École Polytechnique Fédérale de Lausanne

CS477 Advanced Operating Systems Lab1 Report

by Yuchen Ouyang

EPFL IC IINFCOM HEXHIVE
BC 160 (Bâtiment BC)
Station 14
CH-1015 Lausanne

October 13, 2024

Chapter 1

Round Robin

1.1 Solutions

Basic idea:

1. Retrieve CPU count and iterator from bpf map.
2. Increase and module the iterator by the count.
3. Validate the target CPU.
4. Redirect the packet to the target index.

1.2 Explanation

(CPU0 is unavailable all time, the following data is acquired from 7 working cores)

1. **Evaluate and document the results you see for queueing delay and round trip time (RTT) as seen on the server and client respectively, for both unimodal and bimodal traffic.**

   Given the results in Figure 1.1 and Figure 1.2 about the queueing delay and RTT in unimodal, and Figure 1.3 and Figure 1.4 respectively for bimodal traffic, we can see that both the types of traffic will cause an increase delay or RTT as the traffic grows exponentially (neglect the initial spikes):

   The unimodal case has queueing delay and RTT growing up to about 1000 us and 11000 us respectively.

   Both the short and long services in the bimodal case has queueing delay and RTT growing up to about 1000us and 10000us respectively.

2. **Explain the effect of the packet scheduling policy for both kinds of traffic, and provide a convincing explanation for why that happens.**

Both the cases have delay increasing with the traffic load. As the load grows, more tasks are pushed into the waiting queues of CPUs. So the CPU queueing delay and total response time will definitely increase.

For comparison between the unimodal and bimodal cases, the delay of bimodal should be slightly higher than the unimodal case for the existence of head-of-line blocking phenomenon. (However, the performance results of bimodal case are slightly better than the unimodal case during many experiments in my working environment). One long service blocks all short services after it, which increases the queueing delay of short services by about one long service time.

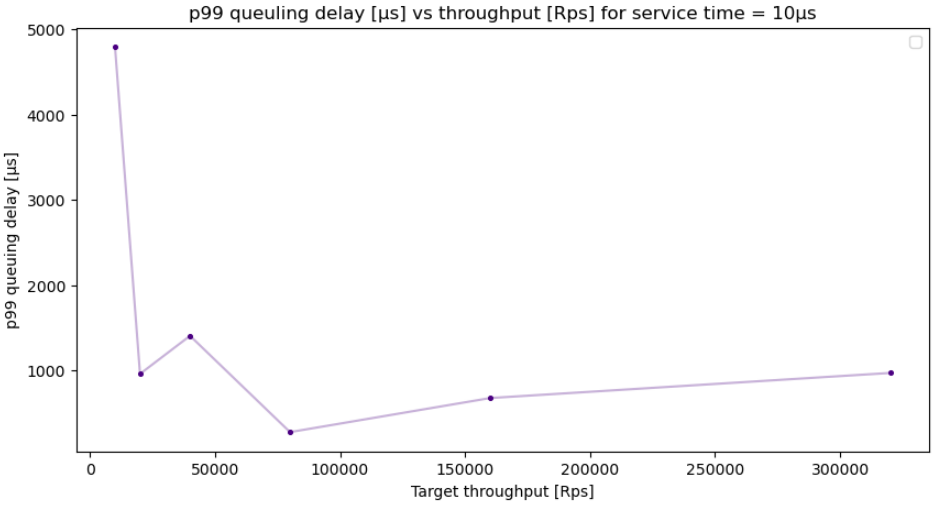

Figure 1.1: Queueing Delay for Unimodal (RR)

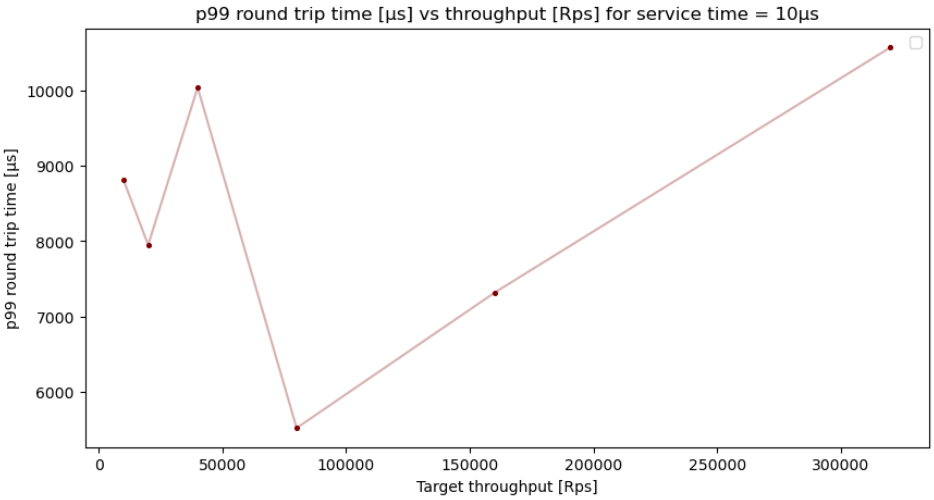

Figure 1.2: RTT for Unimodal (RR)

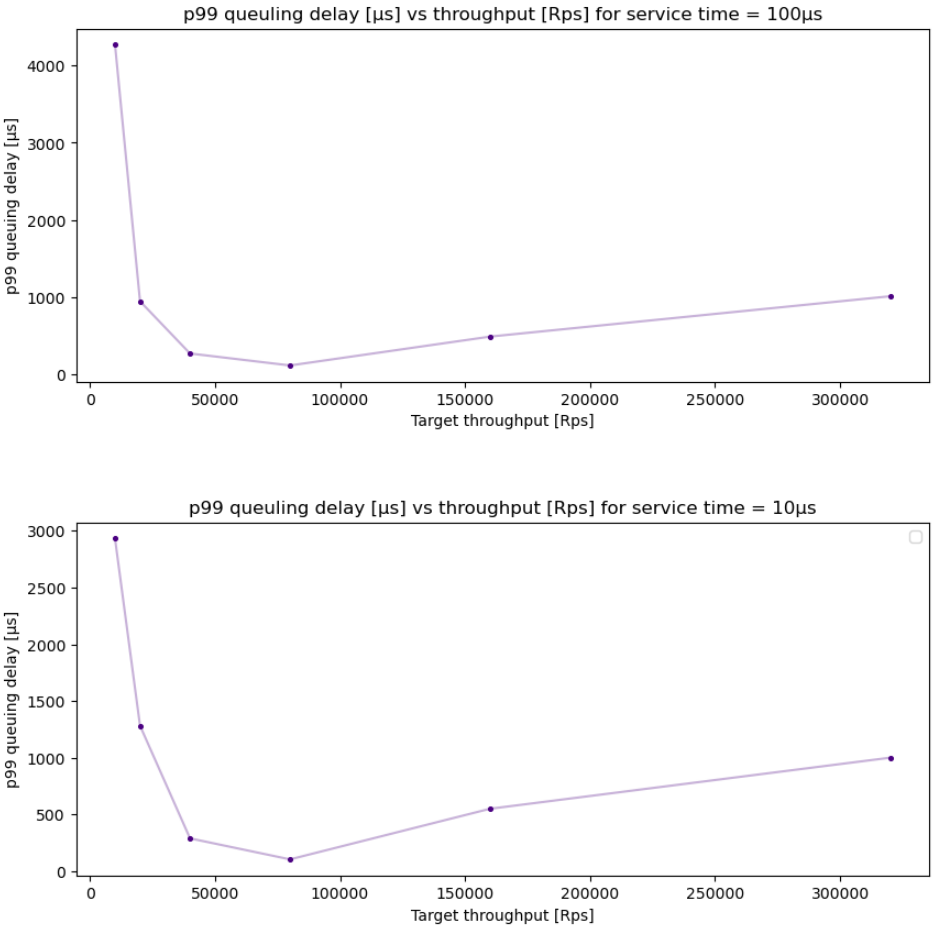

Figure 1.3: Queueing Delay for Bimodal (RR)

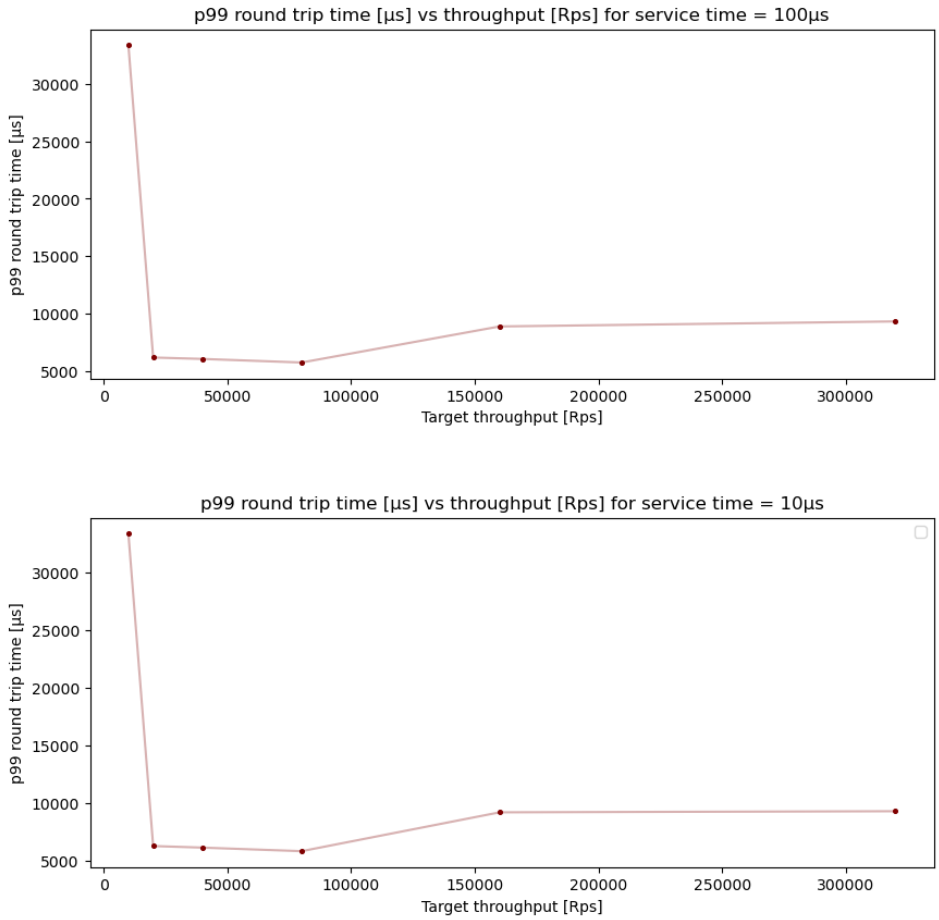

Figure 1.4: RTT for Bimodal (RR)

Chapter 2

Round Robin Core Separated

2.1 Solutions

Basic idea:

1. Parse the packet header to identify whether it is a long or short service.
2. Retrieve both short and long CPU counts and iterators from bpf maps.
3. Increase and module the corresponding iterator by the corresponding count.
4. Validate the target CPU.
5. Redirect the packet to the target index by adding the short count offset.

2.2 Explanation

(CPU0 is unavailable all time, the following data is acquired from 3 working cores for short services and 4 working cores for long services)

1. **Evaluate and document the results you see for queueing delay and round trip time (RTT) as seen on the server and client respectively, for both unimodal and bimodal traffic.**

   Given the results in Figure 2.1 and Figure 2.2 about the queueing delay and RTT in unimodal, and Figure 2.3 and Figure 2.4 respectively for bimodal traffic, we can see that both the types of traffic will cause an increase delay or RTT as the traffic grows exponentially:

   The unimodal case has queueing delay and RTT growing up to about 14000 us and 30000 us as the traffic load increases respectively.

   The short services in the bimodal case has queueing delay and RTT growing up to about 8000 us and 23000us. And the long services are less than 500 us and 22000 us respectively.

   We can see that the long services are processed and also finished more rapidly than the short services despite their longer service time.

2. **Explain the effect of the packet scheduling policy for both kinds of traffic, and provide a convincing explanation for why that happens.**

   - Both the cases have delay increasing with the traffic load. As the load grows, more tasks are pushed into the waiting queues of CPUs. So the CPU queueing delay and total response time will increase.
   - comparison between the unimodal and bimodal cases:
     - the delay metrics of short service in bimodal are smaller than the unimodal case. This could be resulted by the decreasing of the number of tasks in the bimodal task distribution, so that the waiting queue per CPU is shorter.
     - the delay metrics of long service in bimodal are quite low, because the number of the long task is small and there are separated CPUs that only serve the long tasks. The waiting queue in long-service CPUs are quite short. The exponentially loaded traffic still could not fill the queue in the long-service CPUs, which leads to the uniform queueing of the long service.
   - comparison between the RRCS and RR scheduling policy:
     - unimodal: The delay metrics of unimodal tasks are higher in the RRCS than RR, for the reason that there are fewer CPUs working for short tasks (3 in RRCS and 7 in RR) which causes load imbalance between the short-service CPUs and long-service ones.
     - bimodal: The head-of-line blocking delay is mitigated by the separated CPUs. Therefore, the short tasks will benefit from the RRCS scheduling for shorter waiting time, which compensates the negative impacts of load imbalance. The long tasks will definitely get quicker processing for their small amount and the monopoly of long-service CPUs.

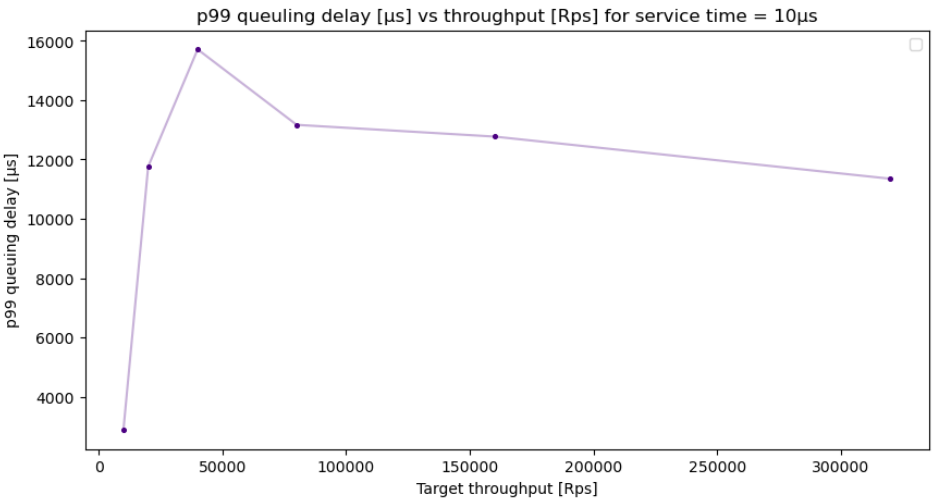

Figure 2.1: Queueing Delay for Unimodal (RRCS)

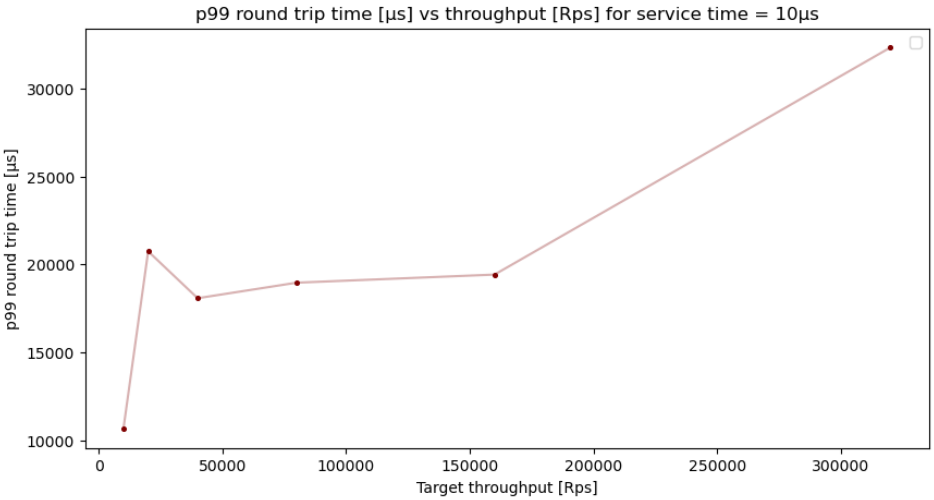

Figure 2.2: RTT for Unimodal (RRCS)

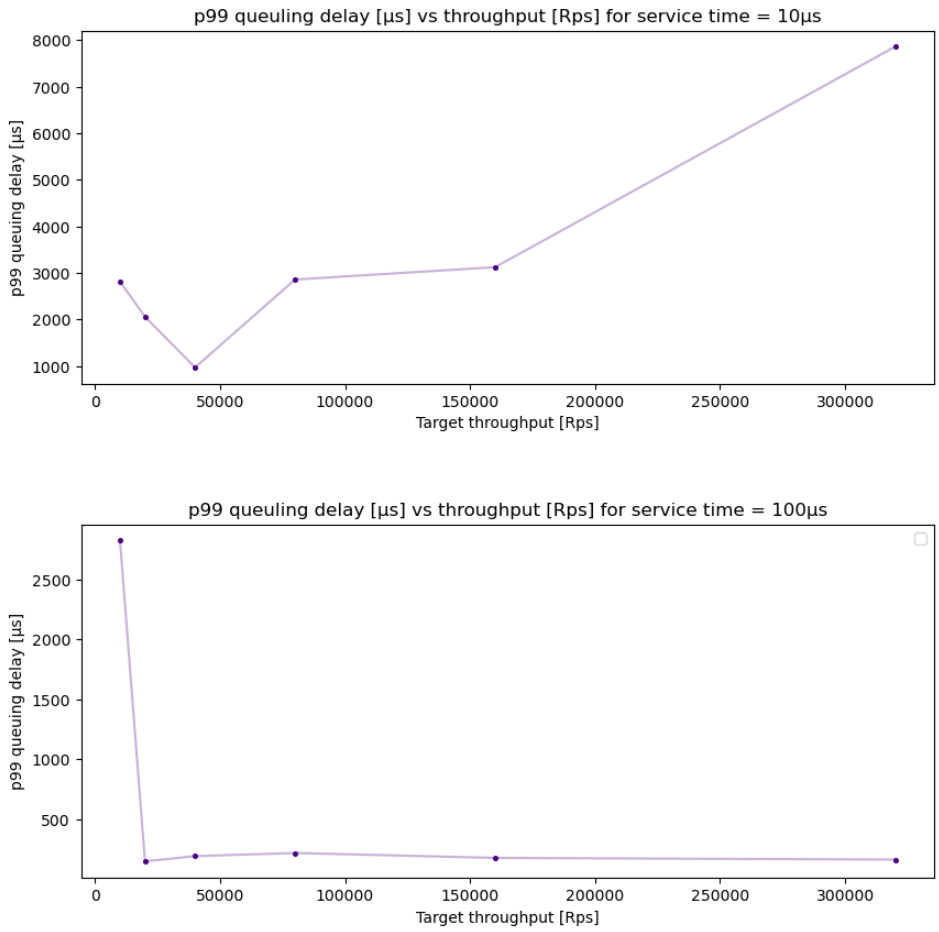

Figure 2.3: Queueing Delay for Bimodal (RRCS)

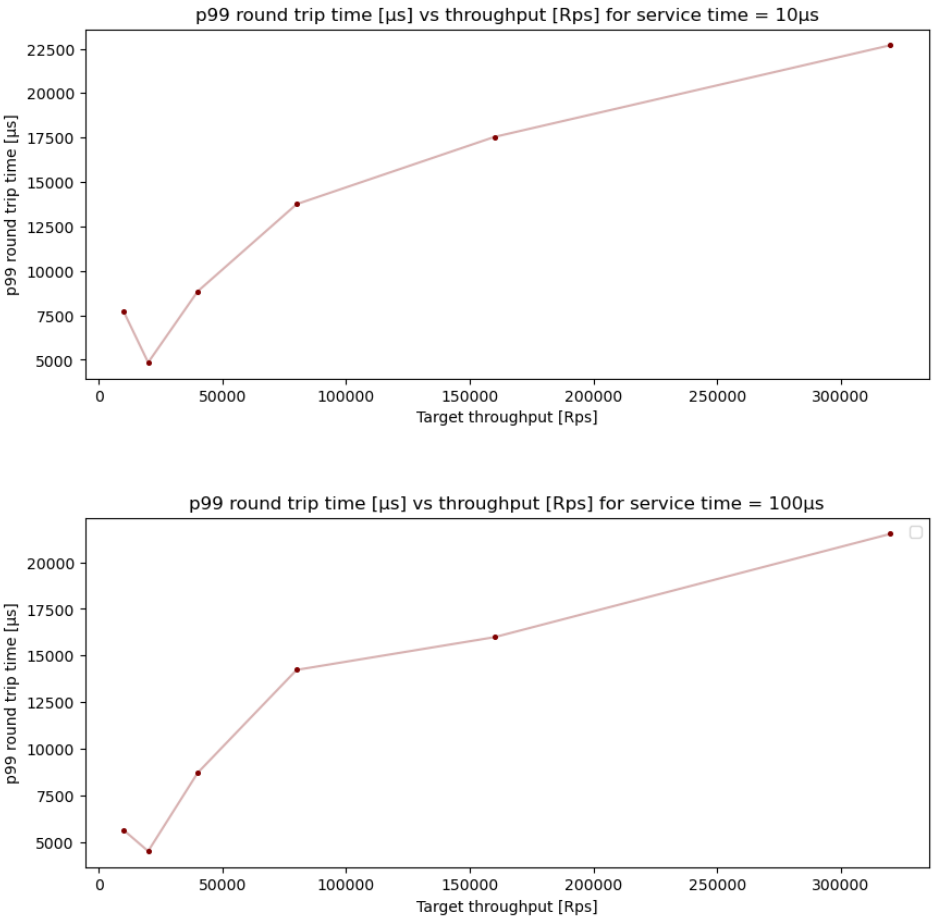

Figure 2.4: RTT for Bimodal (RRCS)

Chapter 3

Dynamic Core Allocation

3.1 Solutions

Basic idea in the main for loop:

1. Calculate the average queueing delay and convert it to microsecond unit.
2. If delay is higher than the ADD threshold, add one CPU in the bpf map by checking the maximum CPU count (8) and incrementing by 1 if available.
3. If delay is lower than the REMOVE threshold, reduce one CPU in the bpf map by checking the minimum CPU count (2) and decrementing by 1 if available.

3.2 Explanation

(CPU0 is unavailable all time. The results are acquired from 7 working cores)

1. **Evaluate and document the results you see for queueing delay and round trip time (RTT) as seen on the server and client respectively, for both unimodal and bimodal traffic.**

   Given the results in Figure 3.1 and Figure 3.2 about the queueing delay and RTT in unimodal, and Figure 3.3 and Figure 3.4 respectively for bimodal traffic, we can see that:

   The unimodal case has almost uniform queueing delay of about 750 us, and its RTT growing up to about 11000 us as the traffic load increases.

   The short services in the bimodal case has queueing delay and RTT growing up to about 2000 us and 14000 us. And the long services are less than 2500 us and 14000 us respectively.

2. **Explain the effect of the packet scheduling policy for both kinds of traffic, and provide a convincing explanation for why that happens.**

   - Both the cases have delay increasing with the traffic load. As the load grows, more tasks are pushed into the waiting queues of CPUs. So the CPU queueing delay and total

response time will definitely increase. DCA enhances the CPU resource utilization with scaling on demand, which tend to rebalance the average queueing delay to the preset threshold (50 us).

- comparison between the unimodal and bimodal cases: the 2 kinds of service in the bimodal case have longer delay than the unimodal case due to the lack of Head-of-Line blocking handling in the DCA scheduling. DCA only distributes the incoming tasks into more queues but cannot prevent the long service from positioning in the front of the short services.
- comparison between DCA and RR(CS) scheduling policy:
  - unimodal: The delay metrics of unimodal tasks are smaller in the RR and DCA than RRCS, for the reason that there are fewer CPUs working for the tasks (3 in RRCS, 7 in RR, and dynamic in DCA) which causes load imbalance between the short-service CPUs and long-service CPUs.
    DCA has slightly higher overhead than RR, due to the reason that DCA only increases the CPU number on demand instead of pre-allocating all CPUs for work in RR.
  - bimodal: The head-of-line blocking delay could be mitigated by the dynamic CPUs partially: When the long service is followed by many short services, the queueing delay of short services will increase for HoL blocking and hence enables DCA to allocate more cores to decrease the queueing delay. However, the HoL delay still exists in DCA scheduling.
    The dynamic usage number of CPUs in DCA could not rival against the performance of using all cores directly in RR. But the high CPU utilization of DCA could accelerate the workload distribution in comparison with limited number of CPUs only for short tasks in RRCS.
    Therefore, DCA performs better in short task processing than RRCS. But it is not as effective as RR in both cases and RRCS in the long task processing.

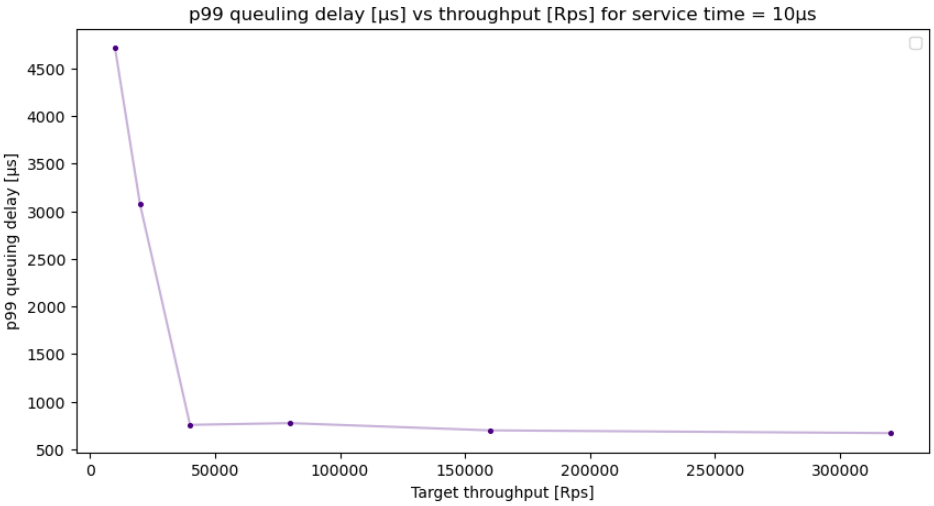

Figure 3.1: Queueing Delay for Unimodal (DCA)

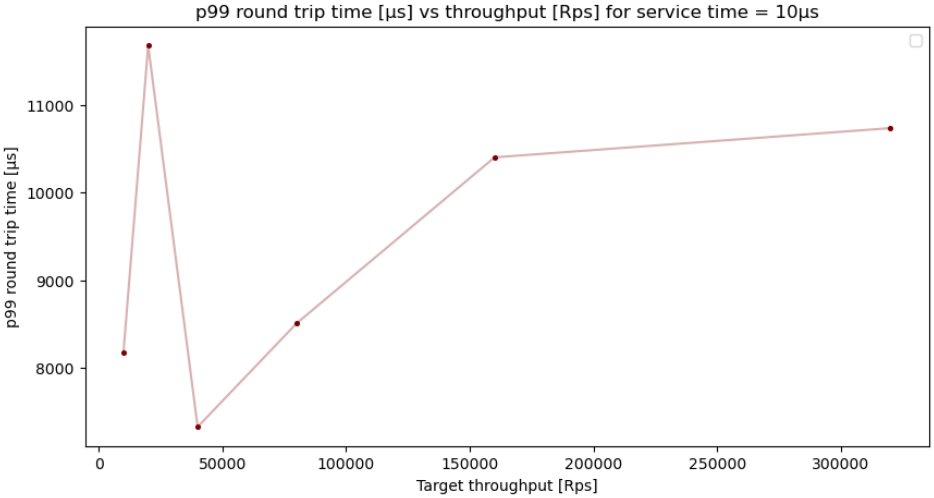

Figure 3.2: RTT for Unimodal (DCA)

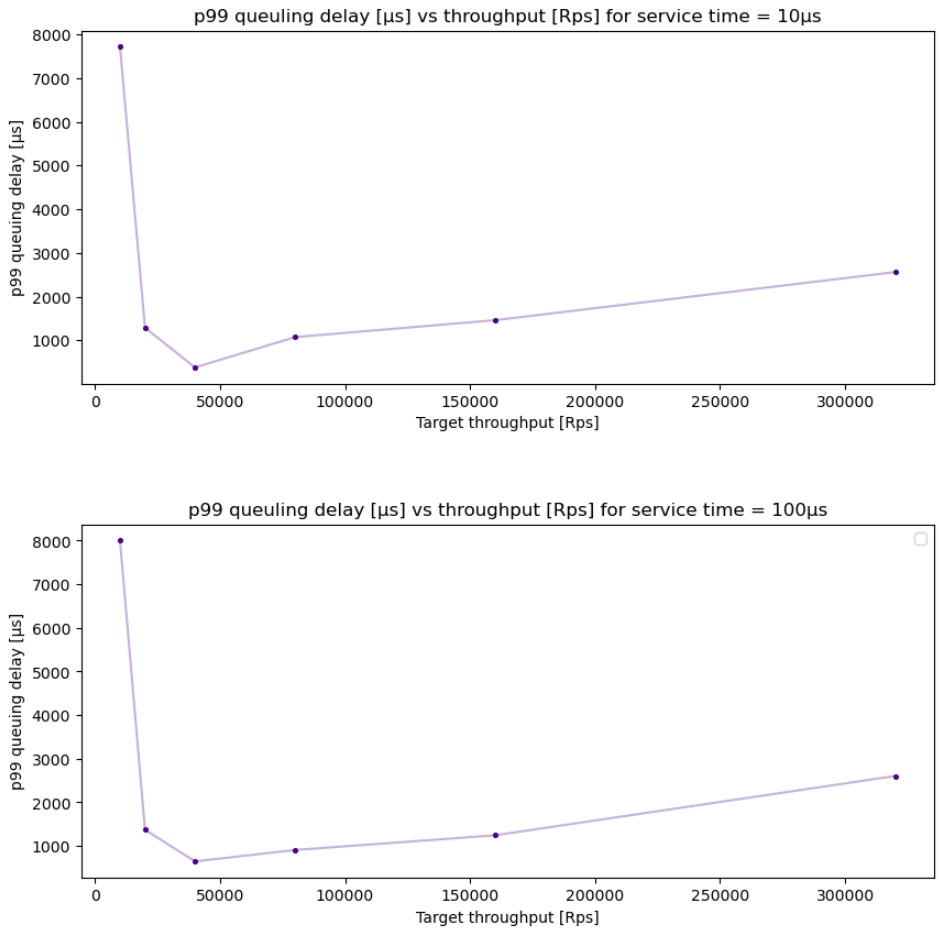

Figure 3.3: Queueing Delay for Bimodal (DCA)

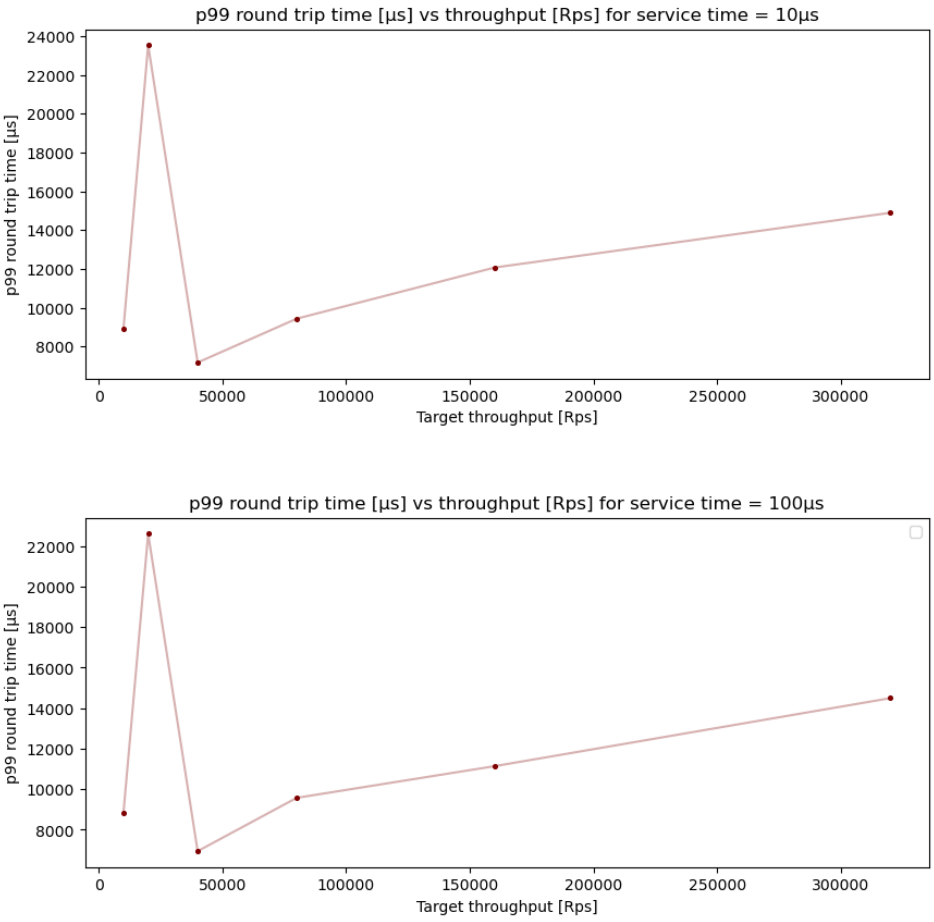

Figure 3.4: RTT for Bimodal (DCA)

Chapter 4

Conclusion

In conclusion, I learned the properties of these 3 kinds of scheduling policies in face of different traffic models:

- Round Robin: effective for unimodal traffic, low utilization of CPU resources in the light traffic, without Head-of-Line blocking handling in the bimodal case
- Round Robin Core Separated: effective for bimodal traffic for reducing the HoL blocking, bad performance in unimodal case for load imbalance
- Dynamic Core Allocation: high utilization of CPU resources, lower power consumption, limited reduction in HoL delay

To handle multi-modal traffic and enhance the CPU utilization rate, a method to combine the core separation and dynamic allocation mechanisms could be effective to use.