Guaranteeing Co-running Program’S Performance in Data Center

Ti Guan, Yong Liu, Guanbin Wu, Daning You and Xuekai Zhang
State Grid Shandong Electric Power Company, Jinan 250001, Shandong Province, China
Email: sddl_zdh@163.com, 15963922979@163.com, wuguanbin@sd.sgcc.com.cn, youdaning@sd.sgcc.com, 84190464@qq.com

Abstract. The data centers are more popular nowadays. In order to improve the utilization of the servers, programs commonly co-running with each other on one server, and programs may interference with each other. Otherwise, for programs who running in cluster or data centers, commonly they can be classified to different kind of program, for example, the batch program, the latency-sensitive program and so on. Different kind of programs has different kind of duty for performance. In order to improve the utilization of the servers while concurrently guarantee the performance of latency-sensitive program, we propose performance guaranteed program co-running algorithm. We design three engines to realize the performance guaranteed program co-running algorithm, they are Program Monitor Engine, Resource Division Engine and Resource Partition Engine respectively. The Program Monitor Engine collects the program’s resource usage information about CPU and memory, and then label the programs. The Resource Division Engine calculate the size of CPU and memory resource should be divided for latency-sensitive program. The Resource Division Engine set the resource control group with the CPU and memory value. All the latency-sensitive programs are running in this resource control group. Meanwhile, out side the resource control group, multiple batch programs can be executed. The experiment result show that by making use of the performance guaranteed program co-running algorithm, most of the latency-sensitive program’s performance sustain with in 95%, at the same time, the batch programs are used to increase the throughput as more as possible.

1. Introduction
The data centers are build based on the Internet network infrastructure to accelerate the calculation, display and data storage, which are more popular nowadays [1]. It provides a serious of value-added service on the internet, including the specific application domain name, virtual hosting space, hosting and other business. Data center is a complex set of facilities, not only including computer systems and other related equipment (such as communication and storage system), also contains redundant data communication connection environmental control equipment, monitoring equipment, and various safety devices.

In data center or cluster, in order to improve the utilization of the servers [2], programs commonly co-running with each other on one server, and programs may interference with each other. Some times the interference is small, which do little effect on programs; but some times the interference is big, which drop down program’s performance dramatically.

Otherwise, for programs who running in cluster or data centers, commonly they can be classified to different kind of program, for example, the batch program, the latency-sensitive program and so on. Different kind of programs has different kind of duty for performance. Take an example, commonly people have little expect on batch program, but the latency-sensitive program should guarantee QoS
agreement. The QoS requirement can be appointed by the custom, for instance the 85% of the program’s performance when running alone.

In order to improve the utilization of the servers while concurrently guarantee the performance of latency-sensitive program, a smart algorithm should be proposed to solve the trade-off problem. In this paper we propose performance guaranteed program co-running algorithm to guarantee the latency-sensitive program’s performance. This algorithm can guarantee the latency-sensitive program’s performance and improve the utilization of the servers as much as possible. We design three engines to realize the performance guaranteed program co-running algorithm, they are Program Monitor Engine, Resource Division Engine and Resource Partition Engine respectively.

The Program Monitor Engine collects the program’s resource usage information about CPU and memory, and then label the programs. We use Linux tool Top to collect the CPU and memory usage information off-line. Then this information are transferred to Resource Division Engine.

The input of the Resource Division Engine is the program queue and the information transferred from the Program Monitor Engine. The output of the Resource Division Engine is the size of CPU and memory resource should be divided for latency-sensitive program, and the rest of the resource is applied for the batch program. Then the value of the CPU and memory resource for resource guarantee group are transferred to the Resource Partition Engine.

The input of the Resource Partition Engine is the information passed from Resource Division Engine, and the output of the Resource Partition is the size to set the CPU and memory resource. After processed by the Resource Partition Engine, A resource control group would be set up for latency-sensitive programs, and all the latency-sensitive programs are running in this resource control group. Meanwhile, out side the resource control group, multiple batch programs can be executed.

The experiment result show that by making use of the performance guaranteed program co-running algorithm, most of the latency-sensitive program’s performance sustain with in 95%, at the same time, the batch programs are used to increase the throughput as more as possible.

2. Related work
There are a lot of related work on co-running program’s performance in CMP, cluster and data centers.

Some work scheduling programs on CMP to avoid serious interference [3, 4]. They make use of on-chip event to predict program’s performance. Some work scheduling programs on cluster to increase the utilization [5]. They use pressure test programs to predict program’s performance when co-running with other programs.

Otherwise we have to introduce the implementation tool Cgroups. Cgroups [6] is a Linux resource isolation tools for resource management. It is provided by Linux kernel and isolate the process by making use of material resource, such as CPU, memory and network. Cgroups provide a virtual file system as the group management subsystem and interface setting. For each group, Cgroups can set the size of resource according the operator. Then we can drag the program to the control groups for execution [7].

3. Resource isolation method
We use Linux resource isolate tool Cgroups to realize the performance guaranteed program co-running algorithm. One one hand, this algorithm can ensure program’s performance by divide enough resource for program, one the other hand, this algorithm can improve the server’s resource utilization by make the most use of the surplus resource. In order to realize the plan, the following engine should be designed, and the flow chart is in Figure 1.

Module 1: Program Monitor Engine. This work should be done off-line. For new programs come into cluster, this engine collects the program’s resource usage information about CPU and memory, and then label the programs. The label (resource usage information) can be used for Resource Division Engine to decide how to divide the resource. The general performance profiling tools can be used to help us to collects the program’s resource usage information about CPU and memory. In this paper we use Linux tool Top. Every program’s (latency-sensitive program and batch program) resource usage of CPU and memory information are topped and passed to Module 2.
Module 2: Resource Division Engine. This part is executing on-line. In order to ensure the efficiency of on-line process. This engine should be simple and efficient. The input of the Resource Division Engine is the program queue and the information transferred from Module 1, which are the resource usage information on CPU and memory of the latency-sensitive program and batch program. The output of the Resource Division Engine is the size of CPU and memory resource should be divided for latency-sensitive program, and the rest of the resource is applied for the batch program. Furthermore, we set a resource guarantee group on the server, and the resource size of the resource guarantee group is the sum of the latency-sensitive program scheduling on this server, which are in formula (1) and formula (2).

\[
C_{\text{guarantee group}} = C_{\text{lat sen1}} + C_{\text{lat sen2}} + \ldots + C_{\text{lat sen n}} 
\] (1)

\[
M_{\text{guarantee group}} = M_{\text{lat sen1}} + M_{\text{lat sen2}} + \ldots + M_{\text{lat sen n}} 
\] (2)

In the above two formula, \(C_{\text{guarantee group}}\) is the size of the CPU resource of the resource guarantee group for latency-sensitive program, and \(M_{\text{guarantee group}}\) is the size of the memory resource of the resource guarantee group for latency-sensitive program. \(\text{lat sen1}, \text{lat sen2}, \ldots, \text{lat sen n}\) are the latency-sensitive programs scheduled to the server.

The resource size for batch program is the size of server resource minus the size of the resource guarantee group, which is in formula (3) and formula (4).

\[
C_{\text{batch group}} = C_{\text{server}} - C_{\text{guarantee group}} 
\] (3)

\[
M_{\text{batch group}} = M_{\text{server}} - M_{\text{guarantee group}} 
\] (4)
In the above two formula, $C_{\text{batch group}}$ is the size of the CPU resource for the batch program, and $M_{\text{batch group}}$ is the size of the memory resource for batch program. $C_{\text{server}}$ is the CPU resource of the server and $M_{\text{server}}$ is the memory resource of the server.

Then the value of $C_{\text{guarantee group}}$ and $M_{\text{guarantee group}}$ are transferred to Module 3.

Module 3: Resource Partition Engine. The input of the Resource Partition Engine is the information passed from Module 2, and the output of the Resource Partition is the size to set the CPU and memory resource. After processed by the Resource Partition Engine, A resource control group would be set up for latency-sensitive programs, the size of the resource control is the value calculated by formula (1) and formula (2), and all the latency-sensitive programs are running in this resource control group. Meanwhile, out side the resource control group, multiple batch programs can be executed.

By this method, the latency-sensitive programs can be executed in resource control group with enough resource, the sufficient resource would guarantee the latency-sensitive program’s performance. Meanwhile, the batch programs can be executed outside the resource control group, in order to improve the resource utilization, we can start multiple batch programs to fulfill the resource outside the resource control group, with the price of scarifies every single program’s performance.

However, the number of co-running programs outside the resource control group is not unlimited. Too many co-running programs contending the limited resource will result in killing of the server. In the next section we will discuss the number of the co-running batch programs.

4. the discussion of batch programs
This section explores the number of batch programs co-running on the server. As we know, when too many co-running programs contending the limited resource, the server will be killed.

On one hand we want to run as many as possible programs to improve the resource utilization and the throughput of the server. One the other hand we are afraid the server down because of the server can not deal with too many programs. So we have to found the trade-off point.

Assume there are $C_{\text{batch group}}$ CPU resource and $M_{\text{batch group}}$ memory resource for the batch program. We have to set the maximum number of batch program. We start batch programs one by one, and accumulate the program’s resource usage on CPU and memory, once the sum of the CPU and memory are outstrip the 200% of the resource for batch programs, then wen stop start the new program until some batch program finish. The maximum number of batch program should obey the algorithm in Figure 2.

The maximum number of batch program co-running on server

```plaintext
1: cpu_sum=cpu[p1]
2: mem_sum=mem[p1]
3: for i = p1 to pns do
4: cpu_sum=cpu_sum+cpu[i+1];
5: mem_sum=mem_sum+mem[i+1];
6: if cpu_sum >cpu batch*2 or mem_sum >mem batch*2
7: { 
8: num_pro=i
9: break;
10: }
11: end for
```

Figure 2 Pseudo-code of caculating the maximum number of co-running program running on one server

5. Experiment
The experiment servers are two socket, 4 cores are on one socket, and the detailed server information is in TABLE 1. We select programs from NAS Parallel Benchmarks and take them as latency-sensitive
programs. We select programs from The Princeton Application Repository for Shared-Memory Computers (PARSEC) and take them as batch programs. The programs used by the experiment are in TABLE 2.

| CPU          | Intel Xeon E5620             |
|--------------|------------------------------|
| core         | 4 cores@2.13G                |
| L1 caches    | 32K                          |
| L2 caches    | 256K                         |
| L3 caches    | 4M                           |
| Threads per core | 1 thread                  |
| Sockets      | 2                            |
| Memory       | 8GB, DDR3                    |

### Table 1 Server configuration

| Benchmark                | program       | feature     |
|--------------------------|---------------|-------------|
| NAS Parallel Benchmarks  | ft.C.4, cg.C.4, mg.C.8 | latency-sensitive |
| PARSEC 3.0               | blackscholes, facesim, bodytrack | batch |

We compare co-running program’s performance with the program’s performance when running alone. Then check the QoS of latency-sensitive program’s performance. We select three workloads. In every workload there is one program selected from latency-sensitive programs and one program selected from batch programs. We run as more as possible batch programs according our algorithm.

Figure 3 shows the latency-sensitive program’s performance and the batch program’s performance of workload 1 when co-running on one server. The workload is ft.C.4 and blackscholes. When ft.C.4 co-running with blackscholes, there are two batch programs can co-running at the same time. It is because ft.C.4 consume almost all the memory of the server, there is no memory space for the third batch program, although there is idle CPU resource. It is shows that the latency-sensitive program can sustain it’s performance with 95%, and the batch program executing with 62%. The low performance is the trade off of throughput.

![Comparing co-running program’s performance](image)

**Figure 3** Comparing co-running program’s performance of workload 1

Figure 4 shows the latency-sensitive program’s performance and the batch program’s performance of workload2 when co-running on one server. The workload is cg.C.4 and facesim. When cg.C.4 co-running with facesim, there are three batch programs can co-running at the same time. It is because there is no more CPU space for the forth batch program. It is shows that the latency-sensitive program can sustain it’s performance with 97%, and the batch program executing with 45%.
Figure 4 Comparing co-running program’s performance of workload 2

Figure 5 shows the latency-sensitive program’s performance and the batch program’s performance of workload 3 when co-running on one server. The workload is mg.C.4 and bodytrack. When mg.C.4 co-running with facesim, there is only one batch programs can co-running. It is because there is no more CPU space for the other batch program. It is shows that the latency-sensitive program can sustain it’s performance with 98%, and the batch program executing with 92%.

Figure 5 Comparing co-running program’s performance of workload 3

6. Conclusion
In order to improve the utilization of the servers while concurrently guarantee the performance of latency-sensitive program. In this paper we propose performance guaranteed program co-running algorithm to guarantee the latency-sensitive program’s performance. There are three component to realize the algorithm, there are Program Monitor Engine, Resource Division Engine and Resource Partition Engine respectively.

The experiment result show that by making use of the performance guaranteed program co-running algorithm, most of the latency-sensitive program’s performance can be guaranteed by the expect value, and the batch programs are used to increase the throughput of the server.

7. References
[1] A. Mohammad, L. Alexander. A scalable, commodity data center network architecture[J]. ACM SIGCOMM Computer Communication Review, 2008, 38(4): 63-74
[2] X. Li, Z. Qian, S Lu. Energy efficient virtual machine placement algorithm with balanced and improved resource utilization in a data center[J]. Mathematical and Computer Modelling, 2013, 58(5): 1222-1235
[3] S. Zhuravlev, S. Blagodurov, A. Fedorova. Addressing Shared Resource Contention in Multicore Processors via Scheduling[C]. Fifteenth International Conference on Architectural Support for Programming Languages and Operating Systems, 2010: 129-141

[4] J. Mars, N. Vachharajani, R. Hundt. Contention Aware Execution: Online Contention Detection and Response[C]. The 8th annual IEEE/ACM international symposium on Code generation and optimization (CGO ’10), 2010: 257-265

[5] J. Mars, L. Tang, R. Hundt. Bubble-Up: increasing utilization in modern warehouse scale computers via sensible co-locations[C]. The 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-44), 2011: 248-259

[6] Cgroups [EB/OL], https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt

[7] L. Wang, D. Qian, Z. Luan, G. Wei, R. Wang, H. Yang. Speeding up profiling program’s runtime characteristics for workload consolidation. PLoS ONE 12(4): e0175861.