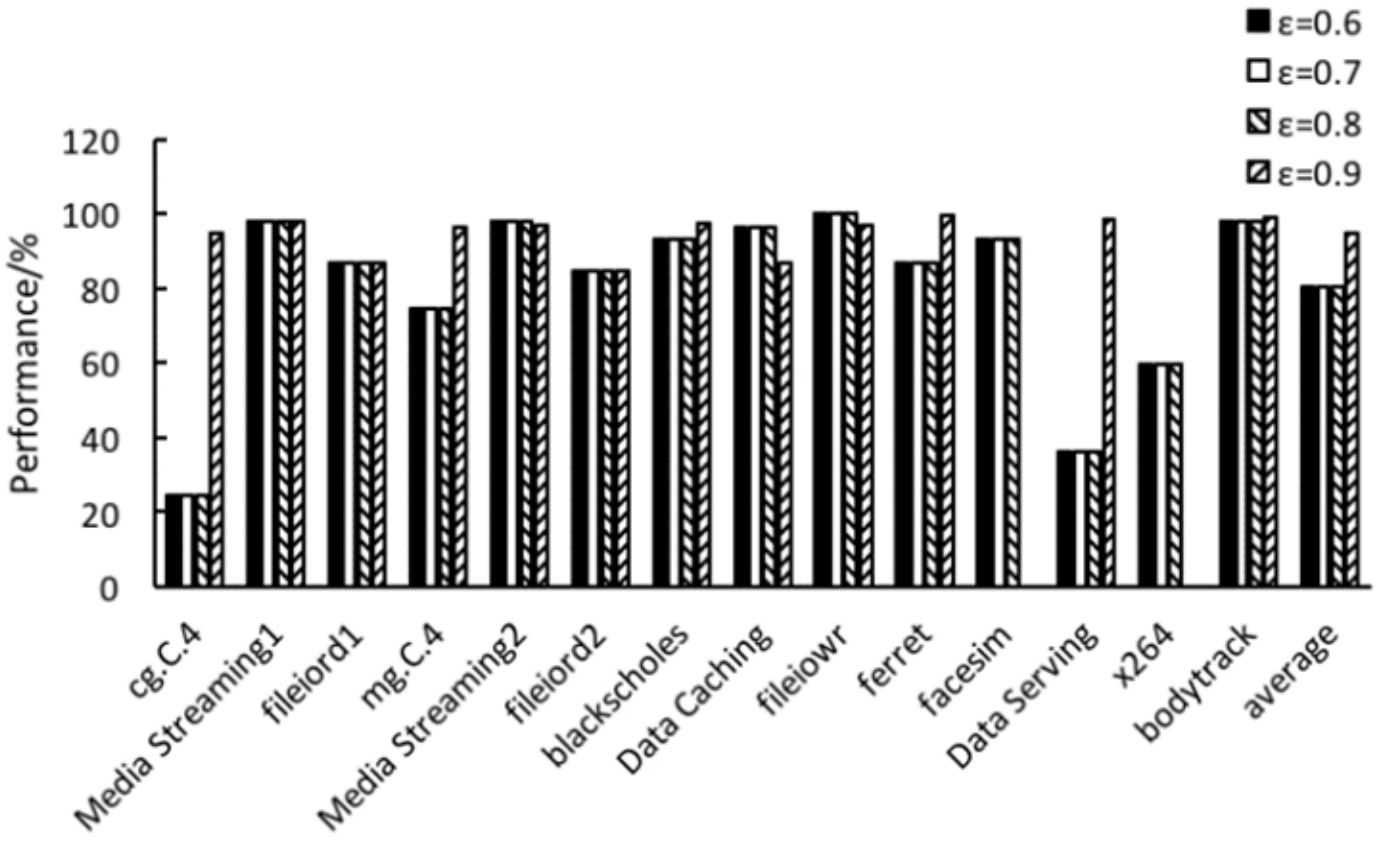

Figure 5

Program performance with different threshold ε with scheduling strategy Sche-index2

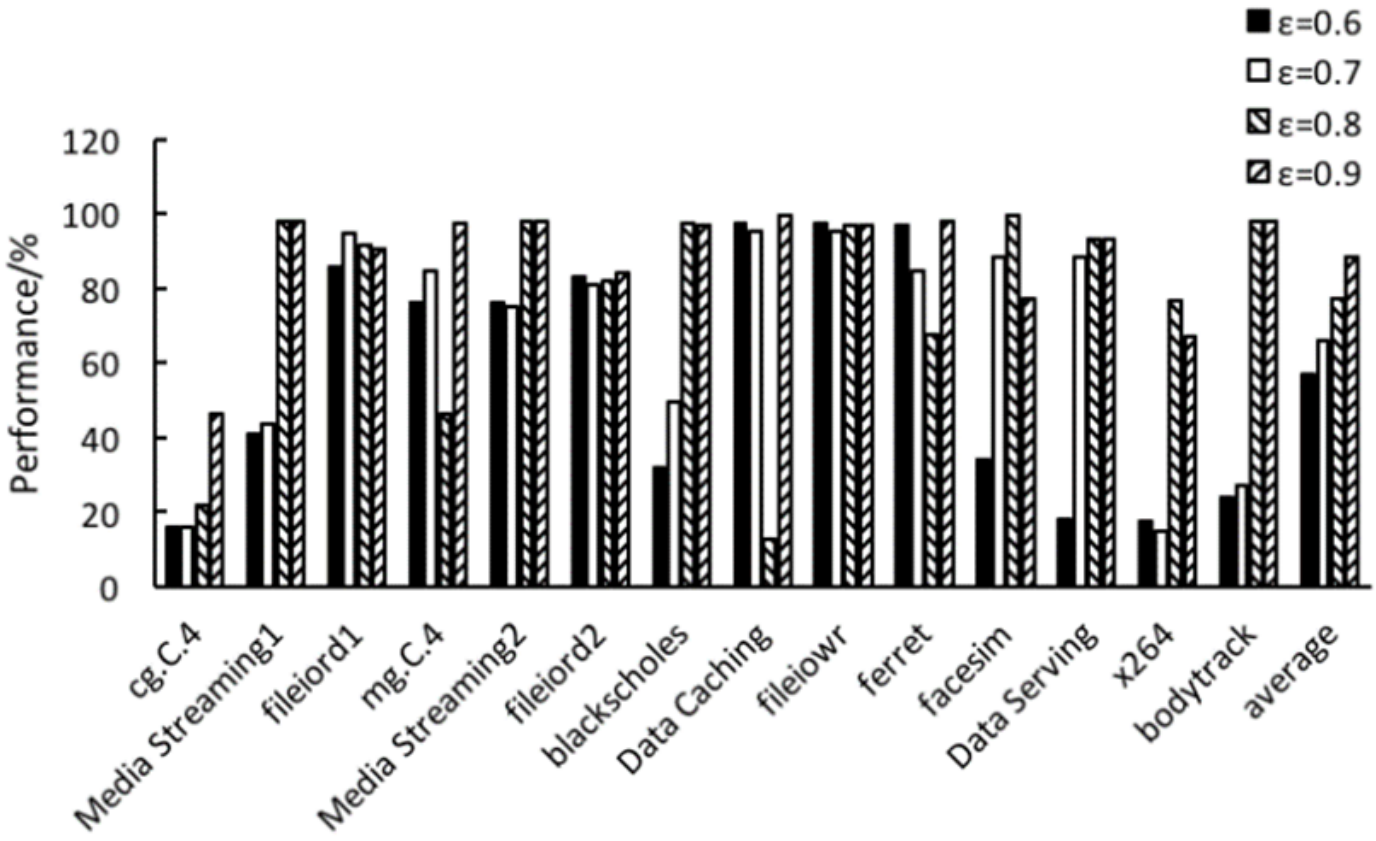

Figure 6

Program performance with different threshold ε with scheduling strategy Sche-utility.

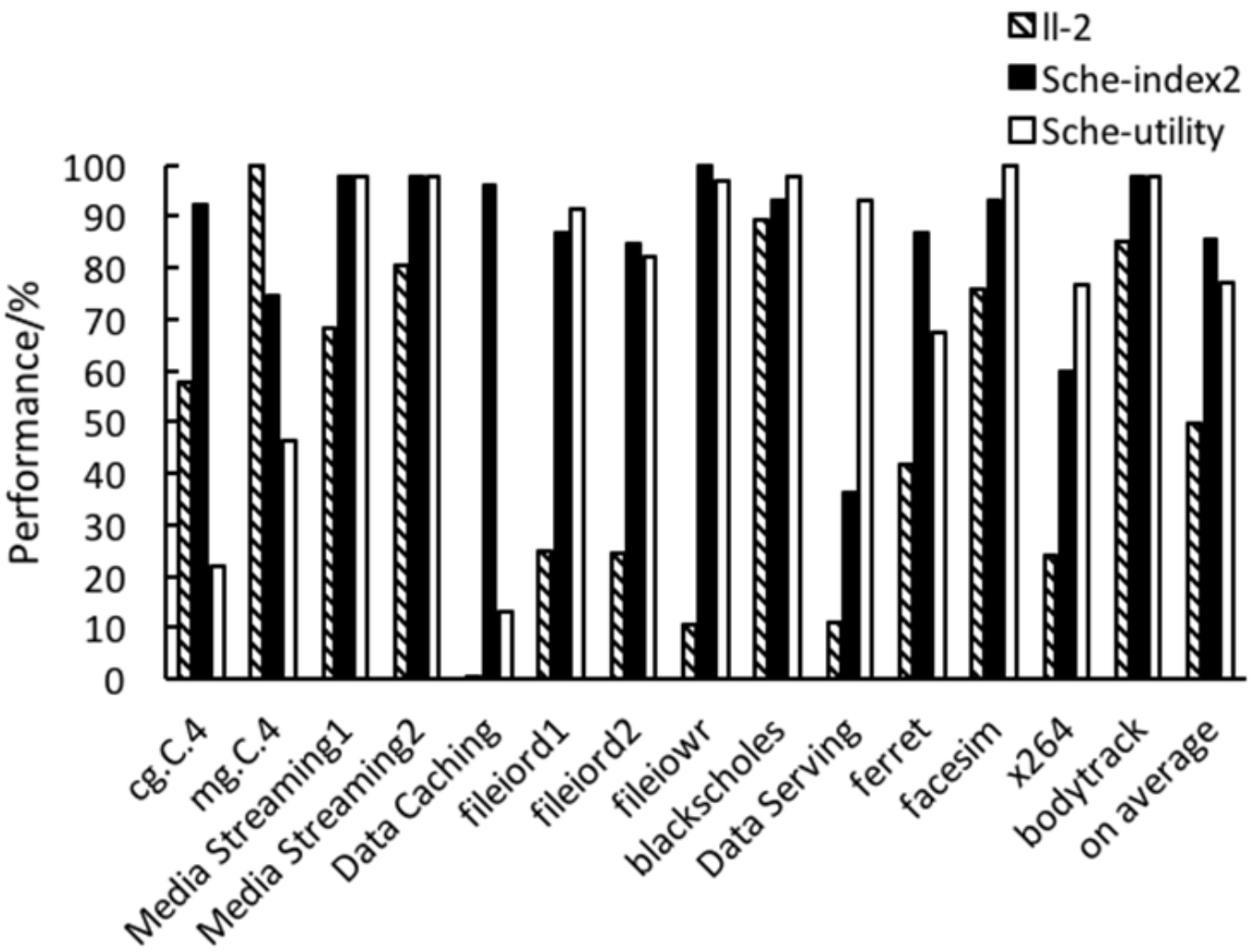

Figure 7

Program performance with different scheduling strategies.

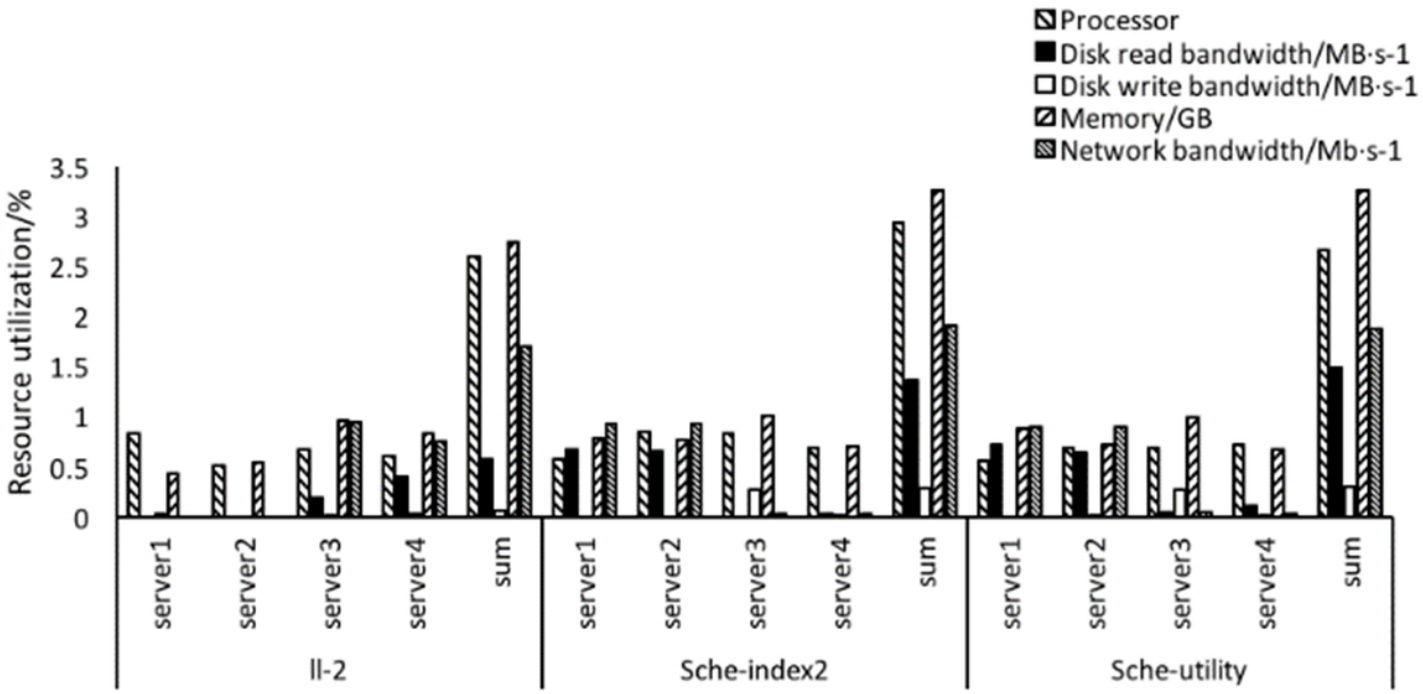

Figure 8

System resource utilization under different scheduling strategies.